Improving Efficiency of Deep Learning Models

By

SOROUSH ABBASI KOOHPAYEGANI
DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

Hamed Pirsiavash, Chair

Muhao Chen

Linxi Fan

Committee in Charge

2024

Dedicated to all the teachers and mentors in my life,
who have shaped me, guided me, and helped me become who I am today.
I am forever grateful for the impact you have had on my journey

Acknowledgments

The journey of a Ph.D. is filled with uncertainty and challenges. Without the guidance of someone with vision and experience, it's easy to feel lost. It was only later in my Ph.D. that I realized this journey requires much more than intellect and technical expertise. It demands resilience, courage, and wisdom. Learning the wisdom needed to navigate this path requires a true mentor. I feel incredibly fortunate and proud to have had Dr. Hamed Pirsiavash as my advisor and to be his student.

I am deeply grateful to Hamed for his guidance, patience, and dedication. Training a deep learning model may be straightforward, but training a human being is an entirely different challenge. Unlike neural networks, students don't always follow instructions; they can be inconsistent, face doubts, and struggle with motivation. Despite these challenges, Hamed's support was a constant. His wisdom, humility, and kindness have left a lasting impact on me, shaping not only my academic journey but also my personal growth.

Words are powerful tools to compress human knowledge into language models, but they are inadequate to fully express my gratitude for Hamed's role in my life. This gratitude remains a sacred feeling, beyond what language can convey. I will forever cherish his kindness and the memories we shared—from my first crab cake in Maryland, which he treated me to, to the countless late-night deadlines where he stayed up to help with submissions, and the many moments we celebrated each accepted paper. Thank you, Hamed, for being a true teacher and mentor, guiding me through this amazing journey.

I want to extend my heartfelt thanks to all my labmates who supported me throughout this journey. A special thanks to Ajinkya, an incredible companion at the start of my Ph.D. I'll always remember the curiosity and passion you brought to our research. Sometimes one plus one is more than two, and our collaboration was one of those rare cases. I'll never forget our first accepted paper, CompRess, at NeurIPS 2020, and the excitement in your voice when I called to share the news. Thank you for helping create such memorable experiences. I'd also like to thank Navaneet. Although our personalities are very different, we worked seamlessly together, and I truly enjoyed our collaboration—especially our lunch brainstorming sessions at Tercero. Finally, I am grateful for the incredible teamwork and friendship of Vipin, Akshay, Aniruddha, Kossar, Parsa, Essam, and Om. Working alongside each of you has been both rewarding and inspiring.

UC Davis and UMBC Computer Vision Lab Members at CVPR 2022, New Orleans

I would like to express my heartfelt gratitude to Hamid Reza Vaezi Joze for hosting and supervising me during my internship at Microsoft. I also extend my thanks to Mohsen Fayyaz and Farnoush Rezaei-Jafari for their incredible collaboration on our paper, Adaptive Tokens Sampling, one of my favorite works during my Ph.D. A heartfelt thanks to Atila Orhon, Rohan Chandra, and Vignesh Jagadeesh for hosting me during my internship at Apple. That summer was one of the most memorable periods of my life. Working at Apple Park was a unique and inspiring experience, and I thoroughly enjoyed the amazing culture at Apple.

I am also grateful to Anuj Singh and Dr. Hadi Jamali-Rad from Shell AI for their invaluable collaboration and support on our recent paper, GeNIe. Additionally, I wish to thank Dr. Soheil Kolouri, Ali Abbasi, and Chayne Thrash from Vanderbilt University for our collaborations on various projects over the years.

I extend my deepest appreciation to my dissertation and qualifying committee members: Dr. Hamed Pirsiavash, Dr. Linxi (Jim) Fan, Dr. Muhao Chen, Dr. Chen-Nee Chuah, and Dr. Xin Liu. Their guidance, support, and invaluable feedback were instrumental in shaping this dissertation.

One of the most important roles of a teacher is to inspire and motivate students about the subjects they teach. I am deeply grateful to Dr. Mostafa Kamali Tabrizi for his impactful teaching of the Computer Vision class in 2017 at Sharif University of Technology. This course profoundly changed the trajectory of my life and sparked my passion for studying computer vision. I also owe much to his support during my undergraduate studies. Additionally, I would like to thank Dr. Mansour Jamzad for his encouragement and guidance throughout my undergraduate journey.

I want to acknowledge Sharif University of Technology and its incredible instructors for providing me with a strong foundation in computer science. I am equally grateful to the University of Maryland, Baltimore County, and the University of California, Davis, for accepting me as a graduate student and supporting me throughout this academic journey.

I am deeply grateful to my parents, and especially to my mother, Mina, and my grandmother, Mali, for their endless support and encouragement. Throughout my life, my mother has always placed my education above all else, often making countless sacrifices to ensure my success. As the years pass, I gain an even greater appreciation for the profound impact she has had on my life and education. Her love and dedication are the foundation of my achievements.

I would like to express my heartfelt thanks to my wife, Wenzhen, whom I had the incredible fortune of meeting at CVPR 2022 in New Orleans. She is not only an exceptional machine learning scientist but also a supportive and fun companion who believed in me, encouraged me, and stood by my side throughout my Ph.D. Also I want to thank our parents, the Chinese mama and baba she brought to my life. It is truly a joy to have them as part of our life. Sometimes, a CVPR can be more than computer vision, and it can offer something far more than a paper. Thank you, Wenzhen, and thank you, CVPR! Never underestimate CVPR!

One of my greatest goals in life is to create unforgettable memories with friends, so that as we grow older, we can look back and cherish the wonderful times we shared. During my Ph.D., I was incredibly fortunate to live with amazing friends who had a profound impact on my journey. A heartfelt thanks to Mehdi and Reza for their support and companionship. Above all, I want to express my deepest gratitude to Ebi, who was like a brother to me. I will always cherish the unforgettable nights of celebration, laughter, and music we shared. And Ebi's cooking? Absolutely extraordinary, especially his legendary geimeh-nesar.

Thank you, Ebi, your friendship was a true gift, and the memories we created together will always hold a special place in my heart.

One unique aspect of my Ph.D. journey was completing nearly three years of it remotely. Just six months after starting my Ph.D., COVID-19 became a global pandemic, forcing us to work from home. Surprisingly, I found myself more productive in this setup. Oddly enough, I feel a strange gratitude toward the COVID-19. While I'm unsure if it possesses consciousness, I've often regarded it as a unique form of distributed intelligence. If it had pushed just a little harder, my entire Ph.D. might have been remote!

Lastly, I want to acknowledge the profound role of my two best friends, Sarvi and Brian(Sarvi's Son)—my loyal and loving golden retrievers. My love for them inspired me to include their photos throughout my papers (Fig. 2.1: right is Sarvi, left is Brian; Fig. 2.13; Fig. 4.1: Sarvi; Fig. 4.6: Brian; Fig. 4.12: Brian; Fig. 5.1: Brian's 11th birthday; Fig. 6.2: right is Sarvi, left is Brian).

Photo of my dearest friend Sarvi! Rest in peace—you will always be in my heart

Sadly, Sarvi passed away recently after an incredible 17 years of life. Sarvi, you were an amazing companion, full of unconditional love and loyalty. Thank you for being part of my life and leaving behind so many cherished memories.

Stay hungry, stay foolish

— Steve Jobs, Co-founder of Apple

Contents

Abstract

Deep learning has revolutionized problem-solving by leveraging the power of deep neural networks. AlexNet and ImageNet marked a significant milestone, demonstrating the immense potential of scaling both data and computational resources to enhance model performance. This trend is particularly evident in neural language processing, where scaling Transformers has driven the development of the large language models (LLMs). Ultimately, data and computational power form the core foundations of deep learning's success.

As models grow more complex, their demand for computational resources increases, leading to higher costs and energy consumption. These rising expenses are progressively limiting machine learning research to large industry labs. For instance, while many recent studies are open-sourced, the cost of reproducing them restricts AI research for most academic institutions. To address this, developing affordable, efficient AI models that are accessible to academic labs is a crucial step toward democratizing AI research. Achieving this will require optimizing models for both data efficiency and computational resource usage.

Moreover, Accessibility, affordability, and trustworthiness are crucial factors in the development of AI models. However, many deep learning models are designed for high-end, expensive hardware, limiting their broader adoption. Additionally, reliance on centralized computing raises significant privacy concerns, as user data must be transferred to remote servers for processing, diminishes trust in AI systems. Edge computing offers a promising alternative by processing data locally on devices, making it more cost-effective, energy-efficient, and enhancing both accessibility and trust. Ideally, AI models should be efficient and optimized for edge devices, reducing dependency on centralized systems.

These motivations inspire me to explore new approaches for enhancing the efficiency of deep learning models. My research focuses on various aspects of efficiency, including data efficiency, parameter efficiency, training compute efficiency, and inference efficiency. By prioritizing efficiency, I aim to bridge the gap between cutting-edge research and deployment of these models in real-world applications, while also fostering diversity in AI development. Ultimately, my goal is to make AI inclusive and accessible to all. I believe that meaningful progress builds on the contributions of many past works, making it crucial to expand access to AI for a broader range of researchers and developers, thereby accelerating advancements in the field.

CHAPTER 1

Introduction

In recent years, a new problem-solving paradigm has emerged, leveraging deep neural networks to approximate functions that solve specific tasks (e.g., image classification). These models are trained on data using gradient descent, optimizing their parameters to fit the training set. The key underlying assumption is that the distribution of the training data closely mirrors that of the test data, allowing the model to generalize effectively during testing.

The effectiveness of deep learning has been demonstrated across a wide range of tasks, from discriminative tasks like image classification, detection, and segmentation to more recent applications in generative tasks such as image and language generation. While numerous studies have refined various components of deep learning models, the primary factors that most significantly influence a model's performance are the quantity and quality of data, the scale of model parameters, and the computational resources dedicated to training. AlexNet [249] marked a major breakthrough as the first large-scale Convolutional Neural Network, containing 60 million parameters. It demonstrated the ability to learn rich representations on the large-scale ImageNet dataset [377], which includes 1.2 million images. NVIDIA's CEO, Jensen Huang, famously referred to AlexNet as the "First Contact" [6]. Together, AlexNet and ImageNet were pivotal contributions, highlighting the potential of scaling both data and computational resources in deep learning models.

Since then, it has become evident that increasing both computational power and available data consistently leads to breakthroughs in deep learning. This trend is particularly clear in the advancements of Natural Language Processing, most notably through the scaling of Transformer models [447], which paved the way for the development of today's Large Language Models. Compute and data are the two essential pillars of intelligence in deep learning models, with increased computational resources and larger datasets consistently leading to enhanced model performance. However, computational resources continue to pose a significant bottleneck in both scaling and deployment of these models. As their complexity grows, so does their demand for computational power, necessitating advanced and expensive hardware and leading to substantial energy consumption.

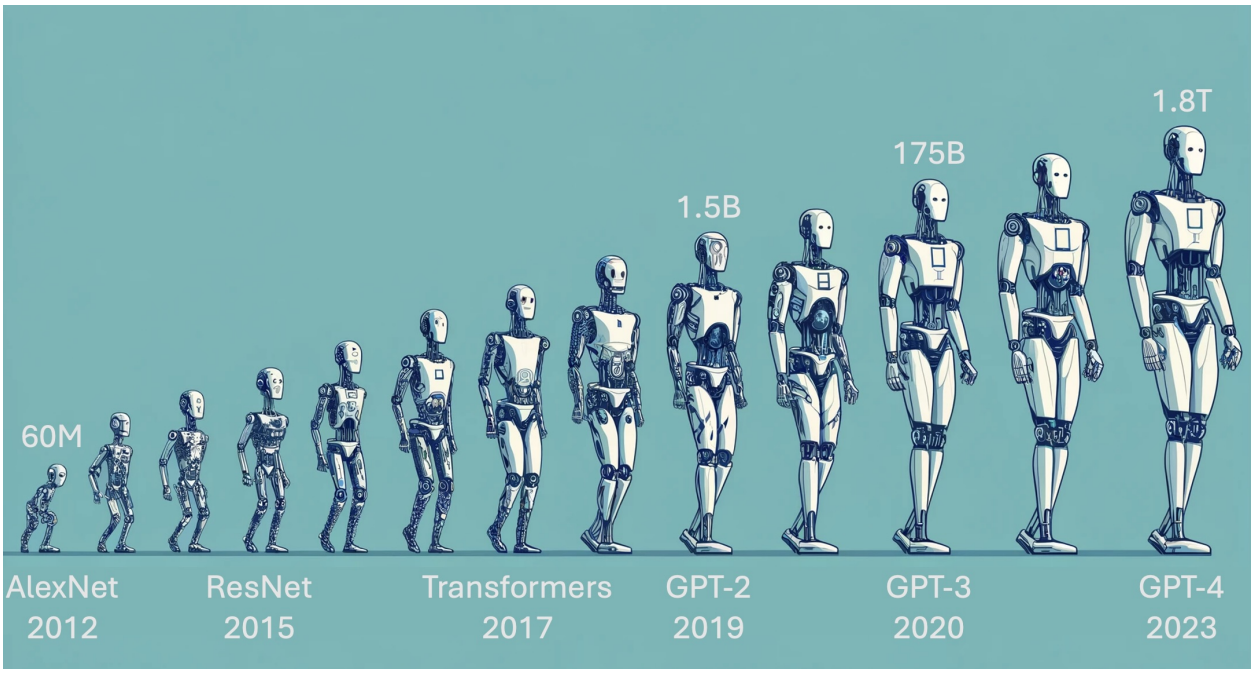


FIGURE 1.1. **The Twin Pillars of Deep Learning: Compute and Data:** Compute and data form the foundational pillars of intelligence in deep learning models. Scaling computational resources and leveraging larger datasets have consistently proven to drive significant improvements in model performance, enabling breakthroughs across a wide range of applications.

1.1 Intelligence as a New High-Demand Resource

Artificial Intelligence (AI) has emerged as one of the most transformative technologies ever created, with the potential to surpass human capabilities across a wide range of domains. However, as the demand for AI continues to grow, so does the need for energy and computational power. This demand is closely tied to advanced and expensive hardware, making computational resources both costly and scarce. The rising expense of these resources presents a significant barrier to the democratization of AI, restricting its accessibility to a small subset of individuals and industries. For example, the $20 monthly subscription fee for OpenAI's recent models illustrates how AI tools remain financially out of reach for many people globally.

AI must be affordable, accessible, and trustworthy to achieve widespread adoption and maximize its societal benefits. Making AI financially viable across diverse applications can expand its accessibility, ultimately improving quality of life for more people. However, many current deep learning models rely on high-end hardware, such as advanced GPUs, which are prohibitively expensive. This issue is exacerbated by the dominance of a few companies in the AI hardware market, driving up GPU prices and escalating computing costs for users.

Moreover, reliance on cloud services for training and inference presents additional challenges. Transferring user data to the cloud raises significant privacy concerns, undermining trust in AI systems. Dependence on centralized computing infrastructure also creates vulnerabilities, as system failures or outages can disrupt AI services and compromise reliability.

A promising solution lies in the development of efficient AI models compatible with edge devices, such as personal computers and smartphones. Edge computing processes data locally on devices rather than relying on the cloud, offering multiple advantages: it requires less expensive hardware, is more energy-efficient, and strengthens both accessibility and privacy. Ideally, these efficient models should also support training on edge devices, further reducing dependence on centralized systems. This shift would not only lower costs but also enhance data security, build trust, and create more resilient AI systems.

1.2 The Vital Role of Democratizing AI Research

Research is fundamentally about exploration. As efforts within a field grow, they naturally lead to the development of new branches and diversity of perspectives. However, in the field of AI, the high cost of computational resources significantly limits these efforts, largely confining state-of-the-art research to well-funded industry labs. For example, training Meta's LLAMA 2-70B language model required a staggering $1,720,320$ GPU hours (using NVIDIA A100 GPUs) on a dataset of 2 trillion tokens [436]. This immense investment covers the training of just one instance of the model, and in practice, multiple instances are trained to fine-tune hyperparameters for optimal performance. Although LLAMA 2 is open-source, the prohibitive cost of training makes it unfeasible for many academic labs to replicate or build upon. As a result, academia is often left behind in AI advancements, constrained by a lack of resources.

The following quote is from a recent report by Stanford University [306]: "Until 2014, most significant machine learning models were released by academia. Since then, industry has taken over. In 2022, there were 32 significant industry-produced machine learning models compared to just three produced by academia. Building state-of-the-art AI systems increasingly requires large amounts of data, computer power, and money—resources that industry actors inherently possess in greater amounts compared to nonprofits and academia.". According to Dr. Fei-Fei Li, a respected professor of computer science at Stanford University, "The public sector is now significantly lagging in resources and talent compared to that of industry" [13].

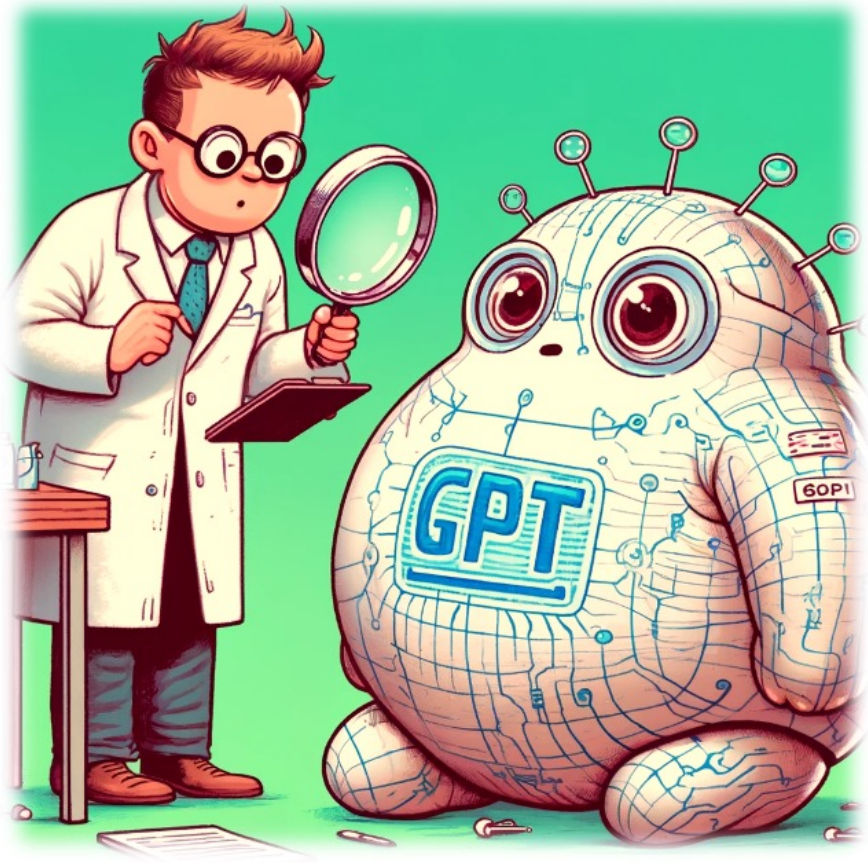


FIGURE 1.2. **High Cost of AI Research:** While some large language models are open-source, the prohibitive cost of training these massive models makes them largely inaccessible to many academic labs. This financial barrier significantly limits academia's ability to replicate or advance upon these innovations, leaving researchers constrained by limited resources and often excluded from the forefront of AI development.

She further emphasizes, "This will have profound consequences because industry is focused on developing technology that is profit-driven, whereas public-sector AI goals are focused on creating public goods."

The future of AI is largely shaped by researchers and their discoveries, making their motivation and participation pivotal in determining the direction of AI development. As AI holds the potential to become one of humanity's most transformative tools, it is crucial to engage a diverse range of researchers and perspectives to prevent the concentration of influence in the hands of a few. With the increasing demand for AI computing, reducing the overall cost of AI research may be challenging. However, a viable alternative lies in prioritizing the development of efficient AI models that are both cost-effective and reproducible for academic labs. Achieving this will require optimizing models for efficiency in both data utilization and computational resource consumption. By lowering the barriers to entry, we can make advanced AI research more accessible, inclusive, and reflective of a broader range of perspectives.

1.3 Efficiency in Deep Learning

These motivations inspire me to explore innovative methods for enhancing the efficiency of deep learning models. By prioritizing efficiency, I aim to facilitate the seamless integration of research concepts into real-world applications while promoting diversity in AI research. My ultimate goal is to make AI inclusive and accessible to all. I believe that meaningful progress is built on the foundation of numerous past contributions, making it essential to expand access to AI research across a wide range of researchers. This inclusivity will help accelerate advancements in the field. My research addresses efficiency from multiple angles, including data efficiency, parameter efficiency, training compute efficiency, inference efficiency, and the robustness of efficient models.

Efficiency at Inference:

Transformers [447], originally introduced in the field of Natural Language Processing, have become the main backbone of many deep learning models, extending their influence across various domains including image recognition [54, 111, 432] and image/video generative models [43, 338]. Tokens are unit of data in transformers. For instance, ViT [111] demonstrates how images can be converted into smaller patches, with each patch serving as a token in the transformer architecture. Due to widespread adoption of transformers in various domains, the concept of tokens has become widely recognized as a fundamental unit of data. Notably, LLAMA [436], a recent family of Large Language Models (LLMs), quantify the size of training data in terms of tokens, while NVIDIA employs tokens per second as a performance metric to compare their latest GPU models such as H200 and H100 [10]. Apple has recognized the importance of transformers and has optimized their efficiency for deployment on the Apple Neural Engine (ANE) [4]. Given the significance of transformers, we address two efficiency limitations in vision transformers:

ATS: Adaptive Token Sampling in vision transformers. [126] Adaptive Token Sampling (ATS) leverages the insight that not all tokens in an input image are equally important for processing. By assigning significance scores to tokens at each transformer layer, ATS dynamically drops less important tokens, such as background tokens, during the forward pass. One can replace the standard attention layer in pre-trained vision transformers with the ATS attention layer without requiring additional training. ATS achieves comparable performance to traditional transformers while significantly reducing computational costs, with an average reduction of 37% in GFLOPS.

SimA: Simple Softmax-free Attention for Vision Transformers. [236] Deploying transformers in various applications can be computationally expensive, partly due to the Softmax layer in the attention block. To address this challenge, we propose SimA, a Softmax-free attention block that normalizes query and key matrices using a simple ℓ_1-norm instead of Softmax. This simplifies the attention mechanism to a straightforward multiplication of three matrices. SimA has the capability to dynamically adjust the computation order at test time, enabling linear computation scaling with respect to the number of tokens or the number of channels. We demonstrate that integrating SimA into state-of-the-art transformer variants, including DeiT, XCiT, and CvT, yields comparable accuracy to existing models without the need for a Softmax layer.

CompRess: Self-Supervised Learning by Compressing Representations [239]. An alternative approach to achieving faster inference times is by deploying a lightweight model with fewer parameters. However, smaller models often suffer from sub-optimal optimization due to their limited parameter count. This issue becomes particularly pronounced in Self-Supervised contrastive learning tasks, where lighter models, such as MobileNet, exhibit significantly lower performance compared to larger counterparts like ResNet50. To address this limitation, we propose a novel approach for self-supervised learning with lighter models. Initially, we train a high-capacity model to learn a rich representation. Subsequently, we leverage this larger model as a teacher and employ our self-supervised distillation framework to transfer its knowledge to the lighter student model. This method enhances the accuracy of smaller models by at least 20%.

Moreover, in our work SimReg [319], we enhance the CompRess method by introducing regression as a simple distillation loss. The core concept involves employing a predictor layer (MLP layer) atop the student model to predict the teacher's representation. Later, we discard the predictor layer during the transfer learning phase. SimReg emerges as a straightforward yet effective self-supervised distillation technique, achieving state-of-the-art performance.

Model Parameters Efficiency:

The performance of deep learning models scales up with larger models and datasets. However, a significant challenge for large-scale deep learning models is the storage of model parameters and intermediate activations, especially in edge devices with limited GPU memory. It's important to note that the required memory is considerably higher during training due to the storage of gradients and states of the optimizer.

To highlight the importance of memory, let's compare the advancements in GPUs over the last 14 years. AlexNet [249], for instance, was trained on two NVIDIA GTX 580 GPUs, each with 3GB of memory. It's noteworthy that the GTX 580 featured 3GB of memory and 1.58 TFLOPS of FP32 computing power, whereas the RTX 4090 showcases a substantial upgrade with 24GB of memory and a staggering 82.58 TFLOPS of FP32 computing power. This indicates a 52 times increase in compute power, while memory capacity only improved by a factor of 8. Thus, we observe that the scaling of memory in these GPUs is challenging, as advancements occurring slower than those in compute power. Moreover, memory becomes an even more significant issue in edge devices with limited memory capacity. For example, the NVIDIA Jetson Nano is equipped with only 4GB of memory [8]. Consequently, the deployment of large deep learning models poses a considerable challenge in such devices.

In PRANC [322], we introduce a novel reparametrization technique for deep learning models that significantly reduces the number of parameters. We demonstrate this reparametrization by expressing a deep model as a linear combination of multiple randomly initialized and frozen weights, referred to as 'basis' networks. The model can be reconstructed using a single scalar 'seed', which is used to generate the pseudo-random 'basis' networks, along with the learned linear mixture coefficients. PRANC effectively addresses the challenge of efficiently storing and communicating deep models, thereby alleviating common bottlenecks in scenarios such as multi-agent learning, continual learners, federated systems, and edge devices.

With the rise in popularity of Large Language Models (LLMs), we anticipate the emergence of numerous variations of LLMs, tailored to specific tasks. For instance, OpenAI has introduced the GPT Store [5], allowing users to fine-tune GPT on their own data. It's conceivable that in the near future, there will be a personalized GPT model for each individual, customized for specific tasks or preferences. However, fine-tuning LLMs with billions of parameters poses feasibility challenges on GPUs with limited memory capacity. Moreover, we aim to keep multiple variations of a single Large Language Model (LLM) in GPU memory, enabling rapid switching between different variations based on user queries. To address this challenge, we propose a novel method for parameter-efficient fine-tuning:

NOLA: Compressing LoRA using Linear Combination of Random Basis [235]. Fine-tuning all parameters of LLMs and maintaining unique models for each downstream task or domain becomes impractical due to the immense size of checkpoints, such as the 350GB size of GPT-3. Current literature, such

as LoRA, highlights the potential of low-rank modifications to the original weights of an LLM, facilitating efficient adaptation and storage for task-specific models. These methods significantly reduce the number of parameters required for fine-tuning an LLM. However, they encounter two primary limitations: firstly, the parameter reduction is lower-bounded by the rank one decomposition, and secondly, the degree of reduction is heavily influenced by both the model architecture and the selected rank. For instance, in larger models, even a rank one decomposition might surpass the actual number of parameters needed for adaptation. In this paper, we introduce NOLA, which overcomes the limitations of LoRA. NOLA achieves this by re-parameterizing the low-rank matrices in LoRA using linear combinations of randomly generated matrices (basis) and optimizing the linear mixture coefficients only. This approach enables us to disentangle the number of trainable parameters from both the choice of rank and the network architecture. We present adaptation results using LLAMA 2 and vision transformers in natural language and computer vision tasks. NOLA performs comparably to, or even better than, models with equivalent parameter counts. Furthermore, we demonstrate that we can fine-tune LLAMA 2 70-B with fewer than $0.6M$ parameters without sacrificing performance compared to LoRA with rank one.

The pursuit of training efficient parameter models extends beyond deep learning models. For instance, 3D Gaussian Splatting is a new technique for novel view synthesis where properties of 3D Gaussians [228] (location, shape, color) are optimized to model a 3D scene. The method performs better than SOTA NeRF approaches, is extremely fast to train and can be rendered in real time during inference. In NeRF, the geometry is implicitly modeled within a multi-layer perceptron network (MLP), resulting in a constant-size model regardless of the geometry complexity. However, unlike NeRFs, the memory requirement of 3D Gaussians increases with the complexity of the scenes and number of Gaussians.

In CompGS [318], we substantially reduce the size of trained 3D Gaussian Splat models by 10-20 times through vector quantization of the Gaussian parameters. This involves employing K-Means quantization on the covariance and color parameters of all Gaussians, replacing each value with the corresponding entry in the codebook (i.e., the cluster center). Importantly, this quantization process is seamlessly integrated with the training of parameter values. Remarkably, we find that the models can be compressed by up to 20 times without a significant drop in performance.

Training Compute Efficiency:

The annotation process in many applications may be costly, prone to biases, ambiguous, or raise privacy concerns. Self-supervised learning (SSL) algorithms tackle these challenges by learning rich representations directly from unlabeled images or videos. These representations can then be used alongside limited annotated data to develop accurate visual recognition models. In SSL, data preparation is typically cheaper since there is no need for annotation. Therefore, training self-supervised models is usually more cost-effective, as data can be gathered without the expense of annotation.

A major drawback of recent contrastive self-supervised learning methods is their training inefficiency compared to supervised learning. For instance, training MoCo requires 800 epochs to achieve a rich representation comparable to its supervised counterpart, which typically requires only 100 epochs. This can restrict research on self-supervised methods to large industry labs. To mitigate training inefficiency, we address the limitations of recent contrastive learning by introducing three novel self-supervised learning methods:

ISD: Self-Supervised Learning by Iterative Similarity Distillation [421]. In contrastive learning, the aim is to pull the representations of two views (augmentations) of an image closer together (positives), while simultaneously pushing them away from other random images (negatives). However, without labels, some negatives may inadvertently share the same label as the positives, resulting in false negatives. ISD [421] addresses this issue by relaxing the hard positive/negative classification to a soft classification, assigning probabilities to the negatives. Interestingly, we observed that maintaining a moving average of the model throughout training, effectively creating a temporal ensemble of the model, yields superior performance compared to the original model. Motivated by this insight, we leverage the moving average of the model as the teacher to compute soft probability distributions for the negative images. Consequently, we iteratively distill the knowledge from the teacher (temporal ensemble) to the student (original model). This iterative distillation process enhances the model's ability to discriminate between positive and negative samples, ultimately improving its performance. By mitigating the false negatives error inherent in contrastive learning, ISD [421] accelerates the convergence of the learning process. Our experiments demonstrate that ISD surpasses the performance of the 800-epoch MoCo V2 [76] model with just 200 epochs of training.

MSF: Mean Shift for Self-Supervised Learning [241]. BYOL introduces a simplified framework for self-supervised learning by eliminating the need for negative samples in contrastive learning. The loss function simply pulls the query image towards its corresponding target image (positive sample), where the target image is another augmentation of the query image. However, due to the high degree of augmentation, using a single positive target view can introduce noise into the learning process. We observed that the nearest neighbors of the target view are often semantically similar to the query image during training. Therefore, we maintain a memory bank of random images and expand the positive set to include not only the target but also its nearest neighbors. The key idea behind this approach is that the average of nearest neighbors provides a more robust target compared to a single target image. Our method effectively pulls the representation of one view of an image closer to the representations of its other view and its nearest neighbors. Through our experiments, we demonstrate that with only 200 epochs of training, our method outperforms BYOL trained for 800 epochs. This underscores the effectiveness and efficiency of our approach in self-supervised learning tasks.

CMSF: Constrained Mean Shift Using Distant Yet Related Neighbors [320]. We further enhance MSF by restricting the nearest neighbor search to an external source of knowledge. The concept involves identifying distant samples with semantically similar images. By incorporating additional sources of knowledge to constrain the memory bank, we expand the applicability of MSF to semi-supervised and self-supervised learning scenarios. CMSF represents a generalized version of MSF, surpassing its predecessor in self-supervised learning tasks.

Robustness of Efficient Models:

The reliability of deep learning models is a crucial factor for their adoption in various critical applications. Extensive research has been conducted on the robustness of deep learning models against different adversarial attack scenarios. For instance, in [407], we investigate the robustness of Vision Transformers to Backdoor Attacks. However, most attack threat models primarily focus on compromising the accuracy of deep learning models. An often overlooked yet intriguing attack scenario involves designing attacks to target the efficiency of deep learning models.

In essence, attackers may devise strategies to increase the deployment cost or energy consumption of deep learning models. For example, in self-supervised learning methods, it's commonly assumed that data collection is inexpensive due to the absence of the need for meticulous inspection and annotation. However, in [382], we demonstrate that self-supervised methods are vulnerable to backdoor attacks. Here, attackers inject a trigger (an image patch chosen by the attacker) into a small portion of the unlabeled data. Despite the model performing well on clean test images, the attacker can manipulate the model's decision by presenting the trigger during test time.

One potential solution to mitigate this vulnerability is to conduct a comprehensive inspection of large-scale unlabeled data. However, this strategy may come at a significant cost and could compromise the efficiency of self-supervised learning, particularly in terms of data collection.

In the preceding discussion, we explored how efficient transformers can dynamically adapt the number of tokens during inference for each image. Given that the computational load of these models depends on their input, the following study investigates whether attackers can manipulate the input image to compromise the compute and energy efficiency of efficient transformers.

SlowFormer: Adversarial Attack on Compute of Efficient Vision Transformers [317]. Efficient transformers are designed to minimize computational requirements and power usage in deep learning models. However, our research exposes their vulnerability to universal adversarial patch attacks. In these attacks, adversaries optimize a patch that, when overlaid on any image, substantially increases compute and power usage. Our experiments, conducted across three different efficient vision transformer methods, unveil scenarios where attackers can significantly amplify computation by applying a patch covering just 8% of the image area. Additionally, we demonstrate that employing a standard adversarial training defense method can partially mitigate the success of these attacks.

Training Data Efficiency:

Recent vision models demonstrate impressive zero- and few-shot performance across a range of tasks. However, achieving further improvement in performance on downstream tasks necessitates careful curation of task-specific data. It is also vital for the training data to encompass the rare and challenging scenarios encountered in these downstream tasks.

For example, let's consider the scenario of self-driving tasks. Unexpected events like sudden pedestrian crossings present significant challenges. To handle these effectively, the model needs training on similar instances to learn appropriate responses. However, collecting data for these rare cases is inherently difficult due to their scarcity in real-world environments. Traditional approaches, such as gathering and annotating random data followed by fine-tuning, are inefficient for improving model performance on failure cases. Hence, we are interested in efficient data collection approach aimed at mitigating the model's failure modes.

It's worth mentioning that the literature on Few-shot Learning has proposed an alternative solution: training a model that generalizes well even with scarce training data (e.g., 5-shot scenarios). For instance, in Contrastive Grad-CAM Consistency [345], we enforce two views (crops) of an image to have similar Grad-CAM [390] explanations. We demonstrate that applying this form of consistency in explanation serves as effective regularization for the model in few-shot learning.

An alternative solution lies in leveraging generative models to generate additional data. Multi-Modal Language Models (MMLMs) excel in reasoning, while advanced image and video generative models produce realistic samples. A promising approach involves combining the generative power of these models with the reasoning capabilities of MMLMs to generate and annotate data specifically designed to address failure cases. This enables us to efficiently augment datasets using generative models, effectively converting computational resources into high-quality, task-specific data. Inspired by this, we introduce an efficient methodology for leveraging generative models to generate difficult samples for classification tasks.

GeNIe: Generative Hard Negative Images Through Diffusion [238]. We introduce GeNIe, a novel augmentation technique that harnesses a latent diffusion model conditioned on a text prompt to blend contrasting data points—an image from the source category and a text prompt from the target category—to generate challenging samples. Inspired by recent advancements in diffusion-based image editing, we control the number of diffusion iterations to preserve low-level and background features from the source image while representing the target category, thereby creating a hard negative sample for the source category. Additionally, we enhance our approach by dynamically adapting the noise level for each image (GeNIe-Ada), resulting in improved performance. Through extensive experiments conducted in both few-shot and long-tail distribution scenarios, we showcase the effectiveness and superior performance of our novel augmentation method compared to existing techniques.

1.4 Contributions

In this dissertation, I explored the efficiency of deep learning models from various perspectives, including training compute efficiency, inference compute efficiency, training data efficiency, parameter efficiency, and the robustness of efficient models. Below is a summary of the key contributions for each chapter:

- Chapter 2.1: A novel adaptive token sampling method was proposed to enhance the inference efficiency of vision transformers by dynamically selecting the most informative tokens for each task and input image.
- Chapter 2.2: Introduced a simple and innovative Softmax-free attention block, enabling dynamic computation of QKV matrix multiplication based on the number of input tokens.
- Chapter 2.3: Developed a novel recipe for training smaller, lightweight self-supervised deep learning models (e.g., MobileNet) and introduced a new self-supervised knowledge distillation loss.
- Chapter 3: Proposed a novel parameter-efficient fine-tuning (PEFT) method that is more compact than LoRA models. Unlike other PEFT methods, this approach offers flexibility in the number of fine-tuned parameters by decoupling them from the architecture choice or the rank of LoRA.
- Chapter 4.1: Introduced a relaxed self-supervised learning framework inspired by contrastive learning. This approach replaces strict binary negatives with soft negatives, addressing the issue of false negatives in contrastive loss.
- Chapter 4.2: Proposed a novel self-supervised learning method inspired by the mean-shift algorithm. This method generalizes BYOL by including additional positive samples through the nearest neighbors of the target samples in the BYOL objective function.
- Chapter 4.3: Further refined the mean-shift-based objective from Chapter 4.2 by clustering faraway but semantically relevant images. This was achieved by constraining the nearest neighbor search space using auxiliary knowledge. Demonstrated that this approach extends easily to semi-supervised and fully supervised settings, improving robustness to label noise.
- Chapter 5: Investigated the robustness of efficient transformers, revealing that dynamic compute mechanisms may be vulnerable to universal adversarial patch attacks that target their efficiency and energy consumption. Proposed a simple adversarial training to mitigate these vulnerabilities.
- Chapter 6: Developed a novel data augmentation method using diffusion models to generate hard negative samples. This approach combines two sources of information: the label of the target class and the image of a source class, creating challenging scenarios for improved model performance.

CHAPTER 2

Compute Efficiency at Inference

2.1 ATS: Adaptive Token Sampling For Vision Transformers

While state-of-the-art vision transformer models achieve promising results in image classification, they are computationally expensive and require many GFLOPs. Although the GFLOPs of a vision transformer can be decreased by reducing the number of tokens in the network, there is no setting that is optimal for all input images. In this work, we therefore introduce a differentiable parameter-free Adaptive Token Sampler (ATS) module, which can be plugged into any existing vision transformer architecture. ATS empowers vision transformers by scoring and adaptively sampling significant tokens. As a result, the number of tokens is not constant anymore and varies for each input image. By integrating ATS as an additional layer within the current transformer blocks, we can convert them into much more efficient vision transformers with an adaptive number of tokens. Since ATS is a parameter-free module, it can be added to the off-the-shelf pre-trained vision transformers as a plug and play module, thus reducing their GFLOPs without any additional training. Moreover, due to its differentiable design, one can also train a vision transformer equipped with ATS. We evaluate the efficiency of our module in both image and video classification tasks by adding it to multiple SOTA vision transformers. Our proposed module improves the SOTA by reducing their computational costs (GFLOPs) by 2×, while preserving their accuracy on the ImageNet, Kinetics-400, and Kinetics-600 datasets. The code is available at https://adaptivetokensampling.github.io/.

2.1.1 Introduction

Over the last ten years, there has been a tremendous progress on image and video understanding in the light of new and complex deep learning architectures, which are based on the variants of 2D [177, 246, 398] and 3D [104, 106, 131, 132, 439, 441] Convolutional Neural Networks (CNNs). Recently, vision transformers have shown promising results in image classification [112, 219, 430, 476] and action recognition [38, 48, 288] compared to CNNs. Although vision transformers have a superior representation power, the high computational cost of their transformer blocks make them unsuitable for many edge devices. The computational cost of a vision transformer grows quadratically with respect to the number of tokens it uses.

To reduce the number of tokens and thus the computational cost of a vision transformer, DynamicViT [359] proposes a token scoring neural network to predict which tokens are redundant. The approach then keeps a fixed ratio of tokens at each stage. Although DynamicViT reduces the GFLOPs of a given network, its scoring network introduces an additional computational overhead. Furthermore, the scoring network needs to be trained together with the vision transformer and it requires to modify the loss function by adding additional loss terms and hyper-parameters. To alleviate such limitations, EViT [274] employs the attention weights as the tokens' importance scores. A further limitation of both EViT and DynamicViT is that they need to be re-trained if the fixed target ratios need to be changed (*e.g.* due to deployment on a different device). This strongly limits their applications.

In this work, we propose a method to efficiently reduce the number of tokens in any given vision transformer without the mentioned limitations. Our approach is motivated by the observation that in image/action classification, all parts of an input image/video do not contribute equally to the final classification scores and some parts contain irrelevant or redundant information. The amount of relevant information varies depending on the content of an image or video. For instance, in Fig. 2.7, we can observe examples in which only a few or many patches are required for correct classification. The same holds for the number of tokens used at each stage, as illustrated in Fig. 2.2. Therefore, we propose an approach that automatically selects an adequate number of tokens at each stage based on the image content, *i.e.* the number of the selected tokens at all network's stages varies for different images, as shown in Fig. 2.6. It is in contrast to [274, 359], where the ratio of the selected tokens needs to be specified for each stage and is constant after training. However, selecting a static number of tokens will on the one hand discard important information for challenging images/videos, which leads to a classification accuracy drop. On the other hand, it will use more tokens than necessary for the easy cases and thus waste computational resources. In this work, we address the question of how a transformer can dynamically adapt its computational resources in a way that not more resources than necessary are used for each input image/video.

To this end, we introduce a novel *Adaptive Token Sampler (ATS)* module. ATS is a differentiable parameter-free module that adaptively down-samples input tokens. To do so, we first assign significance scores to the input tokens by employing the attention weights of the classification token in the self-attention layer and then select a subset of tokens using inverse transform sampling over the scores. Finally, we softly down-sample the output tokens to remove redundant information with the least amount of information loss. In contrast to [359], our approach does not add any additional learnable parameters to the network. While

the ATS module can be added to any off-the-shelf pre-trained vision transformer without any further training, the network equipped with the differentiable ATS module can also be further fine-tuned. Moreover, one may train a model only once and then adjust a maximum limit for the ATS module to adapt it to the resources of different edge devices at the inference time. This eliminates the need of training separate models for different levels of computational resources.

We demonstrate the efficiency of our proposed adaptive token sampler for image classification by integrating it into the current state-of-the-art vision transformers such as DeiT [431], CvT [476], and PS-ViT [511]. As shown in Fig. 2.4, our approach significantly reduces the GFLOPs of vision transformers of various sizes without significant loss of accuracy. We evaluate the effectiveness of our method by comparing it with other methods designed for reducing the number of tokens, including DynamicViT [359], EViT [274], and Hierarchical Pooling [330]. Extensive experiments on the ImageNet dataset show that our method outperforms existing approaches and provides the best trade-off between computational cost and classification accuracy. We also demonstrate the efficiency of our proposed module for action recognition by adding it to the state-of-the-art video vision transformers such as XViT [48] and TimeSformer [38]. Extensive experiments on the Kinetics-400 and Kinetics-600 datasets show that our method surpasses the performance of existing approaches and leads to the best computational cost/accuracy trade-off. In a nutshell, the adaptive token sampler can significantly scale down the off-the-shelf vision transformers' computational costs and it is therefore very useful for real-world vision-based applications.

2.1.2 Related Work

The transformer architecture, which was initially introduced in the NLP community [447], has demonstrated promising performance on various computer vision tasks [55,80,112,289,361,430,506,534,536,540]. ViT [112] follows the standard transformer architecture to tailor a network that is applicable to images. It splits an input image into a set of non-overlapping patches and produces patch embeddings of lower dimensionality. The network then adds positional embeddings to the patch embeddings and passes them through a number of transformer blocks. An extra learnable class embedding is also added to the patch embeddings to perform classification. Although ViT has shown promising results in image classification, it requires an extensive amount of data to generalize well. DeiT [430] addressed this issue by introducing a distillation token designed to learn from a teacher network. Additionally, it surpassed the performance of ViT. LV-ViT [219] proposed a new objective function for training vision transformers and achieved better

performance. TimeSformer [38] proposed a new architecture for video understanding by extending the self-attention mechanism of the standard transformer models to video. The complexity of the TimeSformer's self-attention is $O(T^2S + TS^2)$ where T and S represent temporal and spatial locations respectively. X-ViT [48] reduced this complexity to $O(TS^2)$ by proposing an efficient video transformer.

Besides the accuracy of neural networks, their efficiency plays an important role in deploying them on edge devices. A wide range of techniques have been proposed to speed up the inference of these models. To obtain deep networks that can be deployed on different edge devices, works like [415] proposed more efficient architectures by carefully scaling the depth, width, and resolution of a baseline network based on different resource constraints. [194] aims to meet such resource requirements by introducing hyper-parameters, which can be tuned to build efficient light-weight models. The works [152, 459] have adopted quantization techniques to compress and accelerate deep models. Besides quantization techniques, other approaches such as channel pruning [181], run-time neural pruning [358], low-rank matrix decomposition [211, 505], and knowledge distillation [188, 283] have been used as well to speed up deep networks.

In addition to the works that aim to accelerate the inference of convolutional neural networks, other works aim to improve the efficiency of transformer-based models. In the NLP area, Star-Transformer [164] reduced the number of connections from n^2 to $2n$ by changing the fully-connected topology into a star-shaped structure. TinyBERT [220] improved the network's efficiency by distilling the knowledge of a large teacher BERT into a tiny student network. PoWER-BERT [156] reduced the inference time of the BERT model by identifying and removing redundant and less-informative tokens based on their importance scores estimated from the self-attention weights of the transformer blocks. To reduce the number of FLOPs in character-level language modeling, a new self-attention mechanism with adaptive attention span is proposed in [409]. To enable fast performance in unbatched decoding and improve the scalability of the standard transformers, Scaling Transformers [214] are introduced. These novel transformer architectures are equipped with sparse variants of standard transformer layers.

To improve the efficiency of vision transformers, sparse factorization of the dense attention matrix has been proposed [83], which reduces its complexity to $O(n\sqrt{n})$ for the autoregressive image generation task. [374] tackled this problem by proposing an approach to sparsify the attention matrix. They first cluster all the keys and queries and only consider the similarities of the keys and queries that belong to the same cluster. DynamicViT [359] proposed an additional prediction module that predicts the importance of tokens and discards uninformative tokens for the image classification task. Hierarchical Visual Transformer

(HVT) [330] employs token pooling, which is similar to feature map down-sampling in convolutional neural networks, to remove redundant tokens. PS-ViT [511] incorporates a progressive sampling module that iteratively learns to sample distinctive input tokens instead of uniformly sampling input tokens from all over the image. The sampled tokens are then fed into a vision transformer module with fewer transformer encoder layers compared to ViT. TokenLearner [379] introduces a learnable tokenization module that can reduce the computational cost by learning few important tokens conditioned on the input. They have demonstrated that their approach can be applied to both image and video understanding tasks. Token Pooling [305] down-samples tokens by grouping them into a set of clusters and returning the cluster centers. A concurrent work [274] introduces a token reorganization method that first identifies top-k important tokens by computing token attentiveness between the tokens and the classification token and then fuses less informative tokens. IA-RED^2 [328] proposes an interpretability-aware redundancy reduction framework for vision transformers that discards less informative patches in the input data. Most of the mentioned approaches improve the efficiency of vision transformers by introducing architectural changes to the original models or by adding modules that add extra learnable parameters to the networks, while our parameter-free adaptive module can be incorporated into off-the-shelf architectures and reduces their computational complexity without significant accuracy drop and even without requiring any further training.

2.1.3 Adaptive Token Sampler

State-of-the-art vision transformers are computationally expensive since their computational costs grow quadratically with respect to the number of tokens, which is static at all stages of the network and corresponds to the number of input patches. Convolutional neural networks deal with the computational cost by reducing the resolution within the network using various pooling operations. It means that the spatial or temporal resolution decreases at the later stages of the network. However, applying such simple strategies, *i.e*. pooling operations with fixed kernels, to vision transformers is not straightforward since the tokens are permutation invariant. Moreover, such static down-sampling approaches are not optimal. On the one hand, a fixed down-sampling method discards important information at some locations of the image or video, like details of the object. On the other hand, it still includes many redundant features that do not contribute to the classification accuracy, for instance, when dealing with an image with a homogeneous background. Therefore, we propose an approach that dynamically adapts the number of tokens at each stage of the network based on the input data such that important information is not discarded and no computational resources are wasted for processing redundant information.

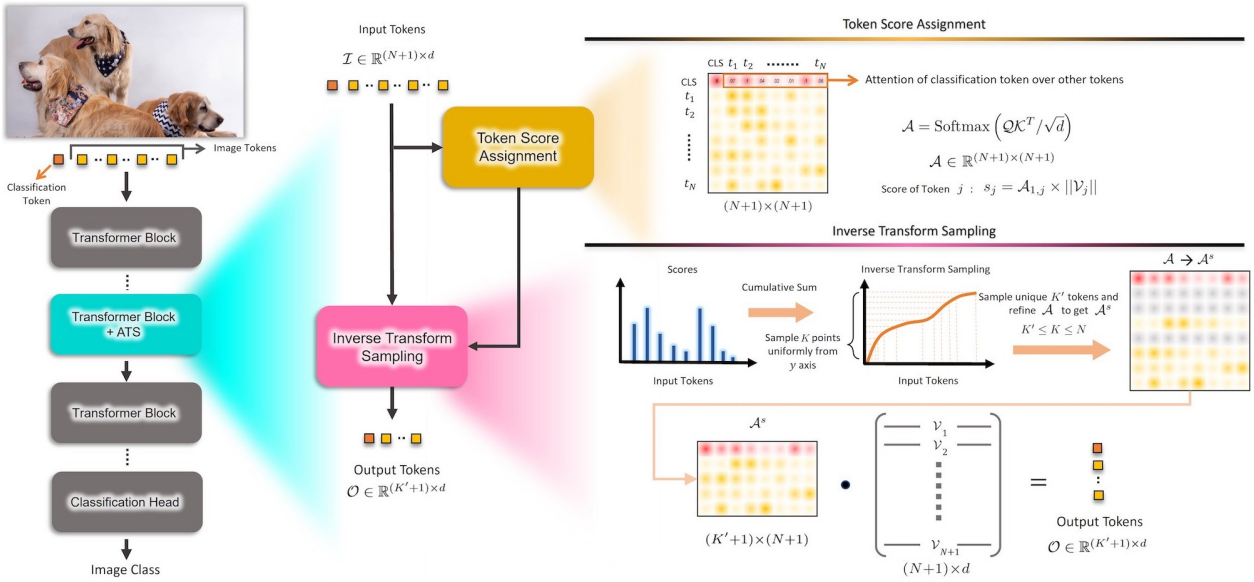


FIGURE 2.1. The Adaptive Token Sampler (ATS) can be integrated into the self-attention layer of any transformer block of a vision transformer model (left). The ATS module takes at each stage a set of input tokens I. The first token is considered as the classification token in each block of the vision transformer. The attention matrix A is then calculated by the dot product of the queries Q and keys K, scaled by $\sqrt{d}$. We use the attention weights $A_{1,2}, \ldots, A_{1,N+1}$ of the classification token as significance scores $S \in \mathbb{R}^N$ for pruning the attention matrix A. To reflect the effect of values V on the output tokens O, we multiply the $A_{1,j}$ by the magnitude of the corresponding value V_j. We select the significant tokens using inverse transform sampling over the cumulative distribution function of the scores S. Having selected the significant tokens, we then sample the corresponding attention weights (rows of the attention matrix A) to get A^s. Finally, we softly downsample the input tokens I to output tokens O using the dot product of A^s and V.

To this end, we propose our novel Adaptive Token Sampler (ATS) module. ATS is a parameter-free differentiable module to sample significant tokens over the input tokens. In our ATS module, we first assign significance scores to the N input tokens and then select a subset of these tokens based on their scores. The upper bound of GFLOPs can be set by defining a maximum limit for the number of tokens sampled, denoted by K. Since the sampling procedure can sample some input tokens several times, we only keep one instance of a token. The number of sampled tokens K' is thus usually lower than K and varies among input images or videos (Fig. 2.6). Fig. 2.1 gives an overview of our proposed approach.

Token Scoring: Let $I \in \mathbb{R}^{(N+1)\times d}$ be the input tokens of a self-attention layer with $N+1$ tokens. Before forwarding the input tokens through the model, ViT concatenates a classification token to the input tokens. The corresponding output token at the final transformer block is then fed to the classification head to get the class probabilities. Practically, this token is placed as the first token in each block and it is considered as a classification token. While we keep the classification token, our goal is to reduce the output tokens $O \in \mathbb{R}^{(K'+1)\times d}$ such that K' is dynamically adapted based on the input image or video and $K' \leq K \leq N$,

where K is a parameter that controls the maximum number of sampled tokens. Fig. 2.6 shows how the number of sampled tokens K' varies for different input data and stages of a network. We first describe how each token is scored.

In a standard self-attention layer [447], the queries $Q \in \mathbb{R}^{(N+1)\times d}$, keys $K \in \mathbb{R}^{(N+1)\times d}$, and values $V \in \mathbb{R}^{(N+1)\times d}$ are computed from the input tokens $I \in \mathbb{R}^{(N+1)\times d}$. The attention matrix A is then calculated by the dot product of the queries and keys, scaled by $\sqrt{d}$:

$$A = \text{Softmax}\left(QK^T/\sqrt{d}\right). \tag{2.1}$$

Due to the Softmax function, each row of $A \in \mathbb{R}^{(N+1)\times(N+1)}$ sums up to 1. The output tokens are then calculated using a combination of the values weighted by the attention weights:

$$O = AV. \tag{2.2}$$

Each row of A contains the attention weights of an output token. The weights indicate the contributions of all input tokens to the output token. Since $A_{1,:}$ contains the attention weights of the classification token, $A_{1,j}$ represents the importance of the input token j for the output classification token. Thus, we use the weights $A_{1,2}, \ldots, A_{1,N+1}$ as significance scores for pruning the attention matrix A, as illustrated in Fig. 2.1. Note that $A_{1,1}$ is not used since we keep the classification token. As the output tokens O depend on both A and V (2.2), we also take into account the norm of V_j for calculating the j^{th} token's significance score. The motivation is that values having a norm close to zero have a low impact and their corresponding tokens are thus less significant. In our experiments, we show that multiplying $A_{1,j}$ with the norm of V_j improves the results. The significance score of a token j is thus given by

$$S_j = \frac{A_{1,j} \times ||V_j||}{\sum_{i=2} A_{1,i} \times ||V_i||} \tag{2.3}$$

where $i, j \in \{2 \ldots N\}$. For a multi-head attention layer, we calculate the scores for each head and then sum the scores over all heads.

Token Sampling: Having computed the significance scores of all tokens, we can prune their corresponding rows from the attention matrix A. To do so, a naive approach is to select K tokens with the highest significance scores (top-K selection). However, this approach does not perform well, as we show in our experiments and it can not adaptively select $K' \leq K$ tokens. is that it discards all tokens with lower scores. Some of these tokens, however, can be useful in particular at the earlier stages when the features are less discriminative. For instance, having multiple tokens with similar keys, which may occur in the early stages,

will lower their corresponding attention weights due to the Softmax function. Although one of these tokens would be beneficial at the later stages, taking the top-K tokens might discard all of them. Therefore, we suggest sampling tokens based on their significance scores. In this case, the probability of sampling one of the several similar tokens is equal to the sum of their scores. We also observe that the proposed sampling procedure selects more tokens at the earlier stages than the later stages as shown in Fig. 2.2.

For the sampling step, we suggest using inverse transform sampling to sample tokens based on their significance scores S (2.3). Since the scores are normalized, they can be interpreted as probabilities and we can calculate the cumulative distribution function (CDF) of S:

$$\mathrm{CDF}_i = \sum_{j=2}^{j=i} S_j. \tag{2.4}$$

Note that we start with the second token since we keep the first token. Having the cumulative distribution function, we obtain the sampling function by taking the inverse of the CDF:

$$\Psi(k) = \mathrm{CDF}^{-1}(k) \tag{2.5}$$

where $k \in [0, 1]$. In other words, the significance scores are used to calculate the mapping function between the indices of the original tokens and the sampled tokens. To obtain K samples, we can sample K-times from the uniform distribution $U[0, 1]$. While such randomization might be desirable for some applications, deterministic inference is in most cases preferred. Therefore, we use a fixed sampling scheme for training and inference by choosing $k = \{\frac{1}{2K}, \frac{3}{2K} \ldots, \frac{2K-1}{2K}\}$. Since $\Psi(.) \in \mathbb{R}$, we consider the indices of the tokens with the nearest significant scores as the sampling indices.

If a token is sampled more than once, we only keep one instance. As a consequence, the number of unique indices K' is often lower than K as shown in Fig. 2.6. In fact, $K' < K$ if there is at least one token with a score $S_j \geq 2/K$. In the two extreme cases, either only one dominant token is selected and $K' = 1$ or $K' = K$ if the scores are more or less balanced. Interestingly, more tokens are selected at the earlier stages, where the features are less discriminative and the attention weights are more balanced, and less at the later stages, as shown in Fig. 2.2. The number and locations of tokens also vary for different input images, as shown in Fig. 2.7. For images with a homogeneous background that covers a large part of the image, only a few tokens are sampled. In this case, the tokens cover the object in the foreground and are sparsely but uniformly sampled from the background. In cluttered images, many tokens are required. It illustrates the importance of making the token sampling procedure adaptive.

FIGURE 2.2. Visualization of the gradual token sampling procedure in the multi-stage DeiT-S+ATS model. As it can be seen, at each stage, those tokens that are considered to be less significant to the classification are masked and the ones that have contributed the most to the model's prediction are sampled. We also visualize the token sampling results with Top-K selection to have a better comparison to our Inverse Transform Sampling.

Having indices of the sampled tokens, we refine the attention matrix $A \in \mathbb{R}^{(N+1)\times(N+1)}$ by selecting the rows that correspond to the sampled $K' + 1$ tokens. We denote the refined attention matrix by $A^s \in \mathbb{R}^{(K'+1)\times(N+1)}$. To obtain the output tokens $O \in \mathbb{R}^{(K'+1)\times d}$, we thus replace the attention matrix A by the refined one A^s in (2.2) such that:

$$O = A^s V. \tag{2.6}$$

These output tokens are then taken as input for the next stage. In our experimental evaluation, we demonstrate the efficiency of the proposed adaptive token sampler, which can be added to any vision transformer.

2.1.4 Experiments

In this section, we analyze the performance of our ATS module by adding it to different backbone models and evaluating them on ImageNet [101], Kinetics-400 [226], and Kinetics-600 [60], which are large-scale image and video classification datasets, respectively. In addition, we perform several ablation studies to better analyze our method. For the image classification task, we evaluate our proposed method on the

ImageNet [101] dataset with 1.3M images and 1K classes. For the action classification task, we evaluate our approach on the Kinetics-400 [226] and Kinetics-600 [60] datasets with 400 and 600 human action classes, respectively. We use the standard training/testing splits and protocols provided by the ImageNet and Kinetics datasets. If not otherwise stated, the number of output tokens of the ATS module are limited by the number of its input tokens. For example, we set $K = 197$ in case of DeiT-S [431]. For the image classification task, we follow the fine-tuning setup of [359] if not mentioned otherwise. The fine-tuned models are initialized by their backbones' pre-trained weights and trained for 30 epochs using PyTorch AdamW optimizer (lr= 5e−4, batch size = 8×96). We use the cosine scheduler for training the networks. For more implementation details and also information regarding action classification models, please refer to the supplementary materials.

2.1.4.1 Ablation Experiments

First, we analyze different setups for our ATS module. Then, we investigate the efficiency and effects of our ATS module when incorporated in different models. If not otherwise stated, we use the pre-trained DeiT-S [431] model as the backbone and we do not fine-tune the model after adding the ATS module. We integrate the ATS module into stage 3 of the DeiT-S [431] model. We report the results on the ImageNet-1K validation set in all of our ablation studies.

Significance Scores. As mentioned in Sec. 2.1.3, we use the attention weights of the classification token as significance scores for selecting our candidate tokens. In this experiment, we evaluate different approaches for calculating significance scores. Instead of directly using the attention weights of the classification token, we sum over the attention weights of all tokens (rows of the attention matrix) to find tokens with highest significance over other tokens. We show the results of this method in Fig. 2.3 labeled as Self-Attention score. As it can be seen, using the attention weights of the classification token performs better specially in lower FLOPs regimes. The results show that the attention weights of the classification token are a much stronger signal for selecting the candidate tokens. The reason for this is that the classification token will later be used to predict the class probabilities in the final stage of the model. Thus, its corresponding attention weights show which tokens have more impact on the output classification token. Whereas summing over all attention weights only shows us the tokens with highest attention from all other tokens, which may not necessarily be useful for the classification token. To better investigate this observation, we also randomly select another token rather than the classification token and use its attention weights for the score assignment. As shown, this approach performs much worse than the other ones both in high and low FLOPs

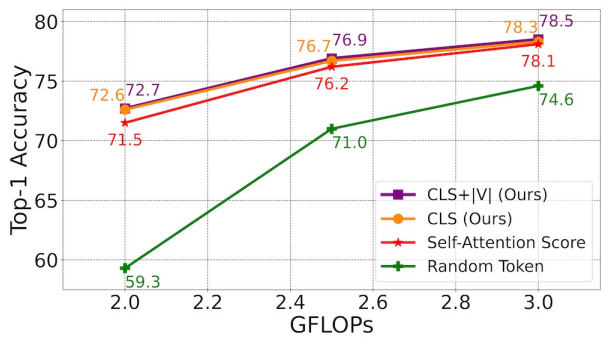


FIGURE 2.3. Impact of different score assignment methods. To achieve different GFLOPs levels, we bound the value of K from above such that the average GFLOPs of our adaptive models over the ImageNet validation set reaches the desired level. For more details, please refer to the supplementary material.

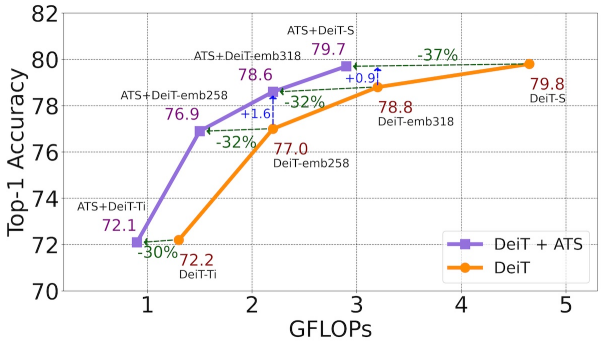
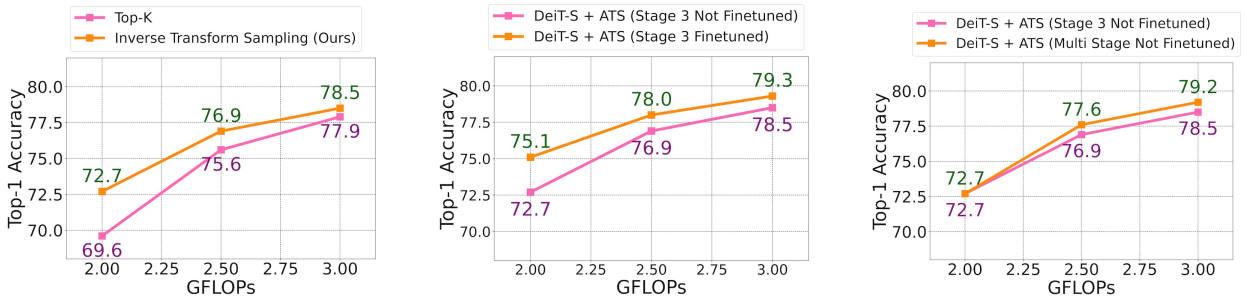


FIGURE 2.4. Performance comparison on the ImageNet validation set. Our proposed adaptive token sampling method achieves a state-of-the-art trade-off between accuracy and GFLOPs. We can reduce the GFLOPs of DeiT-S by 37% while almost maintaining the accuracy.

regimes. We also investigate the impact of using the L_2 norm of the values in Equation (2.3). As it can be seen in Fig. 2.3, it improves the results by about 0.2%.



(a) Sampling Methods (b) Fine-tuning (c) Multi vs. Single Stage

FIGURE 2.5. For the model with Top-K selection (fixed-rate sampling) (2.5a), we set K such that the model operates at a desired GFLOPs level. In all three plots, we control the GFLOPs level of our adaptive models as in Fig. 2.3. We use DeiT-S [431] for these experiments. For more details, please refer to the supplementary material.

Candidate Tokens Selection. As mentioned in Sec. 2.1.3, we employ the inverse transform sampling approach to softly downsample the input tokens. To better investigate this approach, we also evaluate the model's performance when picking the top K tokens with highest significance scores S. As it can be seen in Fig. 2.5a, our inverse transform sampling approach outperforms the Top-K selection both in high and low GFLOPs regimes. As discussed earlier, our inverse transform sampling approach based on the CDF of the scores does not hardly discard all tokens with lower significance scores and hence provides a more diverse set of tokens for the following layers. Since earlier transformer blocks are more prone to predict

noisier attention weights for the classification token, such a diversified set of tokens can better contribute to the output classification token of the final transformer block. Moreover, the Top-K selection method will result in a fixed token selection rate at every stage that limits the performance of the backbone model. This is shown by the examples in Fig. 2.2. For a cluttered image (bottom), inverse transform sampling keeps a higher number of tokens across all transformer blocks compared to the Top-K selection and hence preserves the accuracy. On the other hand, for a less detailed image (top), inverse transform sampling will retain less tokens, which results in less computation cost.

Model Scaling. Another common approach for changing the GFLOPs/accuracy trade-off of networks is to change the channel dimension. To demonstrate the efficiency of our adaptive token sampling method, we thus vary the dimensionality. To this end, we first train several DeiT models with different embedding dimensions. Then, we integrate our ATS module into the stages 3 to 11 of these DeiT backbones and fine-tune the networks. In Fig. 2.4, we can observe that our approach can reduce GFLOPs by 37% while maintaining the DeiT-S backbone's accuracy. We can also observe that the GFLOPs reduction rate gets higher as we increase the embedding dimensions from 192 (DeiT-Ti) to 384 (DeiT-S). The results show that our ATS module can reduce the computation cost of the models with larger embedding dimensions to their variants with smaller embedding dimensions.

Visualizations. To better understand the way our ATS module operates, we visualize our token sampling procedure (Inverse Transform Sampling) in Fig. 2.2. We have incorporated our ATS module in the stages 3 to 11 of the DeiT-S network. The tokens that are discarded at each stage are represented as a mask over the input image. We observe that our DeiT-S+ATS model has gradually removed irrelevant tokens and sampled those tokens which are more significant to the model's prediction. In both examples, our method identified the tokens that are related to the target objects as the most informative tokens.

We show more visual results in Fig. 2.8. We select several images of the ImageNet validation set with various amounts of detail and complexity. We visualize the progressive token sampling procedure of our multi-stage DeiT-S+ATS model for the selected images. The number of output tokens of each ATS module in the multi-stage DeiT-S+ATS model is limited by the number of its input tokens, which is 197. Our adaptive model samples a higher number of tokens when the input images are more cluttered. We can also observe that the sampled tokens are more scattered in images with more details compared to more plain images.

Adaptive Sampling. In this experiment, we investigate the adaptivity of our token sampling approach. We evaluate our multi-stage DeiT-S+ATS model on the ImageNet validation set. In Fig. 2.6, we visualize

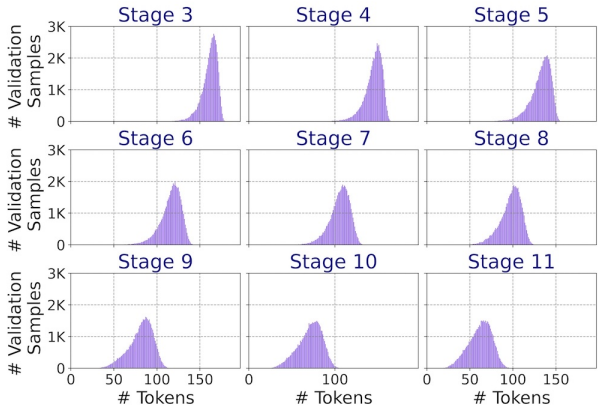


FIGURE 2.6. Histogram of the number of sampled tokens at each ATS stage of our multi-stage DeiT-S+ATS model on the ImageNet validation set. The y-axis corresponds to the number of images and the x-axis to the number of sampled tokens.

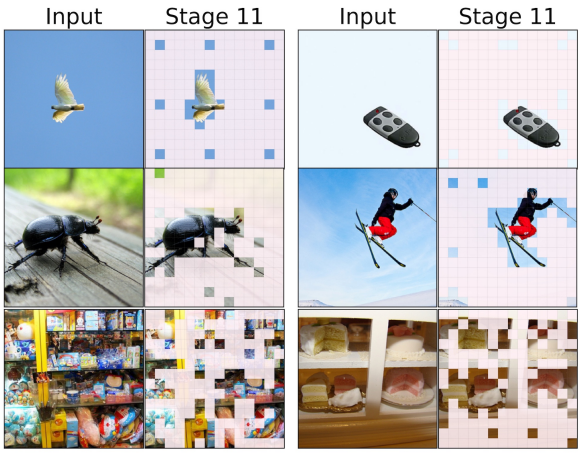


FIGURE 2.7. ATS samples less tokens for images with fewer details (top), and a higher number of tokens for more detailed images (bottom). We show the token downsampling results after all ATS stages. For this experiment, we use a multi-stage Deit-S+ATS model.

histograms of the number of sampled tokens at each ATS stage. We can observe that the number of selected tokens varies at all stages and for all images. We also qualitatively analyze this nice property of our ATS module in Figs. 2.2 and 2.7. We can observe that our ATS module selects a higher number of tokens when it deals with detailed and complex images while it selects a lower number of tokens for less detailed images.

Fine-tuning. To explore the influence of fine-tuning on the performance of our approach, we fine-tune a DeiT-S+ATS model on the ImageNet training set. We compare the model with and without fine-tuning. As shown in Fig. 2.5b, fine-tuning can improve the accuracy of the model. In this experiment, we fine-tune the model with $K = 197$ but test it with different K values to reach the desired GFLOPs levels.

ATS Stages. In this experiment, we explore the effect of single-stage and multi-stage integration of the ATS block into vision transformer models. In the single-stage model, we integrate our ATS module into the stage 3 of DeiT-S. In the multi-stage model, we integrate our ATS module into the stages 3 to 11 of DeiT-S. As it can be seen in Fig. 2.5c, the multi-stage DeiT-S+ATS performs better than the single-stage DeiT-S+ATS. This is due to the fact that a multi-stage DeiT-S+ATS model can gradually decrease the GFLOPs by discarding fewer tokens in earlier stages, while a single-stage DeiT-S+ATS model has to discard more tokens in earlier stages to reach the same GFLOPs level.

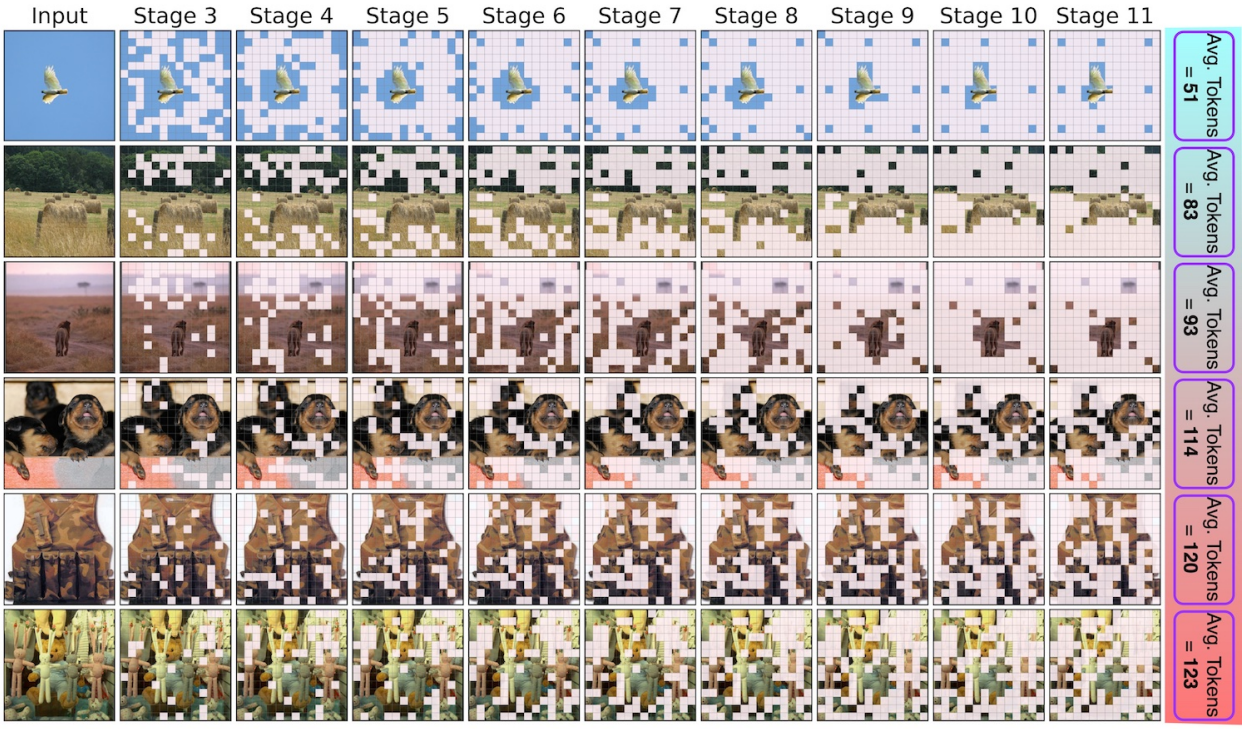


FIGURE 2.8. Visualization of the gradual token sampling procedure in the multi-stage DeiT-S+ATS model. We integrate our ATS module into the stages 3 to 11 of the DeiT-S model. The tokens that are sampled at each stage of the network are shown for images that are ordered by their complexity (from low complexity to high complexity). We visualize the tokens, which are discarded, as masks over the input images. As it can be seen, a higher number of tokens are sampled for more cluttered images while a lower number of tokens are required when the images contain less details. Additionally, we can see that the sampled tokens are more focused and less scattered in images with less details.

2.1.4.2 Comparison with State-of-The-Art

We compare the performances of our adaptive models, which are equipped with the ATS module, with state-of-the-art vision transformers for image and video classification on the ImageNet-1K [101] and Kinetics [60, 226] datasets, respectively. Tables 2.1-2.3 show the results of this comparison. For the image classification task, we incorporate our ATS module into the stages 3 to 11 of the DeiT-S [431] model. We also integrate our ATS module into the 1^{st} to 9^{th} blocks of the 3^{rd} stage of CvT-13 [476] and CvT-21 [476], and into stages 1-9 of the transformer module of PS-ViT [511]. We fine-tune the models on the ImageNet-1K training set. We also evaluate our ATS module for action recognition. To this end, we add our module to the XViT [48] and TimeSformer [38] video vision transformers. For more details, please refer to the supplementary materials.

TABLE 2.1. Comparison of the multi-stage ATS models with state-of-the-art image classification models with comparable GFLOPs on the ImageNet validation set. We equip DeiT-S [431], PS-ViT [511], and variants of CvT [476] with our ATS module and fine-tune them on the ImageNet training set.

Model	Params (M)	GFLOPs	Resolution	Top-1
ViT-Base/16 [?]	86.6	17.6	224	77.9
HVT-S-1 [330]	22.09	2.4	224	78.0
IA-RED^2 [328]	-	2.9	224	78.6
DynamicViT-DeiT-S (30 Epochs) [359]	22.77	2.9	224	79.3
EViT-DeiT-S (30 epochs) [274]	22.1	3.0	224	79.5
DeiT-S+ATS (Ours)	**22.05**	**2.9**	224	79.7
DeiT-S [431]	22.05	4.6	224	79.8
PVT-Small [461]	24.5	3.8	224	79.8
CoaT Mini [489]	10.0	6.8	224	80.8
CrossViT-S [62]	26.7	5.6	224	81.0
PVT-Medium [461]	44.2	6.7	224	81.2
Swin-T [288]	29.0	4.5	766	81.3
T2T-ViT-14 [508]	22.0	5.2	224	81.5
CPVT-Small-GAP [88]	23.0	4.6	817	81.5
CvT-13 [476]	20.0	4.5	224	81.6
CvT-13+ATS (Ours)	20.0	**3.2**	224	81.4
PS-ViT-B/14 [511]	21.3	5.4	224	81.7
PS-ViT-B/14+ATS (Ours)	21.3	**3.7**	224	81.5
RegNetY-8G [355]	39.0	8.0	224	81.7
DeiT-Base/16 [431]	86.6	17.6	224	81.8
CoaT-Lite Small [489]	20.0	4.0	224	81.9
T2T-ViT-19 [508]	39.2	8.9	224	81.9
CrossViT-B [62]	104.7	21.2	224	82.2
T2T-ViT-24 [508]	64.1	14.1	224	82.3
PS-ViT-B/18 [511]	21.3	8.8	224	82.3
PS-ViT-B/18+ATS (Ours)	21.3	**5.6**	224	82.2
CvT-21 [476]	32.0	7.1	224	82.5
CvT-21+ATS (Ours)	32.0	**5.1**	224	82.3
TNT-B [168]	66.0	14.1	224	82.8
RegNetY-16G [355]	84.0	16.0	224	82.9
Swin-S [288]	50.0	8.7	224	83.0
CvT-13$_{384}$ [476]	20.0	16.3	384	83.0
CvT-13$_{384}$+ATS (Ours)	20.0	**11.7**	384	82.9
Swin-B [288]	88.0	15.4	224	83.3
LV-ViT-S [218]	26.2	6.6	224	83.3
CvT-21$_{384}$ [476]	32.0	24.9	384	83.3
CvT-21$_{384}$+ATS (Ours)	32.0	**17.4**	384	83.1

TABLE 2.2. Comparison with state-of-the-art on Kinetics-400.

Model	Top-1	Top-5	Views	GFLOPs
STC [104]	68.7	88.5	112	-
bLVNet [123]	73.5	91.2	3×3	840
STM [276]	73.7	91.6	-	-
TEA [271]	76.1	92.5	10×3	2,100
TSM R50 [217]	74.7	-	10×3	650
I3D NL [463]	77.7	93.3	10×3	10,800
CorrNet-101 [458]	79.2	-	10×3	6,700
ip-CSN-152 [440]	79.2	93.8	10×3	3,270
HATNet [105]	79.3	-	-	-
SlowFast 16×8 R101+NL [132]	79.8	93.9	10×3	7,020
X3D-XXL [131]	80.4	94.6	10×3	5,823
TimeSformer-L [38]	80.7	94.7	1×3	7,140
TimeSformer-L+ATS (Ours)	80.5	94.6	1×3	**3,510**
ViViT-L/16x2 [38]	80.6	94.7	4×3	17,352
MViT-B, 64×3 [122]	81.2	95.1	3×3	4,095
X-ViT (16×) [48]	80.2	94.7	1×3	425
X-ViT+ATS (16×) (Ours)	80.0	94.6	1×3	**259**
TokenLearner 16at12 (L/16) [379]	82.1	-	4×3	4,596

TABLE 2.3. Comparison with state-of-the-art on Kinetics-600.

Model	Top-1	Top-5	Views	GFLOPs
AttentionNAS [466]	79.8	94.4	-	1,034
LGD-3D R101 [351]	81.5	95.6	10×3	-
HATNET [105]	81.6	-	-	-
SlowFast R101+NL [132]	81.8	95.1	10×3	3,480
X3D-XL [131]	81.9	95.5	10×3	1,452
X3D-XL+ATFR [125]	82.1	95.6	10×3	768
TimeSformer-HR [38]	82.4	96	1×3	5,110
TimeSformer-HR+ATS (Ours)	82.2	96	1×3	**3,103**
ViViT-L/16x2 [38]	82.5	95.6	4×3	17,352
Swin-B [288]	84.0	96.5	4×3	3,384
MViT-B-24, 32×3 [122]	84.1	96.5	1×5	7,080
TokenLearner 16at12(L/16) [379]	84.4	96.0	4×3	9,192
X-ViT (16×) [48]	84.5	96.3	1×3	850
X-ViT+ATS (16×) (Ours)	84.4	96.2	1×3	**521**

Image Classification. As it can be seen in Table 2.1, our ATS module decreases the GFLOPs of all vision transformer models without adding any extra parameters to the backbone models. For the DeiT-S+ATS model, we observe a 37% GFLOPs reduction with only 0.1% reduction of the top-1 accuracy. For the CvT+ATS models, we can also observe a GFLOPs reduction of about 30% with 0.1 – 0.2% reduction of the top-1 accuracy. More details on the efficiency of our ATS module can be found in the supplementary materials (*e.g.* throughput). Comparing ATS to DynamicViT [359] and HVT [330], which add additional parameters to the model, our approach achieves a better trade-off between accuracy and GFLOPs. Our method also outperforms the EViT-DeiT-S [274] model trained for 30 epochs without adding any extra trainable parameters to the model. We note that the EViT-DeiT-S model can improve its top-1 accuracy by

around 0.3% when it is trained for much more training epochs (*e.g.* 100 epochs). For a fair comparison, we have considered the 30 epochs training setup used by Dynamic-ViT [359]. We have also added our ATS module to the PS-ViT network [511]. As it can be seen in Table 2.1, although PS-ViT has drastically lower GFLOPs compared to its counterparts, its GFLOPs can be further decreased by incorporating ATS in it.

Action Recognition. As it can be seen in Tables 2.2 and 2.3, our ATS module drastically decreases the GFLOPs of all video vision transformers without adding any extra parameters to the backbone models. For the XViT+ATS model, we observe a 39% GFLOPs reduction with only 0.2% reduction of the top-1 accuracy on Kinetics-400 and a 38.7% GFLOPs reduction with only 0.1% drop of the top-1 accuracy on Kinetics-600. We observe that XViT+ATS achieves a similar accuracy as TokenLearner [379] on Kinetics-600 while requiring 17.6× less GFLOPs. For TimeSformer-L+ATS, we can observe 50.8% GFLOPs reduction with only 0.2% drop of the top-1 accuracy on Kinetics-400. These results demonstrate the generality of our approach that can be applied to both image and video representations.

2.1.5 Conclusion

Designing computationally efficient vision transformer models for image and video recognition is a challenging task. In this work, we proposed a novel differentiable parameter-free module called Adaptive Token Sampler (ATS) to increase the efficiency of vision transformers for image and video classification. The new ATS module selects the most informative and distinctive tokens within the stages of a vision transformer model such that as much tokens as needed but not more than necessary are used for each input image or video clip. By integrating our ATS module into the attention layers of current vision transformers, which use a static number of tokens, we can convert them into much more efficient vision transformers with an adaptive number of tokens. We showed that our ATS module can be added to off-the-shelf pre-trained vision transformers as a plug and play module, thus reducing their GFLOPs without any additional training, but it is also possible to train a vision transformer equipped with the ATS module thanks to its differentiable design. We evaluated our approach on the ImageNet-1K image recognition dataset and incorporated our ATS module into three different state-of-the-art vision transformers. We also demonstrated the generality of our approach by incorporating it into different state-of-the-art video vision transformers and evaluating them on the Kinetics-400 and Kinetics-600 datasets. The results show that the ATS module decreases the computation cost (GFLOPs) between 27% and 50.8% with a negligible accuracy drop. Although our experiments are focused on image and video vision transformers, we believe that our approach can also work in other domains such as audio.

2.2 SimA: Simple Softmax-free Attention for Vision Transformers

Recently, vision transformers have become very popular. However, deploying them in many applications is computationally expensive partly due to the Softmax layer in the attention block. We introduce a simple yet effective, Softmax-free attention block, SimA, which normalizes query and key matrices with simple ℓ_1-norm instead of using Softmax layer. Then, the attention block in SimA is a simple multiplication of three matrices, so SimA can dynamically change the ordering of the computation at the test time to achieve linear computation on the number of tokens or the number of channels. We empirically show that SimA applied to three SOTA variations of transformers, DeiT, XCiT, and CvT, results in on-par accuracy compared to the SOTA models, without any need for Softmax layer. Interestingly, changing SimA from multi-head to single-head has only a small effect on the accuracy, which further simplifies the attention block. Moreover, we show that SimA is much faster on small edge devices, e.g., Raspberry Pi, which we believe is due to higher complexity of Softmax layer on those devices. The code is available here: https://github.com/UCDvision/sima

2.2.1 Introduction

Recently, vision transformers have become very popular. Compared to CNNs, they achieve better accuracy, however, deploying transformers in devices with smaller computational resources is challenging. One reason is that a transformer model calls the Softmax layer several times which calls $exp(.)$ operation consequently. We know that the $exp(.)$ operation is costly particularly in smaller devices with limited computational resources. For instance, implementing $exp(.)$ on FGPA is much more costly compared to implementing simple multiplication or addition operations.

As an example observation, Table A1 of [209] measures the run-time of each component for a BERT encoder on V100 GPUs. Softmax consumes more time compared to any other components including query (Q), key (K), value (V) operation (Softmax: 453 μs , QKV projections: 333 μs, QK^T: 189 μs). This is remarkable since the FLOPS of Softmax is much lower than those other components (Softmax: 0.2 GFLOPS, QKV projections: 25.7 GFLOPS, QK^T: 4.3 GFPLOS). Similar observation are made in [405, 448].

We are interested in simplifying the attention mechanism by removing the Softmax layer. We believe one role of the Softmax layer is to normalize the attention values so that tokens can compete with each other. Our main idea is to enable this competition by normalizing the query and key matrices with their ℓ_1-norm

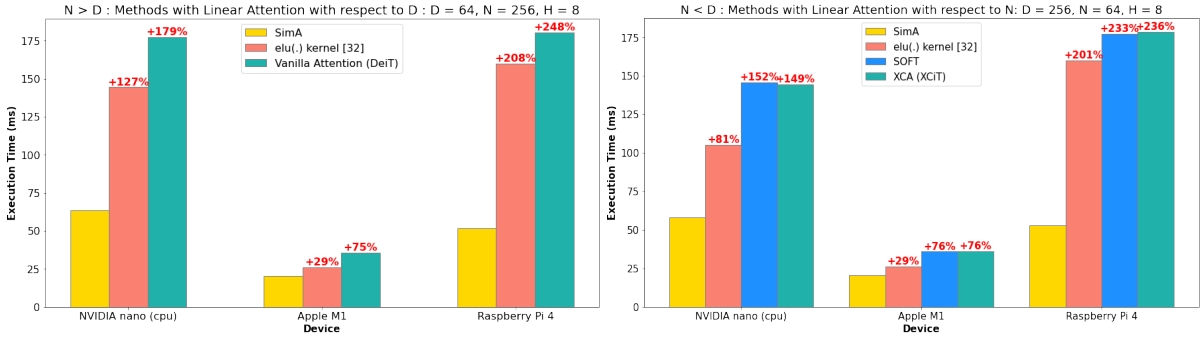


FIGURE 2.9. **Comparison on Edge devices:** We evaluate performance of a single attention block for each model on 3 different devices: Raspberry Pi 4 (Quad core Cortex-A72 @ 1.5GHz), NVIDIA Jetson Nano (Quad-core ARM A57 @ 1.43 GHz), and Apple M1. To measure the effect of $exp(.)$ only, we fix the order of (QK^TV) product so that all models have the same dot product complexity. We set $N > D$ for left and $N < D$ for the right plots. We repeat average of the execution time over 1000 runs. We observe that SimA is faster than other methods, which we believe is due to the increased complexity of $exp(.)$ operation compared to ℓ_1 normalization on edge devices.

before multiplying them. Then, removing the Softmax layer results in the whole attention mechanism to boil down to simply multiplying three matrices "query", "key", and "value". While ℓ_1-norm has been used in transformers before [163], the way we are using it to simplify the computational flow of the transformer is novel.

As a bi-product, due to the associative property of multiplication, there are two possible orderings of multiplying these three matrices at the test time. Depending on the ordering, the computation can be quadratic on the number of tokens, N, or that of channels, D. Hence, we can reduce the computation further by dynamically deciding on the ordering at the test time by comparing N and D without affecting the training process. Moreover, since we normalize the vectors before multiplying, our method is numerically more stable so we use half-precision floating point without overflowing.

The attention mechanism deals with the tokens without considering their ordering. This is an interesting property that opens the door to many applications. For instance, the distribution of the tokens is relatively robust compared to CNNs when we mask (drop) 75% of the tokens in masking auto-encoder (MAE [174]). Moreover, the tokens can be seen as a non-ordered set that can come from various sources (e.g., multiple cameras or non-camera sensors). Note that this permutation equivariance property does not exist in some other models like MLP-Mixer [428]. Hence, instead of using MLP-Mixer that does not have Softmax by default, we are interested in removing Softmax from the original transformers to keep this permutation equivariance property.

We perform experiments with our simple attention block, denoted SimA, by using it in standard vision transformers, DeiT, CvT, and XCiT. SimA achieves on-par results with SOTA on ImageNet classification, MS-COCO object detection and segmentation, and also self-supervised learning.

In summary, our SimA attention block does not use Softmax, which makes it computationally efficient generally (see Fig. 2.9 and Table 2.6), and on the edge devices specifically. SimA can dynamically choose to be linear on N or D at the test time depending on the image resolution or the number of tokens. Changing Multi-head attention to Single-head one or changing GELU activation function to ReLU, has a very small effect on the accuracy of SimA. This makes SimA simple and effective for various applications.

2.2.2 Method

2.2.2.1 Background on Vision Transformers:

Self-Attention Block: The original vision transformer [110] uses the self-attention block introduced in [446]. Self-attention block gets $X \in \mathbb{R}^{N \times D}$ as the input where N is the number of tokens and D is the dimensionality of each token. Then $W_q \in \mathbb{R}^{D \times D}$, $W_k \in \mathbb{R}^{D \times D}$ and $W_v \in \mathbb{R}^{D \times D}$ projects X into three $N \times D$ matrices: query ($Q = XW_q$), key ($K = XW_k$) and value ($V = XW_v$). We calculate attention matrix $A \in \mathbb{R}^{N \times N}$ defined as $A = Softmax(QK^T / \sqrt{D})$ where $Softmax$ is applied to each row independently, so each row in A sums to one. Then, we calculate the output $O = AV$. Each row of $O \in \mathbb{R}^{N \times D}$ corresponds to one token and since rows of A sum to one, each token in a weighted average of the values of all tokens.

Additionally, Multi-Head Self-Attention (MSA) transformers divide Q, K, and V of each token into H heads, where each head has its own attention over the corresponding head in all tokens. For example, $Q = [Q_1; Q_2; ...; Q_H]$ where $Q_i \in \mathbb{R}^{N \times \frac{D}{H}}$ is the query matrix for the i'th head. Then, we calculate H self-attention for all heads in parallel and concatenate the outputs to get $O = [O_1; O_2; ...; O_H]$. Finally, the self-attention block has an additional output projection $W_{proj} \in \mathbb{R}^{D \times D}$, thus the final output of the self-attention block is OW_{proj} which is of size $\mathbb{R}^{N \times D}$.

Cross-covariance Attention Block (XCA): Vanilla self-attention block has a complexity of $O(DN^2)$ which is quadratic on N. [23, 392] introduce an attention mechanism that is linear on N. In XCA, we calculate the attention matrix with $A = K^T Q$ where A is a $D \times D$ matrix. Next, we apply Softmax on each columns, so that columns sum to one. Then we calculate output as $O = VA$. Note that A is an attention of channels on each other rather than tokens. Compared to vanilla self-attention (MSA), XCA has complexity of $O(D^2N)$. Since XCA is linear on N, it is more efficient when $N \gg D$ and it is less efficient when $N \ll D$.

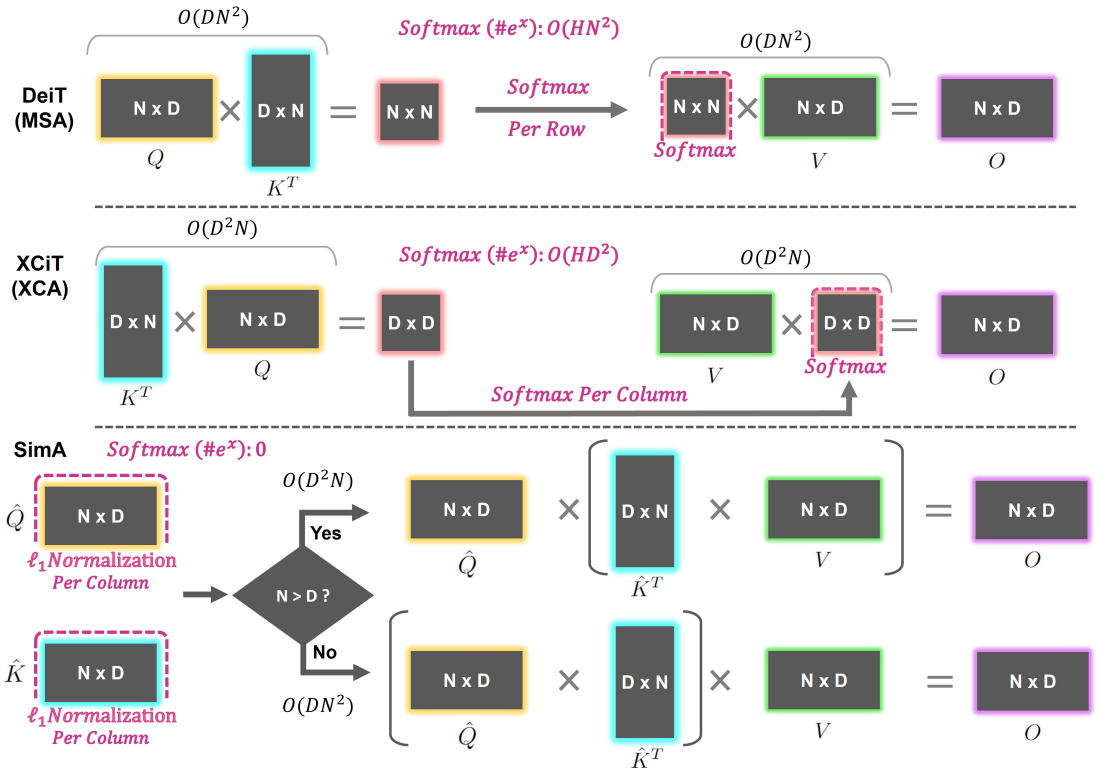


FIGURE 2.10. **Our Simple Attention (SimA):** First, we normalize each channel in Q and K with ℓ_1-norm across the tokens, to get $\hat{Q}$ and $\hat{K}$. Next, we can choose either $(\hat{Q}\hat{K}^T)V$ or $\hat{Q}(\hat{K}^T V)$ depending on the number of input tokens N. Compared to XCA and MSA, our method has following benefits: (1) It is free of Softmax, hence it is more efficient. (2) At test time we can dynamically switch between $(\hat{Q}\hat{K}^T)V$ and $\hat{Q}(\hat{K}^T V)$ based on the number of input tokens (e.g., different image resolution).

Vision Transformer Block: Vision transformers architecture contains n consecutive Vision Transformer blocks. Each block has MSA block followed by a Feed-Forward Network (FFN) both with skip connection. FFN is a simple 2-layer MLP which projects tokens from D dimension to $4D$ and again back to D dimensions. FFN uses GELU [185] as the activation function. Moreover, we use LayerNorm [30] on each token before forwarding them through MSA or FFN blocks. The following two updating rules summarize each block of the vision transformer:

$$(Step1) \quad X \leftarrow X + MSA(\text{LayerNorm}_1(X))$$

$$(Step2) \quad X \leftarrow X + FFN(\text{LayerNorm}_2(X))$$

2.2.2.2 Simple Attention (SimA):

Our main goal is to reduce the computation by removing the Softmax ($exp(.)$) layer. We believe one of the roles of the Softmax layer is to normalize the attention so that each token is a weighted average of the

values of all tokens. This ensures that the attention values are bounded. Hence, we introduce an alternative normalization method that does not need a Softmax layer.

In the regular attention block, if a channel in Q and/or K has large values, that channel may dominate the dot product QK^T. This results in other channels being ignored in calculating the attention. We believe this may be one of the reasons leading to superior performance of the multi-head attention (MSA) compared to the single-head one. Since in MSA, the dominating channel can dominate a single head only leaving the other heads still operational. We propose a method to take this solution to the extreme where we can normalize each channel in Q and K across tokens so that different channels become more comparable. We do this by simply dividing the values of each channel by the ℓ_1 norm of that channel across all tokens:

$$\hat{Q}^i := \frac{Q^i}{|Q^i|_1} \quad \text{and} \quad \hat{K}^i := \frac{K^i}{|K^i|_1}$$

where Q^i is the i'th column of Q (values of the i'th channel for all tokens) and $\hat{Q}$ and $\hat{K}$ are the normalized query and key matrices. Given this simple normalization method, we remove the Softmax layer, so the attention block can be written as:

$$O = \hat{Q}\hat{K}^T V$$

where $O \in \mathbb{R}^{N \times D}$. Similar to standard transformers, we use this block for each head separately, concatenate the outputs, and finally apply the output projection OW_{proj}.

One can assume $\hat{Q}\hat{K}^T$ is the attention matrix that quantifies the effect of a token on another token. Interestingly, if the query and key vectors have a large angle, the attention values can become negative, meaning that a token can affect another one negatively. This is in contrast to regular transformers where the attention is always non-negative. A simple pseudo-code is provided in the appendix.

Due to our normalization, the attention values are bounded between $-D$ and D. The extremes happen when only a single row of Q and a single row of K are nonzero. In this case, all other tokens will have zero query and key vectors. One may divide the attention by D to bound it between -1 and 1. This constant scalar multiplier can be absorbed into W_v, the projection matrix for V.

We visualize SimA in Fig 2.10 and the pseudocode of SimA in Algorithm 1.

The cost of Softmax: Both XCA and MSA use Softmax for normalization. Softmax needs running $exp(.)$ which is costly. MSA uses Softmax on a matrix of size $N \times N$ while XCA uses Softmax on a matrix of size $D \times D$. Hence, the order of $exp(.)$ operations is $O(HN^2)$ for MSA and $O(HD^2)$ for XCA. Therefore, Softmax will be bottleneck when increasing the number of tokens (higher image resolutions) in MSA and

number of channels (higher capacity transformers) in XCA. On the other hand, our attention block does not use $exp(.)$ operation at all. Moreover, in the last row of Table 2.4, we show that changing GELU to ReLU in SimA gets comparable accuracy to the main experiment (79.6% vs 79.8%). This version of SimA does not use any $exp(.)$ operation at the inference time. The reduction in the computation cost of Softmax for a single attention block is shown in Fig. 2.9-left for $N > D$ and in Fig. 2.9-right for $N < D$ and Table 2.6. This Figure shows the speed-up due to removing Softmax only and does not include the speed-up due to changing the order of multiplications. We believe removing the cost of $exp(.)$ operation can have a large impact particularly in edge devices with limited resources.

Algorithm 1: Pseudocode of SimA (Single Head) in a PyTorch-like style.

```
# self.qkv: nn.Linear(dim, dim * 3, bias=qkv_bias) ; query, key, value projection
# self.proj: nn.Linear(dim, dim, bias=output_proj_bias) ; output projection

def forward(self, x):
    B, N, D = x.shape # B: batch size, N: number of Tokens, D: Dimension of Tokens
    qkv = self.qkv(x).reshape(B, N, 3, D).permute(2, 0, 1, 3) # (3 x B x N x D)
    q, k, v = qkv[0], qkv[1], qkv[2] # split into query (B x N x D), key (B x N x D) and value (B x N x D)

    k = torch.nn.functional.normalize(k, p=1.0, dim=-2) # Normalized key (B x N x D)
    q = torch.nn.functional.normalize(q, p=1.0, dim=-2) # Normalized query (B x N x D)

    if (N/D) < 1:
        x = (q @ k.transpose(-2, -1)) @ v # (B x N x D)
    else:
        x = q @ (k.transpose(-2, -1) @ v) # (B x N x D)

    x = self.proj(x) # Output (B x N x D)
    return x
```

2.2.3 Related Work

Vision Transformers: Convolutional Neural Networks (CNNs) have become ubiquitous as the most commonly used network architecture for computer vision tasks [84, 178, 195, 249]. Transformers have recently emerged as a promising alternative to CNNs. Transformers [446] rely entirely on self-attention mechanism and was originally introduced for NLP tasks. ViT [110] adapts transformers to obtain convolution-free architecture for computer vision tasks by dividing each image in 16×16 patches and considering each patch as a token input. DeiT [431] improves training efficiency of ViT on smaller dataset. The Scaled Dot-Product Attention module [446] used by transformers rely on the softmax operation for normalization. Unlike CNNs/MLP [249, 397, 428, 429] based architectures, softmax is an important part of transformer architecture. In this paper, we address replacing the softmax ($exp(.)$) operation in the self-attention module of vision transformers.

Efficient Vision Transformers: Transformers have a large memory footprint, so deploying them on edge devices with limited resources is difficult. Many works study efficiency of transformers [44, 49, 227,

504]. LeViT [157] uses down-sampling in stages to improve efficiency. [308, 477] integrate convolution in transformer. [190, 289] improve the self-attention efficiency by limiting the attention of each token to subset of tokens. [294] uses distillation to improve the efficiency of the network. [128, 304, 360] decrease the number of tokens by token pruning. [285] apply quantization on transformers. Although these works limit the computation generally, softmax or $exp(.)$ function is still required to calculate the attention. Our idea is orthogonal to these methods since we can replace attention block in any transformer with our $exp(.)$ free attention block.

Linear Attention: Vanilla attention has $O(N^2D)$ computation and memory complexity, where N is number of input tokens and D is dimension of each token. Some works target this issue by replacing vanilla attention with a linear attention with $O(ND^2)$ complexity. XCiT [23, 392] uses attention across feature channels rather than tokens. Some works use similarity kernels to approximate softmax, thus it is possible to have linear complexity by doing $\phi(Q)(\phi(K)^T\phi(V))$ instead of $(\phi(Q)\phi(K)^T)\phi(V)$ where $\phi(x)$ is the kernel function. [225] uses $\phi(x) = 1 + elu(x)$, whereas [293, 341] use Gaussian kernel functions. [488] use SVD decomposition and [86] use positive random features to approximate softmax. [460] approximate attention with a low rank matrix. All these methods either use exponential function. For example, SOFT [293] removes Softmax without reducing the number of $exp(.)$ operations. Our ideas are different since we aim to remove the costly $exp(.)$ operation. Moreover, the focus of those methods is to have linear attention with respect to number of tokens which is not the main focus of this paper. A recent work in the NLP community, CosFormer [350], passes Q and K through a ReLU unit and normalizes their product. It also adds a re-weighting method that improves the locality of the data using $sine(.)$ and $cosine(.)$ functions. Our idea is simpler and we apply it to visual recognition rather than NLP. Moreover, cosine re-weighting in CosFormer requires 4× more FLOPs in K and V dot product compared to ours.

Softmax Approximation: Softmax is an expensive operation on hardware since it requires $exp(.)$ operation. More specifically, softmax in transformer architecture contributes to major part of computation when the input is large [405]. [33] approximates softmax with Taylor expansions, whereas [115, 146, 166, 543] target designing a hardware architecture to approximate softmax. Softermax [405] uses a low-precision implementation of 2^x. [515] uses lower precision computation. [280, 347] use quantized softmax. While these works approximate Softmax at the hardware, we replace Softmax completely with ℓ_1 normalization at the model architecture.

2.2.4 Experiments

We evaluate effectiveness of SimA attention block by replacing self-attention in three popular vision transformer families: DeiT, XCiT and CvT. We evaluate our model on image classification, object detection, image segmentation, and self-supervised learning.

2.2.4.1 Image Classification

Dataset: We train on ImageNet1K [101] and report Top-1 accuracy on the validation set.

Implementation Details: We use PyTorch [335] and Timm [473] libraries to train our models with a setup similar to [23,431]. We use AdamW [292] optimizer. We train CvT and DeiT models with 300 epochs and XCiT models with 400 epochs. We set the batch size to 1024 and weight decay to 0.05. We use cosine scheduling with an initial learning rate of $5e-4$. We use Stochastic depth drop rate [202] of 0.05. Data augmentations are the same as those in [431] including Rand-Augment [93], CutMix [512] and Mixup [519]. Following [23,435], we train our models with images of resolution 224 and evaluate it using images with a crop ratio of 1.0. Training DeiT-S or XCiT-S12/16 with 8 RTX 6000 GPUs takes approximately 100 hours.

We use SimA along with the following three transformer architectures to show its generalization:

- DeiT: [431] is a well-known transformer architecture based on ViT [110]. Since we do not use the distillation token introduced in DeiT, in our setting, DeiT is very similar to ViT except that it converged faster due to better optimization parameters. We use DeiT-S which has the following settings: patch size= 16, embedding dimensions= 384, number of heads= 6 and layers= 12. Self-attention in DeiT has complexity of $O(DN^2)$ which is quadratic on the number of tokens N.

- XCiT: [23] is a state-of-the-art vision transformer architecture with a linear attention. XCiT has 2 major differences compared to DeiT: [label=(0)]

XCiT has Local Patch Interaction (LPI) in each block, which consists of one depth-wise 3×3 convolution followed by Batch Normalization, GELU and another depth-wise 3×3 convolution.

XCiT has separate class attention layers similar to [435]. The CLS token is added at the end of the initial self-attention stage and class attention layers are used to aggregate information from image tokens to the class token. This modification adds extra parameters and computation to the model.

We replace SimA in three variant of XCiT: XCiT-S12/16, XCiT-T12/8 and XCiT-T24/16. XCiT-S12/16 has a patch size of 16, embedding dimension of 384, 8 heads, 12 layers, and 2 class attention layers. XCiT-T12/8 is similar to XCiT-S12/16 with a patch size of 8, embedding dimension of 192, and 4 heads. XCiT-T24/16 is similar to XCiT-T12/8 with patch size of 16.

TABLE 2.4. **ImageNet classification:** We denote replacing Softmax attention with SimA by X → SimA. Softmax column indicates the number of $exp(.)$ operations in the attention block. N is the number of tokens, D is the token dimension, H is the number of heads, M is the local window size, and R is the reduction ratio. We also report ResNet50 RA (with RandAug [93]). Models indicated by * use teacher during training. EfficientNet outperforms our method, but it is a convolutional network and uses more FLOPs at higher image resolution. SOFT also has $exp(.)$ function in the backbone which is costly. Purple rows are our method while blue rows are comparable baselines. Our method is a $exp(.)$ free transformer and has on-par accuracy with SOTA transformers. To simplify SimA even further, we investigate two more variations in yellow rows: (1) Replacing GELU with ReLU, (2) Replacing multi-head attention with single head attention. Interestingly, SimA has comparable performance even with single head attention and ReLU. Note that the ReLU version does not need any $exp(.)$ operation at the inference time. Execution time of SimA and other baselines are shown in Fig. 2.9 and Table 2.6.

	Model	params	FLOPs	Resolution	Softmax/#exp	Top1-Acc
CNN	ResNet18 [178]	12M	1.8B	224	0	69.8
Transformer	XCiT-T24/16 [23]	12M	2.3B	224	HD^2	79.4
Transformer	XCiT-T24/16 → SimA	12M	2.3B	224	0	79.8
Transformer	XCiT-T12/8 [23]	7M	4.8B	224	HD^2	79.7
Transformer	XCiT-T12/8 → SimA	7M	4.8B	224	0	79.4
CNN	ResNet50 RA [93]	25M	3.9B	224	0	77.6
	EfficientNet-B5 RA [93]	30M	9.9B	456	0	83.9
	RegNetY-4GF [355]	21M	4.0B	224	0	80.0
	ConvNeXt-T [290]	29M	4.5B	224	0	82.1
MLP	ResMLP-S24 [429]	30M	6.0B	224	0	79.4
	MS-MLP-T [535]	28M	4.9B	224	0	82.1
	Hire-MLP-S [162]	33M	4.2B	224	0	82.1
Hybrid	Twin-SVT-S [87]	24M	3.7B	224	HM^2N	81.7
	CvT-13 [478]	20M	4.5B	224	HN^2	81.6
	CvT-13 → SimA	20M	4.5B	224	0	81.4
Transformer	Swin-T [289]	29M	4.5B	224	HM^2N	81.3
	PVT-S [462]	24M	4.0B	224	HN^2/R	79.8
	T2T-ViT-14 [509]	21M	5.2B	224	HN^2	80.7
	CaiT-XS24* [435]	26M	19.3B	384	HN^2	84.1
	SOFT-S [293]	24M	3.3B	224	HN^2	82.2
	DeiT-S* [431]	22M	4.6B	224	HN^2	81.2
	XCiT-S12/16* [23]	26M	4.8B	224	HD^2	83.3
	DeiT-S [431]	22M	4.6B	224	HN^2	79.8
	XCiT-S12/16 [23]	26M	4.8B	224	HD^2	82.0
	DeiT-S → SimA	22M	4.6B	224	0	79.8
	XCiT-S12/16 → SimA	26M	4.8B	224	0	82.1
Multi-Head/GELU	DeiT-S → SimA	22M	4.6B	224	0	79.8
Multi-Head → Single-Head	DeiT-S → SimA	22M	4.6B	224	0	79.4
GELU → ReLU	DeiT-S → SimA	22M	4.6B	224	0	79.6

- CvT: We apply SimA to CvT [478], which is a SOTA hybrid convolution/transformer architecture. CvT has 3 stages. Each stage has a Convolution Token Embedding layer followed by transformer blocks. We use CvT-13 in our experiments which 13 blocks in total.

Results on ImageNet: We replace MSA and XCA blocks with our SimA block in DeiT, CvT and XCiT respectively, and train our models on ImageNet. Note that we train our models from scratch without distillation from a teacher. Results are in Table 2.4. In XCiT models, we get comparable results when

TABLE 2.5. **Linear Attention Comparison:** We compare SimA with previous linear attention methods introduced in NLP. We report ImageNet Top-1 validation accuracy. Note that the focus of these methods is to have linear attention with respect to the number of tokens while the main focus of SimA is to remove $exp(.)$ operation. * CosFormer is originally in NLP. We ran multiple versions of CosFormer with cosine re-weighting (multiple learning rates and weight decays) for the vision task, however, none of them converged. Moreover, CosFormer with cosine re-weighting requires 4× more FLOPs compared to SimA in multiplying Q, K, and V matrices. More details are in the appendix. Execution time of SimA and SOFT is shown in Fig. 2.9 and Table 2.6.

Model	params	FLOPs	Softmax/#exp	Top1-Acc
Transformer [446]	13M	3.9B	HN^2	79.1
Linformer [460]	13M	1.9B	HN	78.2
Performer [86]	13M	2.2B	ND	76.1
Nyströmformer [488]	13M	2.0B	HN	78.6
SOFT [293]	13M	1.9B	HN^2	79.3
XCiT-T20/16 → SimA	12M	1.9B	0	79.2
XCiT w/ Efficient Attention [392]	22M	4.8B	ND	80.9
CosFormer w/o re-weighting * [350]	22M	4.8B	0	76.1
XCiT-S12/16 → SimA	22M	4.8B	0	82.1

TABLE 2.6. **Execution Time Comparison:** We compare execution time of different attention blocks on 3 different edge devices. SimA has faster execution time in edge devices due to removing $exp(.)$ operation. To measure the effect of $exp(.)$ only, we fix the order of (QK^TV) product so that all models have the same dot product complexity. We set $N > D$ for top table and $N < D$ for the bottom table. These results are also shown in Figure 2.9.

$D = 64$ $N = 256$ $H = 8$ $N > D$			
Model	Execution Time (ms)		
	NVIDIA nano	Apple M1	Raspberry Pi 4
Vanilla Attention (DeiT)	177.4	35.8	180.6
ELU [225]	144.5	26.2	160.0
SimA	**63.6**	**20.4**	**51.9**
$D = 256$ $N = 64$ $H = 8$ $N < D$			
Model	Execution Time (ms)		
	NVIDIA nano	Apple M1	Raspberry Pi 4
SOFT [293]	145.9	36.0	177.4
XCA [392]	144.4	36.1	178.7
ELU [225]	105.2	26.4	160.1
SimA	**58.0**	**20.5**	**53.2**

replacing XCA block with SimA block. Compared to DeiT-S, our attention block performs on-par with DeiT-S. Moreover, our method with no Softmax layer, achieves comparable accuracy (0.2 point lower) compared to CvT-13. This suggests that one can replace attention block with SimA in these standard SOTA transformers without degrading their performance. Since SimA is $exp(.)$ free, it has the advantage over regular attention architectures in terms of efficiency and simplicity.

Comparison to Linear Attention: We compare SimA with other Linear Attention methods in NLP literature in Table 2.5. We train all methods with ImageNet-1K training set and report Top-1 accuracy on

TABLE 2.7. **Transfer to MS-COCO dataset:** Models with * are pretrained with a teacher on ImageNet. Swin-T has more parameters and Softmax overhead. XCiT-S12/8 has 4× more tokens. Our method is $exp(.)$ free, thus it is more efficient for high resolution images and high capacity models (Fig. 2.9). Execution time of SimA and XCA (XCiT) is shown in Fig. 2.9.

			Detection			Segmentation		
Backbone	params	$exp(.)$	AP^{box}	AP^{box}_{50}	AP^{box}_{75}	AP^{mask}	AP^{mask}_{50}	AP^{mask}_{75}
ResNet50 [178]	44.2M	✗	41.0	61.7	44.9	37.1	58.4	40.1
PVT-Small [462]	44.1M	✓	43.0	65.3	46.9	39.9	62.5	42.8
ViL-Small [524]	45.0M	✓	43.4	64.9	47.0	39.6	62.1	42.4
Swin-T [289]	47.8M	✓	46.0	68.1	50.3	41.6	65.1	44.9
XCiT-S12/16*	44.3M	✓	45.3	67.0	49.5	40.8	64.0	43.8
XCiT-S12/8*	43.1M	✓	47.0	68.9	51.7	42.3	66.0	45.4
XCiT-S12/16	44.3M	✓	45.0	66.7	48.9	40.5	63.6	43.2
XCiT-S12/16 → SimA	44.3M	✗	44.8	66.5	48.8	40.3	63.2	43.3

TABLE 2.8. **Self-Supervised Learning:** We train SimA attention block with DINO (SSL). Our method achieves performance comparable to transformer models with Softmax and trained for 100 epochs. Note that methods with different SSL task and higher number of epochs are not directly comparable. Execution time of SimA and XCA (XCiT) is shown in Fig. 2.9.

SSL Method	Model	params	epochs	$exp(.)$	FLOPs	Linear	k-NN
ISD [420]	ResNet50	25M	200	✗	3.9B	69.8	62.0
MoCo v2 [175]	ResNet50	25M	200	✗	3.9B	69.9	-
MSF [242]	ResNet50	25M	200	✗	3.9B	72.4	64.9
BYOL [160]	ResNet50	25M	1000	✗	3.9B	74.3	66.9
MoBY [486]	Swin-T	29M	300	✓	4.5B	75.0	–
DINO [59]	ResNet-50	23M	300	✗	4.1B	74.5	65.6
DINO [59]	ResMLP-S24	30M	300	✗	6.0B	72.8	69.4
DINO [59]	ViT-S/16	22M	300	✓	4.6B	76.1	72.8
DINO [59]	XCiT-S12/16	26M	300	✓	4.9B	77.8	76.0
DINO [59]	ViT-S/16	22M	100	✓	4.6B	74.0	69.3
DINO [59]	XCiT-S12/16	26M	100	✓	4.9B	75.8	71.6
DINO [59]	XCiT-S12/16 → SimA	26M	100	✗	4.9B	75.5	71.2

ImageNet-1K validation set. SimA has better or on-par accuracy compared to other methods. Additionally, SimA is $exp(.)$ free which is the main goal of this work.

2.2.4.2 Transfer To Object Detection and Semantic Segmentation

As shown in Figure 2.9, Table 2.6, and [405], softmax operation represents a large fraction of runtime in vision transformers, especially when the image resolution is high. In object detection and segmentation

tasks we usually forward high resolution images. We demonstrate the transferability of SimA to these dense prediction tasks by fine-tuning our ImageNet pretrained model on them.

Dataset: We use MS-COCO [279] dataset for these tasks. MS-COCO has 118K training images and 5K validation images with 80 categories. Images are annotated with bounding boxes and semantic segmentation masks.

Implementation Details: We follow [23, 65, 289] for the setup and implementation. We use our pre-trained model as the backbone of Mask RCNN [176]. Similar to [23], we use FPN [278] to extract features from layers 4, 6, 8 and 12 of the transformer. We use AdamW [292] optimizer with a learning rate of $1e-4$ and weight decay 0.05. We train our model for 36 epochs with batch size of 16 on 8 RTX2080Ti GPUs. Training takes 36 hours.

Results on MS-COCO: We compare our XCiT-S12/16 → SimA model with other vision transformers and ResNet in Table 2.7. We report the performance on the minival set. For a fair comparison, we limit the comparison to all models which are initialized with ImageNet1K pretrained backbones and trained with the same training time budget (3x schedule) on MS-COCO dataset. In comparison to other transformers, our method gets on-par performance while it is free of Softmax overhead on high resolution images or high capacity models (refer to Fig. 2.9).

2.2.4.3 Self-Supervised Learning

To show the generalizability of SimA, we train our SimA model on a pretext task for self-supervised learning (SSL). We use the non-contrastive task called DINO [59] for SSL pre-training. We train our model on ImageNet train set ($1.2M$) without the use of ground-truth labels. DINO training is relatively expensive since it requires forwarding multi-crop augmentation through two models. Due to limited resources, we train our model and the baselines for 100 epochs. To train our XCiT-S12/16 → SimA model with DINO, we follow the training configuration of XCiT-S12/16 from the official repository of DINO [58]. Similar to DINO, we use AdamW optimizer in PyTorch library with initial learning rate of 0.00025 with cosine scheduling. We use initial weight decay of 0.04 and increase it to 0.4 with cosine scheduling. We train for 100 epochs with minibatches of size 256. The training takes approximately 100 hours on four RTX-3090 GPUs. We use similar settings for training our method and the baseline (XCiT-S12/16).

Results of SSL training: Following [59, 240], we report k-NN and Linear evaluation metrics for evaluating the SSL models. For k-NN evaluation, we forward images of training and validation set through the frozen backbone and extract features. We report 20-NN on the validation set. For Linear evaluation, we

freeze the backbone and train a linear layer on extracted features from the frozen backbone and report Top-1 accuracy on the ImageNet validation set. We adopt a similar approach to DINO [59] for extracting features from XCiT architecture. We extract the classification tokens of the last two class attention layers and global average pooling of the last two regular attention layers. Each of those 4 vectors is of size 384. We concatenate them and train a linear layer of size 4×384 to 1000 classes of ImageNet1K. We use similar training settings as DINO to train a linear layer for both our method and the baseline (XCiT-S12/16). We train for 100 epochs with SGD optimizer and the following settings: learning rate: 0.001 with cosine scheduling, batch size: 1024, and weight decay: 0. Results are shown in Table 2.8. Our $exp(.)$ free method performs comparably with the baselines with 100 epochs of training.

2.2.4.4 Single-head vs Multi-head Attention

In the regular attention block, if a channel in Q and/or K has large values, that channel may dominate the dot product QK^T. We believe multi-head attention (MSA) mitigates this issue to some degree by containing the dominant channel in one head only so that the other heads can have reasonable effect in the final attention. In SimA, by doing ℓ_1 normalization of each channel in Q and K across tokens, different channels become more comparable in the dot product QK^T, so multi-head attention may not have a large effect anymore. To evaluate our hypothesis empirically, we train both DeiT-S → SimA and DeiT-S with single head attention only. Results are in Table. 2.9. Interestingly, we show that with single-head attention, our method gets comparable results (0.4 point lower) while the accuracy of DeiT-S drops by 2.8 points. This suggests that unlike the vanilla attention block, multi-head attention is not critically important in SimA, which leads to simplicity SimA even further.

TABLE 2.9. **Effect of Removing Multi-Head Attention:** In single head variation, our method degrades much less compared to DeiT probably due to normalization of Q and K.

Model	DeiT-S → SimA		DeiT-S	
Attention Heads	6 (Multi-Head)	1 (Single)	6 (Multi-Head)	1 (Single)
ImageNet Top-1 acc.	79.8	79.4 (-0.4)	79.8	77.0 (-2.8)

2.2.4.5 Replacing GELU with ReLU

Similar to Softmax function, GELU activation function also uses $exp(.)$ operation, which is costly. We replace all GELU activation functions in DeiT-S → SimA with ReLU. We observe that DeiT-S → SimA with ReLU gets accuracy of 79.6 which is only 0.2 points lower than DeiT-S → SimA with GELU activation

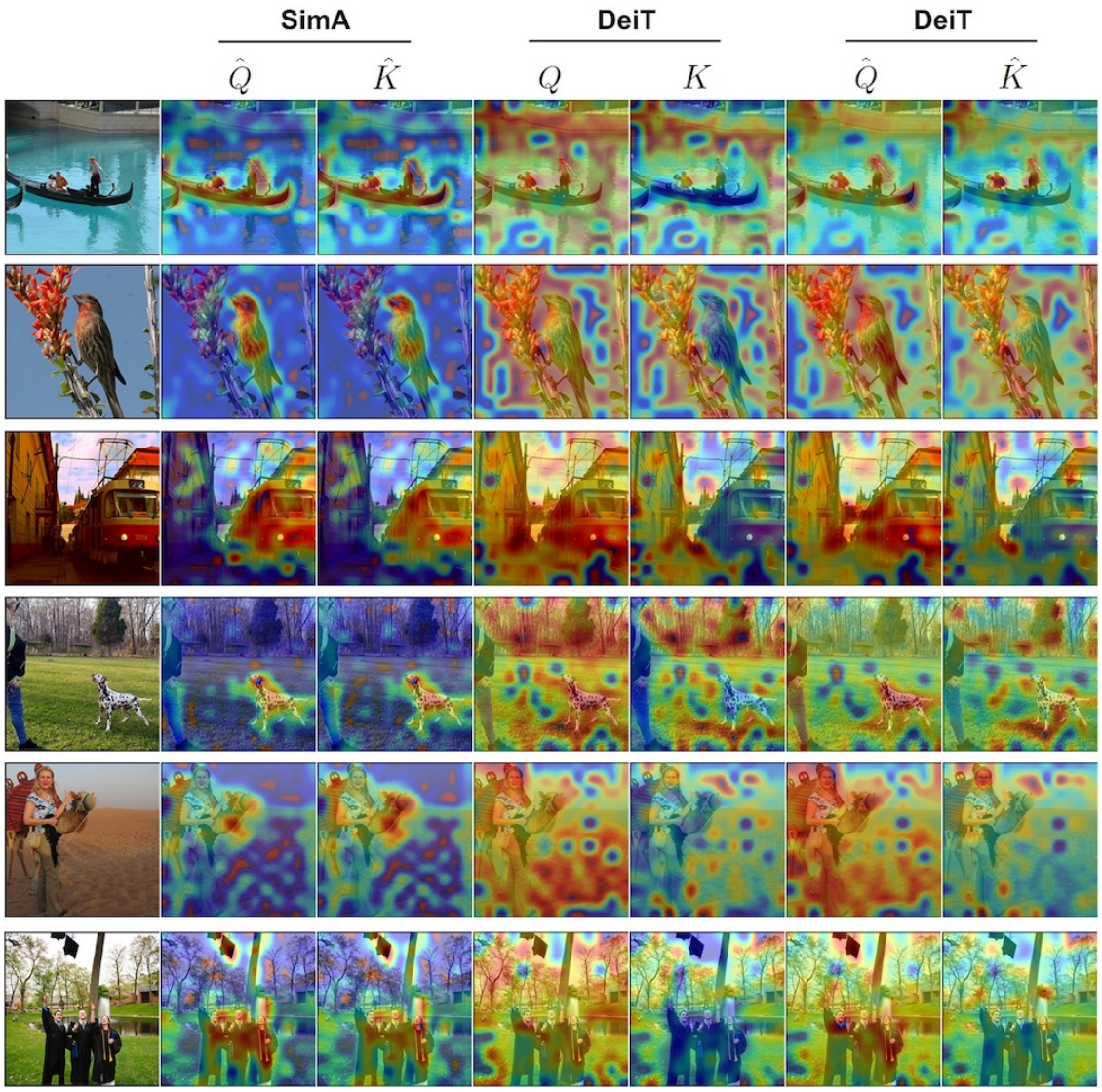


FIGURE 2.11. **Our method (SimA):** Standard attention passes QK^T through Softmax before multiplying with V. However, we multiply $\hat{Q}\hat{K}^T$ directly with V. Hence, in our case, the magnitude of $\hat{Q}\hat{K}^T$ should identify which tokens are more important (their information flows to the next layers). We show that this magnitude is correlated with the importance of tokens. We extract $\hat{Q}$ and $\hat{K}$ from layer 12 of transformer. We get ℓ_2-norm of each token for $\hat{Q}$ and $\hat{K}$, normalize it to range [0,1] and overlay it as a heatmap on the image. We show the same visualization for DeiT in the supplementary for completeness.

function. Note that SimA with ReLU does not use any $exp(.)$ operation at the inference time, leading to further efficiency of the model. Results are in Table 2.4 (yellow rows).

2.2.4.6 Effect of ℓ_1 Normalization

To see the effect of ℓ_1 normalization, we train our model without normalizing Q and K. We use XCiT-S12/16 → SimA with the same hyperparameters as our main experiment in Section 2.2.4.1. Note that without

normalization, the range of QK^T can be from $-\infty$ to $+\infty$. None of our several trials converged as the training becomes unstable and results in a frequent NaN loss. Moreover, we replace ℓ_1 with ℓ_2 normalization, results in 2.9 points drop in accuracy (82.1% vs 79.2%).

2.2.4.7 Visualization

Since in SimA, we multiply the dot product $\hat{Q}\hat{K}^T$ directly with V without any Softmax layer, the $\hat{Q}$ and $\hat{K}$ with larger magnitute will have more effect in the output of the block. Hence, we believe this magnitude can highlight the important tokens or image regions. This can be seen as a form of explanation or saliency map. First, we extract $\hat{Q}$ and $\hat{K}$ in the last layer of transformer (layer 12). Then, we calculate the ℓ_2-norm of $\hat{Q}$ along the channel dimension to get a single non-negative scalar for each token. We reshape this $N \times 1$ vector to the image shape, up-sample it to original image size, normalize it to range $[0, 1]$, and overlay it on the image as a heatmap. We repeat the same for $\hat{K}$. As shown qualitatively in Fig. 2.11, such a visualization highlights the important regions of the image.

2.2.5 Conclusion

We introduced SimA, a simple attention block that does not involve $exp(.)$ operation, to reduce the computational cost of transformers particularly at edge devices. SimA performs normalization on key and query matrices before multiplying them, enabling dynamically switching between $O(DN^2)$ or $O(D^2N)$ depending on the number of tokens (e.g., image resolution). Our extensive experiments show that while reducing the cost of inference, SimA achieves on-par results compared to SOTA methods on various benchmarks including ImageNet classification, MS-COCO object detection and segmentation, and self-supervised learning. Moreover, a single-head variation of SimA, which is even simpler, achieves results on-par with SOTA multi-head attention models. We believe SimA can encourage research in this direction leading to easier adoption of transformers on edge devices with limited resources.

2.3 CompRess: Self-Supervised Learning by Compressing Representations

Self-supervised learning aims to learn good representations with unlabeled data. Recent works have shown that larger models benefit more from self-supervised learning than smaller models. As a result, the gap between supervised and self-supervised learning has been greatly reduced for larger models. In this work, instead of designing a new pseudo task for self-supervised learning, we develop a model compression method to compress an already learned, deep self-supervised model (teacher) to a smaller one (student). We train the student model so that it mimics the relative similarity between the datapoints in the teacher's embedding space. For AlexNet, our method outperforms all previous methods including the fully supervised model on ImageNet linear evaluation (59.0% compared to 56.5%) and on nearest neighbor evaluation (50.7% compared to 41.4%). To the best of our knowledge, this is the first time a self-supervised AlexNet has outperformed supervised one on ImageNet classification. Our code is available here: https://github.com/UMBCvision/CompRess

2.3.1 Introduction

Supervised deep learning needs lots of annotated data, but the annotation process is particularly expensive in some domains like medical images. Moreover, the process is prone to human bias and may also result in ambiguous annotations. Hence, we are interested in self-supervised learning (SSL) where we learn rich representations from unlabeled data. One may use these learned features along with a simple linear classifier to build a recognition system with small annotated data. It is shown that SSL models trained on ImageNet without labels outperform the supervised models when transferred to other tasks [70, 175].

Some recent self-supervised learning algorithms have shown that increasing the capacity of the architecture results in much better representations. For instance, for SimCLR method [70], the gap between supervised and self-supervised is much smaller for ResNet-50x4 compared to ResNet-50 (also shown in Figure 2.12). Given this observation, we are interested in learning better representations for small models by compressing a deep self-supervised model.

In edge computing applications, we prefer to run the model (e.g., an image classifier) on the device (e.g., IoT) rather than sending the images to the cloud. During inference, this reduces the privacy concerns, latency, power usage, and cost. Hence, there is need for rich, small models. Compressing SSL models goes beyond that and reduces the privacy concerns at the training time as well. For instance, one can download a

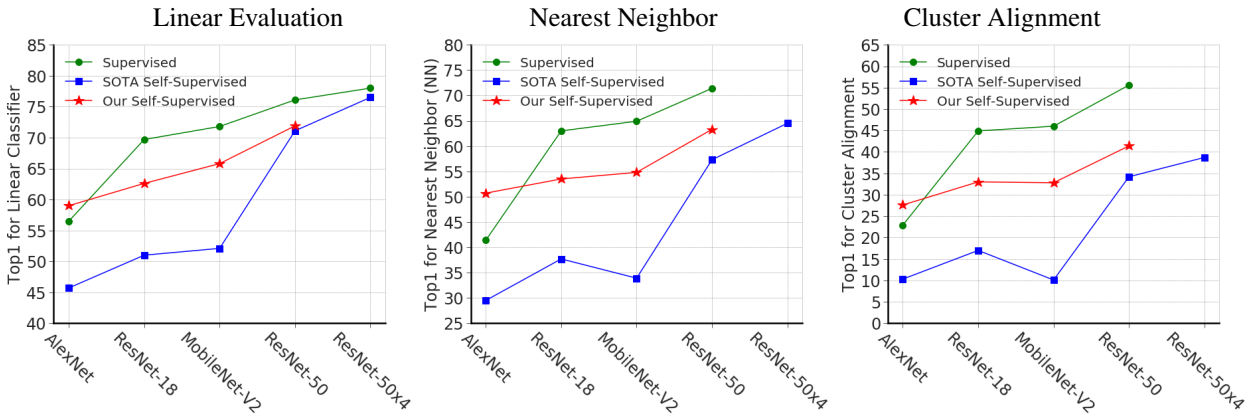


FIGURE 2.12. **ImageNet Evaluation:** We compare "Ours-1q" self-supervised model with supervised and SOTA self-supervised models on ImageNet using linear classification (left), nearest neighbor (middle) and cluster alignment (right) evaluations. Our AlexNet model outperforms the supervised counterpart on all evaluations. This model is compressed from ResNet-50x4 trained with SimCLR method using unlabeled ImageNet. All models have seen ImageNet images only. All SOTA SSL models are MoCo except ResNet-50x4 that is SimCLR. The teacher for our AlexNet and ResNet-50 is SimCLR ResNet-50x4 and for ResNet18 and MobileNet-V2 is MoCo ResNet-50.

rich self-supervised MobileNet model that can generalize well to other tasks and finetune it on his/her own data without sending any data to the cloud for training.

Since we assume our teacher has not seen any labels, its output is an embedding rather than a probability distribution over some categories. Hence, standard model distillation methods [188] cannot be used directly. One can employ a nearest neighbor classifier in the teacher space by calculating distances between an input image (query) and all datapoints (anchor points) and then converting them to probability distribution. Our idea is to transfer this probability distribution from the teacher to the student so that for any query point, the student matches the teacher in the ranking of anchor points.

Traditionally, most SSL methods are evaluated by learning a linear classifier on the features to perform a downstream task (e.g., ImageNet classification). However, this evaluation process is expensive and has many hyperparameters (e.g., learning rate schedule) that need to be tuned as one set of parameters may not be optimal for all SSL methods. We believe a simple nearest neighbor classifier, used in some recent works [480, 502, 545], is a better alternative as it has no parameters, is much faster to evaluate, and still measures the quality of features. Hence, we use this evaluation extensively in our experiments. Moreover, inspired by [216], we use another related evaluation by measuring the alignment between k-means clusters and image categories.

Our extensive experiments show that our compressed SSL models outperform state-of-the-art compression methods as well as state-of-the-art SSL counterparts using the same architecture on most downstream

tasks. Our AlexNet model, compressed from ResNet-50x4 trained with SimCLR method, outperforms standard supervised AlexNet model on linear evaluation (by 2 point), in nearest neighbor (by 9 points), and in cluster alignment evaluation (by 4 points). This is interesting as all parameters of the supervised model are already trained on the downstream task itself but the SSL model and its teacher have seen only ImageNet without labels. To the best of our knowledge, this is the first time an SSL model performs better than the supervised one on the ImageNet task itself instead of transfer learning settings.

2.3.2 Related work

In self-supervised learning for images, we learn rich features by solving a pretext task that needs unlabeled data only. The pseudo task may be colorization [526], inpainting [336], solving Jigsaw puzzles [323], counting visual primitives [324], and clustering images [56].

Contrastive learning: Our method is related to contrastive learning [31,165,183,189,424,445,480,545] where the model learns to contrast between the positive and lots of negative pairs. The positive pair is from the same image and model but different augmentations in [70, 175, 426] and from the same image and augmentation but different models (teacher and student) in [425]. Our method uses soft probability distribution instead of positive/negative classification [545], and does not couple the two embeddings (teacher and student) directly [425] in "Ours-2q" variant. Contrastive learning is improved with a more robust memory bank in [175] and with temperature and better image augmentations in [70]. Our ideas are related to exemplar CNNs [114, 302], but used for compression.

Model compression: The task of training a simpler student to mimic the output of a complex teacher is called model compression in [47] and knowledge distillation in [188]. In [188], the softened class probabilities from the teacher are transferred to the student by reducing KL divergence. The knowledge in the hidden activations of intermediate layers of the teacher is transferred by regressing linear projections [373], aggregation of feature maps [516], and gram matrices [500]. Also, knowledge at the final layer can be transferred in different ways [20, 29, 32, 141, 188, 231, 332, 334, 340, 418, 425, 444, 474, 503]. In [20, 425] distillation is formulated as maximization of information between teacher and student.

Similarity-based distilation: Pairwise similarity based knowledge distillation has been used along with supervised teachers. [332, 340, 444] use supervised loss in distillation. [334] is probably the closest to our setting which does not use labels in the distillation step. We are different as we use memory bank and SoftMax along with temperature, and also apply that to compressing self-supervised models in large scale. We compare with a reproduced variant of [334] in the experiments (subsection 2.3.4.3).

Model compression for self-supervision: Standard model compression techniques either directly use the output of supervised training [188] or have a supervised loss term [332, 425] in addition to the compression loss term. Thus, they cannot be directly applied to compress self-supervised models. In [325], the knowledge from the teacher is transferred to the student by first clustering the embeddings from teacher and then training the student to predict the cluster assignments. In [495], the method of [325] is applied to regularize self-supervised models.

2.3.3 Method

Our goal is to train a deep model (e.g. ResNet-50) using an off-the-shelf self-supervised learning algorithm, and then compress it to a less deep model (e.g., AlexNet) while preserving the discriminative power of the features. Figure 2.13 shows our method.

Assuming a frozen teacher embedding $t(x) \in R^N$ with parameters θ_t that maps an image x into an N-D feature space, we want to learn the student embedding $s(x) \in R^M$ with parameters θ_s that mimics the same behavior as $t(x)$ if used for a downstream supervised task e.g., image classification. Note that the teacher and student may use architectures from different families, so we do not necessarily want to couple them together directly. Hence, we transfer the similarity between data points from the teacher to the student rather than their final prediction.

For simplicity, we use $t_i = t(x_i)$ for the embedding of the model $t(x)$ on the input image x_i normalized by ℓ_2 norm. We assume a random set of the training data $\{x_j\}_{j=1}^n$ are the *anchor* points and embed them using both teacher and student models to get $\{t_j^a\}_{j=1}^n$ and $\{s_j^a\}_{j=1}^n$. Given a *query* image q_i and its embeddings t_i^q for teacher and s_i^q for student, we calculate the pairwise similarity between t_i^q and all anchor embeddings $\{t_j^a\}_{j=1}^n$, and then optimize the student model so that in the student's embedding space, the query s_i^q has the same relationship with the anchor points $\{s_j^a\}_{j=1}^n$.

To measure the relationship between the query and anchor points, we calculate their cosine similarity. We convert the similarities to the form of a probability distribution over the anchor points using SoftMax operator. For the teacher, the probability of the i-th query for the j-th anchor point is:

$$p_j^i(t) = \frac{\exp(t_i^{qT} t_j^a / \tau)}{\sum_{k=1}^n \exp(t_i^{qT} t_k^a / \tau)}$$

where τ is the temperature hyperparamater. Then, we define the loss for a particular query point as the KL divergence between the probabilities over all anchor points under the teacher and student models, and we sum this loss over all query points:

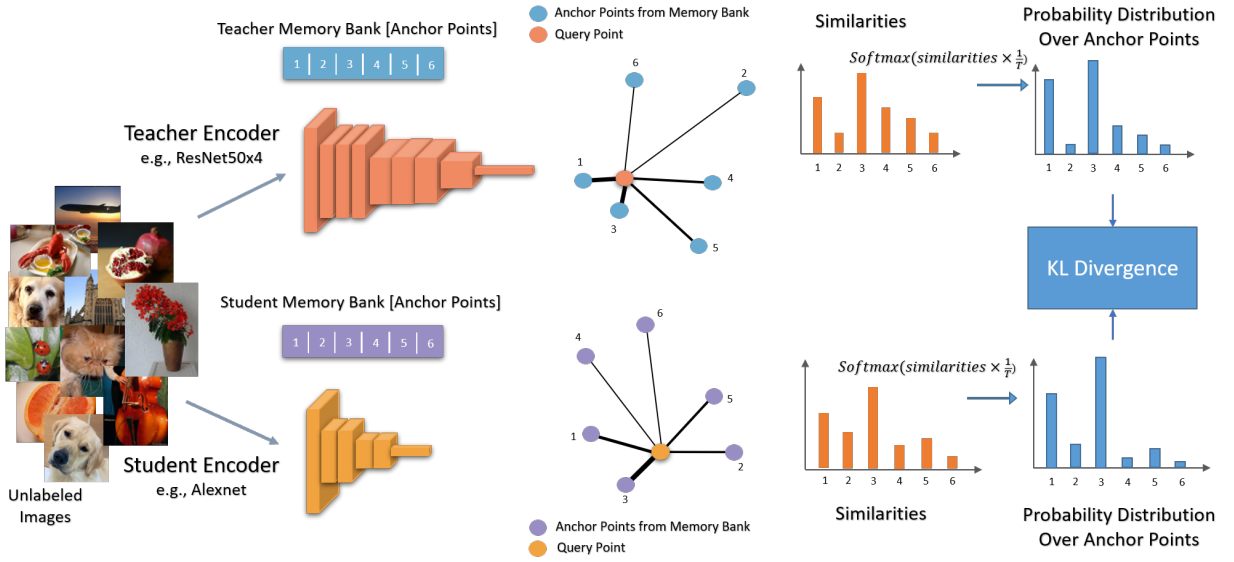


FIGURE 2.13. **Our compression method:** The goal is to transfer the knowledge from the self-supervised teacher to the student. For each image, we compare it with a random set of data points called anchors and obtain a set of similarities. These similarities are then converted into a probability distribution over the anchors. This distribution represents each image in terms of its nearest neighbors. Since we want to transfer this knowledge to the student, we get the same distribution from the student as well. Finally, we train the student to minimize the KL divergence between the two distributions. Intuitively, we want each data point to have the same neighbors in both teacher and student embeddings. This illustrates Ours-2q method. For Ours-1q, we simply remove the student memory bank and use the teacher's anchor points for the student as well.

$$L(t,s) = \sum_i KL(p^i(t)||p^i(s))$$

where $p^i(s)$ is the probability distribution of query i over all anchor points on the student network. Finally, since the teacher is frozen, we optimize the student by solving:

$$\arg\min_{\theta_s} L(t,s) = \arg\min_{\theta_s} \sum_{i,j} -p_j^i(t).log(p_j^i(s))$$

Memory bank: One may use the same minibatch for both query and anchor points by excluding the query from each set of anchor points. However, we need a large set of anchor points (ideally the whole training set) so that they have large variation to cover the neighborhood of any query image. Our experiments verify that using the minibatch of size 256 for anchor points is not enough for learning rich representations. This is reasonable as ImageNet has 1000 categories so the query may not be close to any anchor point in a minibatch of size 256. However, it is computationally expensive to process many images in a single iteration due to limited computation and memory. Similar to [480], we maintain a memory bank of anchor points from several most recent iterations. We use momentum contrast [175] framework for implementing memory bank

for the student. However, unlike [175], we find that our method is not affected by the momentum parameter which requires further investigation. Since the teacher is frozen, we implement its memory bank as a simple FIFO queue.

Temperature parameter: Since the anchor points have large variation covering the whole dataset, many of them may be very far from the query image. We use a small temperature value (less than one) since we want to focus mainly on transferring the relationships from the close neighborhoods of the query rather than faraway points. Note that this results in sharper probabilities compared to $\tau = 1$. We show that $\tau = 1$ degrades the results dramatically. The temperature value acts similarly to the kernel width in kernel density estimation methods.

Student using teacher's memory bank: So far, we assumed that the teacher and student embeddings are decoupled, so used a separate memory bank (queue) for each. We call this method **Ours-2q**. However, we may use the teacher's anchor points in calculating the similarity for the student model. This way, the model may learn faster and be more stable in the initial stages of learning, since the teacher anchor points are already mature. We call this variation **Ours-1q** in our experiments. Note that in "Ours-1q" method, we do not use momentum since the teacher is constant.

Caching the teacher embeddings: Since we are interested in using very deep models (e.g., ResNet-50x4) as the teacher, calculating the embeddings for the teacher is expensive in terms of both computation and memory. Also, we are not optimizing the teacher model. Hence, for such large models, we can cache the results of the teacher on all images of the dataset and keep them in the memory. This caching has a drawback that we cannot augment the images for the teacher, meaning that the teacher sees exact same images in all epochs. However, since the student still sees augmented images, it is less prone to overfitting. On the other hand, this caching may actually help the student by encouraging the relationship between the query and anchor points to be close even under different augmentations, hence, improving the representation in a way similar to regular contrastive learning [175, 480]. In our experiments, we realize that caching degrades the results by only a small margin while is much faster and efficient in learning. We use caching when we compress from ResNet-50x4 to AlexNet.

2.3.4 Experiments

We use different combinations of architectures as student-teacher pairs (listed in Table 2.10). We use three teachers : (a) ResNet-50 model which is trained using MoCo-v2 method for 800 epochs [76], (b) ResNet-50 trained with SwAV [57] for 800 epochs, and (c) ResNet-50x4 model which is trained using SimCLR method for 1000 epochs [70]. We use the officially published weights of these models [12, 14,

17]. For supervised models, we use the official PyTorch weights [16]. We use ImageNet (ILSVRC2012) [378] without labels for all self-supervised and compression methods, and use various datasets (ImageNet, PASCAL-VOC [121], Places [539], CUB200 [471], and Cars196 [243]) for evaluation.

Here, we report the implementation details for Ours-2q and Ours-1q compression methods. The implementation details for all baselines and transfer experiments are included in the appendix. We use PyTorch along with SGD (weight decay=1e−4, learning rate=0.01, momentum=0.9, epochs=130, and batch size=256). We multiply learning rate by 0.2 at epochs 90 and 120. We use standard ImageNet data augmentation found in PyTorch. Compressing from ResNet-50x4 to ResNet-50 takes ~100 hours on four Titan-RTX GPUs while compressing from ResNet-50 to ResNet-18 takes ~90 hours on two 2080-TI GPUs. We adapt the unofficial implementation of MoCo [175] in [2] to implement memory bank for our method. We use memory bank size of $128,000$ and set moving average weight for key encoder to 0.999. We use the temperature of 0.04 for all experiments involving SimCLR ResNet-50x4 and MoCo ResNet-50 teachers. We pick these values based on the ablation study done for the temperature parameter in subsection 2.3.5. For SwAV ResNet-50 teacher, we use a temperature of 0.007 since we find that it works better than 0.04.

2.3.4.1 Evaluation Metrics

Linear classifier (Linear): We treat the student as a frozen feature extractor and train a linear classifier on the labeled training set of ImageNet and evaluate it on the validation set with Top-1 accuracy. To reduce the computational overhead of tuning the hyperparameters per experiment, we standardize the Linear evaluation as following. We first normalize the features by ℓ_2 norm, then shift and scale each dimension to have zero mean and unit variance. For all linear layer experiments, we use SGD with lr=0.01, epochs=40, batch size=256, weight decay=1e−4, and momentum=0.9. At epochs 15 and 30, the lr is multiplied by 0.1.

Nearest Neighbor (NN): We also evaluate the student representations using nearest neighbor classifier with cosine similarity. We use FAISS GPU library [9] to implement it. This method does not need any parameter tuning and is very fast (~25 minutes for ResNet-50 on a single 2080-TI GPU)

Cluster Alignment (CA): The goal is to measure the alignment between clusters of our SSL representations with visual categories, e.g., ImageNet categories. We use k-means (with k=1000) to cluster our self-supervised features trained on unlabeled ImageNet, map each cluster to an ImageNet category, and then evaluate on ImageNet validation set. In order to map clusters to categories, we first calculate the alignment between all (cluster- category) pairs by calculating the number of common images divided by the size of cluster. Then, we find the best mapping between clusters and categories using Hungarian algorithm [250]

that maximizes total alignment. This labels the clusters. Then, we report the classification accuracy on the validation set. This setting is similar to the object discovery setting in [216]. In Figure 2.14 (c), we show some random images from random clusters where images inside each cluster are semantically related.

2.3.4.2 Baselines:

Contrastive Representation Distillation (CRD): CRD [425] is an information maximization based SOTA method for distillation that includes a supervised loss term. It directly compares the embeddings of teacher and student as in a contrastive setting. We remove the supervised loss in our experiments.

Cluster Classification (CC): Cluster Classification [325] is an unsupervised knowledge distillation method that improves self-supervised learning by quantizing the teacher representations. This is similar to the recent work of ClusterFit [495].

Regression (Reg): We implement a modified version of [373] that regresses only the embedding layer features [503]. Similar to [373, 503], we add a linear projection head on top of the student to match the embedding dimension of the teacher. As noted in CRD [425], transferring knowledge from all intermediate layers does not perform well since the teacher and student may have different architecture styles. Hence, we use the regression loss only for the embedding layer of the networks.

Regression with BatchNorm (Reg-BN): We realized that Reg does not perform well for model compression. We suspect the reason is the mismatch between the embedding spaces of the teacher and student networks. Hence, we added a non-parametric Batch Normalization layer for the last layer of both student and teacher networks to match their statistics. The BN layer uses statistics from the current minibatch only (element-wise whitening). Interestingly, this simple modified baseline is better than other sophisticated baselines for model compression.

2.3.4.3 Experiments Comparing Compression Methods

Evaluation on full ImageNet: We train the teacher on unlabeled ImageNet, compress it to the student, and evaluate the student using ImageNet validation set. As shown in Table 2.10, our method outperforms other distillation methods on all evaluation benchmarks. For a fair comparison, on ResNet-18, we trained MoCo for 1,000 epochs and got 54.5% in Linear and 41.1% in NN which does not still match our model. Also, a variation of our method (MoCo R50 to R18) without *SoftMax*, temperature, and memory bank (similar to [334]) results in 53.6% in Linear and 42.3% in NN. To evaluate the effect of the teacher's SSL

TABLE 2.10. **Comparison of distillation methods on full ImageNet:** Our method is better than all compression methods for various teacher-student combinations and evaluation benchmarks. In addition, as reported in Table 2.14 and Figure 2.12, when we compress ResNet-50x4 to AlexNet, we get **59.0%** for Linear, **50.7%** for Nearest Neighbor (NN), and **27.6%** for Cluster Alignment (CA) which outperforms the supervised model. On NN, our ResNet-50 is only 1 point worse than its ResNet-50x4 teacher. Note that models below the teacher row use the student architecture. Since a forward pass through the teacher is expensive for ResNet-50x4, we do not compare with CRD, Reg, and Reg-BN.

Teacher	MoCo ResNet-50			MoCo ResNet-50			MoCo ResNet-50			ResNet-50x4		
Student	AlexNet			ResNet-18			MobileNet-V2			ResNet-50		
	Linear	NN	CA	Linear	NN	CA	Linear	NN	CA	Linear	NN	CA
Teacher	70.8	57.3	34.2	70.8	57.3	34.2	70.8	57.3	34.2	75.6	64.5	38.7
Random-init	10.7	3.2	1.0	5.5	1.0	0.8	5.4	0.7	0.7	5.1	0.2	0.5
Supervised	56.5	41.4	22.9	69.8	63.0	44.9	71.9	64.9	46.0	76.2	71.4	55.6
CC [325]	46.4	31.6	13.7	61.1	51.1	25.2	59.2	50.2	24.7	68.9	55.6	26.4
CRD [425]	54.4	36.9	14.1	58.4	43.7	17.4	54.1	36.0	12.0	-	-	-
Reg	49.9	35.6	9.5	52.2	41.7	25.6	48.0	38.6	25.4	-	-	-
Reg-BN	56.1	42.8	22.3	58.2	47.3	27.2	62.3	48.7	27.0	-	-	-
Ours-2q	56.4	**48.4**	**33.3**	61.7	53.4	**34.7**	63.0	54.4	**35.5**	71.0	63.0	41.1
Ours-1q	**57.5**	48.0	27.0	**62.6**	**53.5**	33.0	**65.8**	**54.8**	32.8	**71.9**	**63.3**	**41.4**

TABLE 2.11. **Comparison of distillation methods on full ImageNet for SwAV ResNet-50 (teacher) to ResNet-18 (student):**. Note that SwAV (concurrent work) [57] is different from MoCo and SimCLR in that it performs contrastive learning through online clustering.

Method	Linear	NN	CA
Teacher	75.6	60.7	27.6
Supervised	69.8	63.0	44.9
CRD	58.2	44.7	16.9
CC	60.8	51.0	22.8
Reg-BN	60.6	47.6	20.8
Ours-2q	62.4	53.7	**26.7**
Ours-1q	**65.6**	**56.0**	26.3

TABLE 2.12. **NN evaluation for ImageNet with fewer labels:** We report NN evaluation on validation data using small training data (both ImageNet) for ResNet-18 compressed from MoCo ResNet-50. For 1-shot, we report the standard deviation over 10 runs.

Model	1-shot	1%	10%
Supervised (entire labeled ImageNet)	29.8 (±0.3)	48.5	56.8
CC [325]	16.3 (±0.3)	31.6	41.9
CRD [425]	11.4 (±0.3)	23.3	33.6
Reg-BN	21.5 (±0.1)	33.4	40.1
Ours-2q	**29.0** (±0.3)	**41.2**	**47.6**
Ours-1q	26.5 (±0.3)	39.6	47.2

method, in Table 2.11, we use SwAV ResNet-50 as the teacher and compress it to ResNet-18. We still get better accuracy compared to other distillation methods.

Evaluation on smaller ImageNet: We evaluate our representations by a NN classifier using only 1%, 10%, and only 1 sample per category of ImageNet. The results are shown in Table 2.12. For 1-shot, "Ours-2q" model achieves an accuracy close to the supervised model which has seen all labels of ImageNet in learning the features.

Transfer to CUB200 and Cars196: We transfer AlexNet student models to the task of image retrieval on CUB200 [471] and Cars196 [243] datasets. We evaluate on these tasks without any fine-tuning. The

TABLE 2.13. **Transfer to CUB200 and Cars196:** We train the features on unlabeled ImageNet, freeze the features, and return top k nearest neighbors based on cosine similarity. We evaluate the recall at different k values (1, 2, 4, and 8) on the validation set.

Method		CUB200				Cars196			
AlexNet	Teacher	R@1	R@2	R@4	R@8	R@1	R@2	R@4	R@8
Sup. on ImageNet	-	33.5	45.5	59.2	71.9	26.6	36.3	45.9	57.8
CRD [425]	ResNet-50	16.6	25.9	36.3	48.7	20.9	28.2	37.7	48.9
Reg-BN	ResNet-50	16.8	25.5	36.2	48.0	20.9	29.0	38.5	49.7
CC	ResNet-50	**23.2**	32.5	**45.1**	**58.2**	**23.7**	31.4	41.1	52.4
Ours-2q	ResNet-50	23.1	**33.0**	**45.1**	58.0	23.6	**32.8**	**42.9**	**54.9**
Ours-1q	ResNet-50	22.7	31.9	43.2	55.8	22.5	30.6	40.4	52.3
CC	ResNet-50x4	23.6	33.6	44.9	58.4	25.4	33.2	43.2	54.3
Ours-2q	ResNet-50x4	**26.5**	**37.0**	**49.4**	**62.4**	**28.4**	**38.5**	**48.7**	**60.4**
Ours-1q	ResNet-50x4	21.9	32.4	43.2	55.9	25.0	34.2	45.1	57.3

results are shown in Table 2.13. Surprisingly, for the combination of Cars196 dataset and ResNet-50x4 teacher, our model even outperforms the ImageNet supervised model. Since in "Ours-2q", the student embedding is less restricted and does not follow the teacher closely, the student may generalize better compared to "Ours-1q" method. Hence, we see better results for "Ours-2q" on almost all transfer experiments. This effect is similar to [325, 495].

2.3.4.4 Experiments Comparing Self-Supervised Methods

Evaluation on ImageNet: We compare our features with SOTA self-supervised learning methods on Table 2.14 and Figure 2.12. Our method outperforms all baselines on all small capacity architectures (AlexNet, MobileNet-V2, and ResNet-18). On AlexNet, it outperforms even the supervised model. Table 2.15 shows the results of linear classifier using only 1% and 10% of ImageNet for ResNet-50.

Transferring to Places: We evaluate our intermediate representations learned from unlabeled ImageNet on Places scene recognition task. We train linear layers on top of intermediate representations similar to [155]. Details are in the appendix. The results are shown in Table 2.14. We find that our best layer performance is better than that of a model trained with ImageNet labels.

Transferring to PASCAL-VOC: We evaluate AlexNet compressed from ResNet-50x4 on PASCAL-VOC classification and detection tasks in Table 2.16. For classification task, we only train a linear classifier on top of frozen backbone which is in contrast to the baselines that finetune all layers. For object detection, we use the Fast-RCNN [150] as used in [149, 325] to finetune all layers.

TABLE 2.14. **Linear evaluation on ImageNet and Places:** Comparison with SOTA self-supervised methods. We pick the best layer to report the results that is written in parenthesis: 'f7' refers to 'fc7' layer and 'c4' refers to 'conv4' layer. R50x4 refers to the teacher that is trained with SimCLR and R50 to the teacher trained with MoCo. On ResNet-50, our model, that is compressed from SimCLR R50x4, is better than SimCLR itself, but worse than SwAV, BYOL, and InfoMin which are concurrent works. $*$ refers to 10-crop evaluation. † denotes concurrent methods.

Method	Ref	ImageNet top-1	Places top-1
AlexNet			
Sup. on ImageNet	-	56.5 (f7)	39.4 (c4)
Inpainting [336]	[502]	21.0 (c3)	23.4 (c3)
BiGAN [109]	[325]	29.9 (c4)	31.8 (c3)
Colorization [526]	[149]	31.5 (c4)	30.3 (c4)
Context [108]	[149]	31.7 (c4)	32.7 (c4)
Jigsaw [323]	[149]	34.0 (c3)	35.0 (c3)
Counting [324]	[325]	34.3 (c3)	36.3 (c3)
SplitBrain [527]	[325]	35.4 (c3)	34.1 (c4)
InstDisc [480]	[480]	35.6 (c5)	34.5 (c4)
CC+Vgg+Jigsaw [325]	[325]	37.3 (c3)	37.5 (c3)
RotNet [149]	[149]	38.7 (c3)	35.1 (c3)
Artifact [215]	[135]	38.9 (c4)	37.3 (c4)
AND [203]	[502]	39.7 (c4)	-
DeepCluster [56]	[56]	39.8 (c4)	37.5 (c4)
LA* [545]	[545]	42.4 (c5)	40.3 (c4)
CMC [424]	[502]	42.6 (c5)	-
AET [523]	[502]	44.0 (c3)	37.1 (c3)
RFDecouple [135]	[135]	44.3 (c5)	38.6 (c5)
SeLa+Rot+aug [502]	[502]	44.7 (c5)	37.9 (c4)
MoCo	-	45.7 (f7)	36.6 (c4)
Ours-2q (from R50x4)	-	57.6 (f7)	**40.4** (c5)
Ours-1q (from R50x4)	-	**59.0** (f7)	40.3 (c5)

Method	Ref	ImageNet top-1
ResNet-18		
Sup. on ImageNet	-	69.8 (L5)
InstDisc [480]	[480]	44.5 (L5)
LA* [545]	[545]	52.8 (L5)
MoCo	-	54.5 (L5)
Ours-2q (from R50)	-	61.7 (L5)
Ours-1q (from R50)	-	**62.6** (L5)
ResNet-50		
Sup. on ImageNet	-	76.2 (L5)
InstDisc [480]	[480]	54.0 (L5)
CF-Jigsaw [495]	[495]	55.2 (L4)
CF-RotNet [495]	[495]	56.1 (L4)
LA * [545]	[545]	60.2 (L5)
SeLa [502]	[502]	61.5 (L5)
PIRL [313]	[313]	63.6 (L5)
SimCLR [70]	[70]	69.3 (L5)
MoCo [76]	[76]	71.1 (L5)
InfoMin† [426]	[426]	73.0 (L5)
BYOL† [160]	[160]	74.3 (L5)
SwAV† [57]	[57]	**75.3** (L5)
Ours-2q (from R50x4)	-	71.0 (L5)

2.3.5 Ablation Study

To speed up the ablation study, we use 25% of ImageNet (randomly sampled ~320k images) and cached features of MoCo ResNet-50 as a teacher to train ResNet-18 student. For temperature ablation study, the memory bank size is 128k and for memory bank ablation study, the temperature is 0.04. All ablations were performed with "Ours-2q" method.

Temperature: The results of varying temperature between 0.02 and 1.0 are shown in Figure 2.14(a). We find that the optimal temperature is 0.04, and the student gets worse as the temperature gets closer to 1.0. We believe this happens since a small temperature focuses on close neighborhoods by sharpening the probability distribution. A similar behavior is also reported in [70]. As opposed to the other similarity based distillation methods [332, 334, 340, 444], by using small temperature, we focus on the close neighborhood of a data point which results in an improved student.

Size of memory bank: Intuitively, larger number of anchor points should capture more details about the geometry of the teacher's embedding thus resulting in a student that approximates the teacher more closely.

TABLE 2.15. **Evaluation of ResNet-50 features on smaller set of ImageNet:** ResNet-50x4 is used as the teacher. Unlike other methods that fine-tune the whole network, we only train the last layer. Interestingly, despite fine-tuning fewer parameters, our method achieves better results on the 1% dataset. This demonstrates that our method can produce more data-efficient models. $*$ denotes concurrent methods.

Method	Top-1		Top-5	
	1%	10%	1%	10%
Supervised	25.4	56.4	48.4	80.4
InstDisc [480]	-	-	39.2	77.4
PIRL [313]	-	-	57.2	83.8
SimCLR [70]	48.3	65.6	75.5	87.8
BYOL* [160]	53.2	68.8	78.4	89.0
SwAV* [57]	53.9	**70.2**	78.5	**89.9**
Only the linear layer is trained.				
Ours-2q	57.8	66.3	80.4	87.0
Ours-1q	**59.7**	67.0	**82.3**	87.5

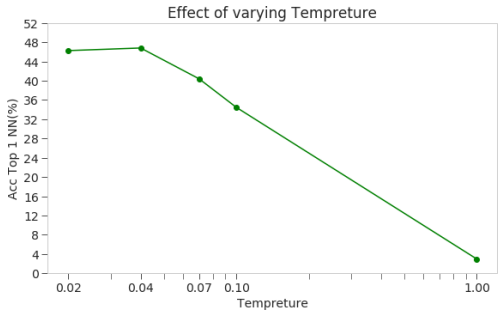
TABLE 2.16. **Transferring to PASCAL-VOC classification and detection tasks:** All models use AlexNet and ours is compressed from ResNet-50x4. Our model is on par with ImageNet supervised model. For classification, we denote the fine-tuned layers in the parenthesis. For detection, all layers are fine-tuned. $*$ denotes bigger AlexNet [502].

Method	Cls. mAP	Det. mAP
Supervised on ImageNet	79.9 (all)	59.1
Random Rescaled [502]	56.6 (all)	45.6
Context* [108]	65.3 (all)	51.1
Jigsaw [323]	67.6 (all)	53.2
Counting [324]	67.7 (all)	51.4
CC+vgg-Jigsaw++ [325]	72.5 (all)	56.5
Rotation [149]	73.0 (all)	54.4
DeepCluster* [56]	73.7 (all)	55.4
RFDecouple* [135]	74.7 (all)	58.0
SeLa+Rot* [502]	77.2 (all)	59.2
MoCo [175]	71.3 (fc8)	55.8
Ours-2q	**79.7** (fc8)	58.1
Ours-1q	76.2 (fc8)	**59.3**

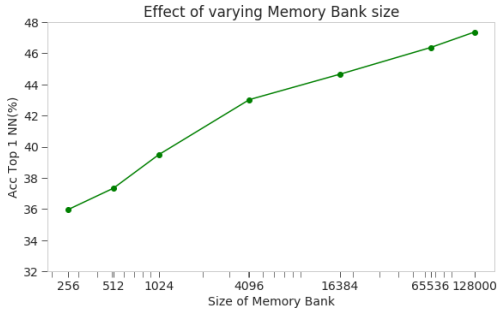
We validate this in Figure 2.14(b) where a larger memory bank results in a more accurate student. When coupled with a small temperature, the large memory bank can help find anchor points that are closer to a query point, thus accurately depicting its close neighborhood.

Effect of momentum parameter: We evaluate various momentum parameters [175] in range (0.999, 0.7, 0.5, 0) and got NN accuracy of (47.35%, 47.45%, 47.40%, 47.34%) respectively. It is interesting that unlike [175], we do not see any reduction in accuracy by removing the momentum. The cause deserves further investigation. Note that momentum is only applicable in case of "ours-2q" method.

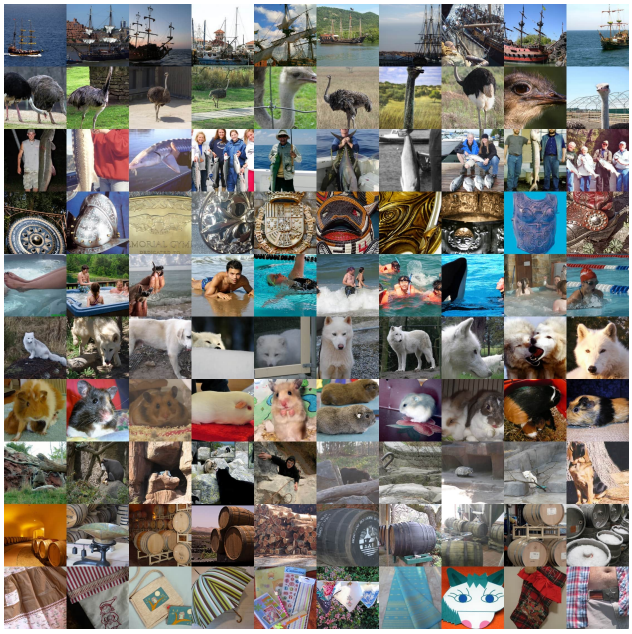
Effect of caching the teacher features: We study the effect of caching the feature of the whole training data in compressing ResNet-50 to ResNet-18 using all ImageNet training data. We realize that caching reduces the accuracy by only a small margin 53.4% to 53.0% on NN and 61.7% to 61.2% on linear evaluation while reducing the running time by a factor of almost 3. Hence, for all experiments using ResNet-50x4, we cache the teacher as we cannot afford not doing so.



(a)



(b)



(c)

FIGURE 2.14. **Ablation and qualitative results:** We show the effect of varying the temperature in **(a)** and memory bank size in **(b)** using ResNet-18 distilled from cached features of MoCo ResNet-50. In **(c)**, we show randomly selected images from randomly selected clusters for our best AlexNet model. Each row is a cluster. This is done *without cherry-picking* or manual inspection. Note that most rows are aligned with semantic categories. We have more of these examples in the Appendix.

2.3.6 Conclusion

We introduce a simple compression method to train SSL models using deeper SSL teacher models. Our model outperforms the supervised counterpart in the same task of ImageNet classification. This is interesting as the supervised model has access to strictly more information (labels). Obviously, we do not conclude that our SSL method works better than supervised models "in general". We simply compare with the supervised AlexNet that is trained with cross-entropy loss, which is standard in the SSL literature. One can use a more advanced supervised training e.g., compressing supervised ResNet-50x4 to AlexNet, to get much better performance for the supervised model.

CHAPTER 3

Model Parameters Efficiency

3.1 NOLA: Compressing LoRA using Linear Combination of Random Basis

Fine-tuning Large Language Models (LLMs) and storing them for each downstream task or domain is impractical because of the massive model size (e.g., 350GB in GPT-3). Current literature, such as LoRA, showcases the potential of low-rank modifications to the original weights of an LLM, enabling efficient adaptation and storage for task-specific models. These methods can reduce the number of parameters needed to fine-tune an LLM by several orders of magnitude. Yet, these methods face two primary limitations: (1) the parameter count is lower-bounded by the rank one decomposition, and (2) the extent of reduction is heavily influenced by both the model architecture and the chosen rank. We introduce NOLA, which overcomes the rank one lower bound present in LoRA. It achieves this by re-parameterizing the low-rank matrices in LoRA using linear combinations of randomly generated matrices (basis) and optimizing the linear mixture coefficients only. This approach allows us to decouple the number of trainable parameters from both the choice of rank and the network architecture. We present adaptation results using GPT-2, LLaMA-2, and ViT in natural language and computer vision tasks. NOLA performs as well as LoRA models with much fewer number of parameters compared to LoRA with rank one, the best compression LoRA can archive. Particularly, on LLaMA-2 70B, our method is almost 20 times more compact than the most compressed LoRA without degradation in accuracy. Our code is available here: https://github.com/UCDvision/NOLA

3.1.1 Introduction

Large pre-trained neural networks have exhibited remarkable generalization abilities across a diverse range of downstream tasks in both natural language processing and computer vision, achieving unprecedented data efficiency. For instance, large language models have demonstrated the capability for few-shot generalization [46] across a variety of tasks, including translation, question-answering, cloze tasks, and reasoning. Similarly, in DINOv2, [327] showcase how a large pre-trained ViT model [110] with more than 1B parameters yields superior all-purpose visual features for a variety of downstream benchmark tasks at both image and pixel levels. Typically, these pre-trained large models are adapted to downstream tasks through

fine-tuning of their parameters. However, fine-tuning and storing the entire set of model parameters for each task incurs a significant storage cost (e.g., 350GB for GPT-3). This challenge has spurred a considerable body of recent works focusing on parameter-efficient fine-tuning of large models [67, 102, 197, 412, 492].

Inspired by the low intrinsic dimensionality of over-parameterized networks' optimal parameters [19, 261], [197] proposed a seminal hypothesis that the change in weights during model adaptation/finetuning has a low "intrinsic rank", leading to the development of Low-Rank Adaptation (LoRA). In essence, LoRA enables the indirect training of a linear layer in a neural network by optimizing the rank-decomposition matrices for the weight change in these layers, resulting in a significant reduction in the number of parameters required for adaptation (e.g., 10,000× parameter reduction for GPT-3). Notably, LoRA has gained popularity, and various extensions of this method have been proposed since its inception [102, 492]. However, LoRA and its derivatives have three inherent limitations: (1) the parameter count is lower-bounded by the rank one decomposition of linear layers, and (2) the number of parameters is quantized since rank is an integer number, and (3) the number of parameters inherently depends on the model's architecture, i.e., the dimensions of the linear matrix, and the choice of rank. In this paper, we introduce a method, denoted as NOLA, that offers the same benefits as LoRA while addressing its limitations. NOLA allows one to decouple the number of trainable parameters from both the choice of rank and the network architecture, and it breaks the rank-one decomposition limit of LoRA.

NOLA is inspired by the recent work by [322], titled PRANC. In this work, we reparameterize a neural network using a linear combination of pseudo-randomly generated weights. Then, we indirectly train the network parameters by optimizing the linear mixture coefficients. This approach results in a significant reduction in the total number of parameters needed to represent the network. Unlike PRANC, our focus in NOLA is on reparameterizing the change of neural weights for fine-tuning large models. More critically, unlike PRANC, NOLA incorporates the invaluable insight from [197], which posits that the weight change during fine-tuning is intrinsically low-rank. In essence, we utilize the rank-decomposition presented in LoRA but assemble the rank-decomposition matrices as a linear combination of pseudo-random matrices (i.e., the 'basis'). Optimizing the rank-decomposition matrices in NOLA is akin to determining the linear mixture coefficients for the random matrices. This design allows us to decouple the number of parameters from the shape of the linear layer and also from the rank choice. Furthermore, the low-rank constraints offer substantial advantages in compute and memory footprint over the methodology proposed in PRANC. Figure 3.1 illustrates the fundamental concept of NOLA.

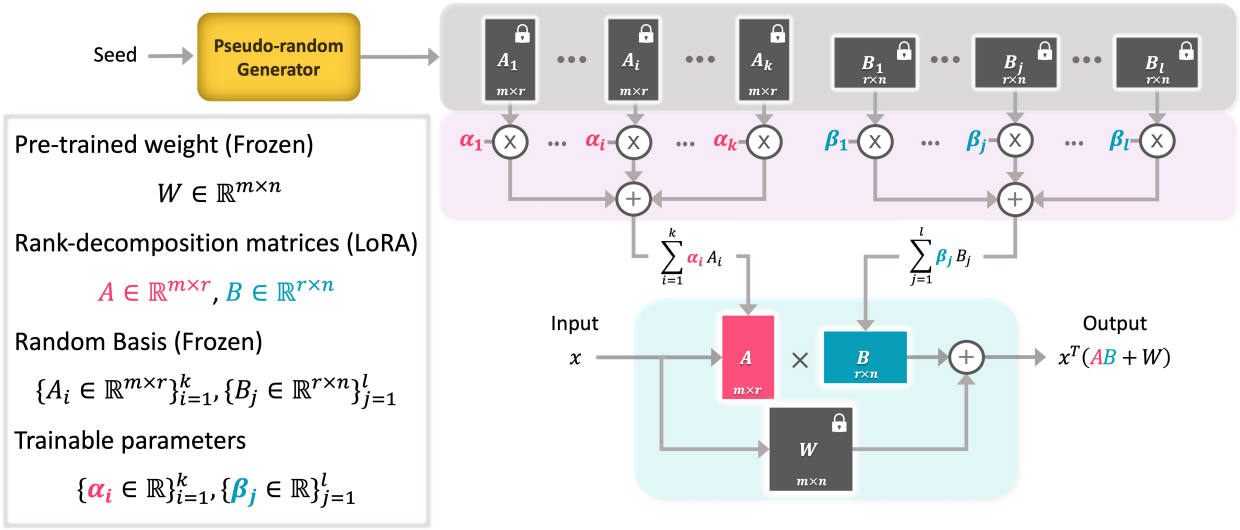


FIGURE 3.1. **Our Method (NOLA):** After constraining the rank of ΔW by decomposing it to $A \times B$, we reparametrize A and B to be a linear combination of several random basis matrices. We freeze the basis and W and learn the combination coefficients. To reconstruct the model, we store the coefficients and the seed of the random generator which is a single scalar. NOLA results in more compression compared to LoRA and more importantly decouples the compression ratio from the rank and dimensions of W. One can reduce the number of parameters to 4 times smaller than rank=1 of LoRA which is not possible with LoRA due to rank being an integer number.

Why Fewer Parameters Matter?

We envision a future where we must efficiently manage and transition between multiple Large Language Models (LLMs), each tailored for specific tasks. This vision arises from the necessity for LLMs customized with private data and/or the concept of crafting a universal LLM that can summon customized LLMs as a versatile toolbox to tackle diverse tasks [388]. However, currently, customized LLMs demand substantial storage, and the process of switching between them lacks efficiency due to large I/O operations. NOLA offers a more compact reparameterization solution that can be stored effectively in GPU memory, allowing for on-demand reconstruction directly on the GPU itself when a new task arises.

Note that while storing parameters in CPU memory is a cost-effective option, the process of transferring them from CPU to GPU incurs substantial time and power consumption. Moreover, this data transfer relies on a shared resource (e.g., PCIe bus), which may experience congestion in busy server environments. Therefore, optimizing model compactness to fit several of them within the limited GPU memory proves advantageous in these scenarios. As an example, 1,000 customized GPT-3 models using LoRA need almost 35GB of memory (assuming LoRA compresses it by a factor of $10,000\times$), which may not fit in the GPU memory along with the LLM model itself. Hence, compacting it by an additional factor of 5 reduces it to 7GB, which can fit in the GPU memory, leading to very efficient switching between the tasks.

Contributions. Our specific contributions in this paper are: 1) A novel reparameterization for compressing task-specific large language models, denoted as NOLA. 2) NOLA decouples the compression ratio from the rank and dimension of the weight matrix, unlocking higher compression ratios while keeping most benefits of LoRA, including reduced memory and computation at training time. 3) NOLA can be further improved by quantizing the coefficients and can be applied to other architectures like CNNs. 4) Applied to PRANC, NOLA speeds it up and reduces its memory footprint.

3.1.1.1 Proposed Method: NOLA

LoRA, short for Low-Rank Adaptation, is a widely embraced method for customizing a pre-trained model, such as GPT, for a specific task. Instead of changing all parameters denoted as W within a given layer, LoRA maintains the original pre-trained parameters W as a constant and learns a residual adjustment ΔW to fine-tune the model for the new task. The resulting updated layer parameters are then computed as $W + \Delta W$. The core concept behind LoRA is to minimize the size of ΔW by constraining its rank. In a more formal context, considering $W \in \mathbb{R}^{m\times n}$ and $\Delta W \in \mathbb{R}^{m\times n}$, LoRA accomplishes this by reparameterizing ΔW as the product of two matrices, $\Delta W = A \times B$, where $A \in \mathbb{R}^{m\times r}$ and $B \in \mathbb{R}^{r\times n}$, with r representing the rank—a hyperparameter. By selecting a relatively small rank ($r << \min(m, n)$), LoRA efficiently reduces memory usage. This optimization is achieved by storing A and B, which exhibit a significantly more compact representation than the full ΔW. The resulting compression ratio is quantified as $\frac{mn}{r(m+n)}$. Unfortunately, this compression ratio is: (1) tied to the shape of the parameters m and n, and hence the model architecture and (2) is upper-bounded by $\frac{mn}{m+n}$, i.e., for $r = 1$.

In this paper, we introduce a novel reparameterization technique for ΔW that effectively decouples the rank from the compression ratio, allowing for a compression ratio higher than $\frac{mn}{m+n}$, which corresponds to $r = 1$ in the LoRA framework. To achieve this, we draw inspiration from PRANC [322] and reparameterize matrices A and B to exist within a lower-dimensional space defined by a set of randomly generated basis matrices. Formally, we express this as:

$$A = \sum_{i=1}^{k} \alpha_i A_i \quad , \quad B = \sum_{j=1}^{l} \beta_j B_j \tag{3.1}$$

where, $A_i \in \mathbb{R}^{m\times r}$ and $B_j \in \mathbb{R}^{r\times n}$ are random matrices generated by a Pseudo Random Number Generator with a fixed seed. We subsequently learn A and B as linear combinations of these predefined and frozen random matrices. Importantly, the random matrices themselves remain constant, and we optimize only the coefficient vectors α and β. Then:

$$\Delta W = \Big(\sum_{i=1}^{k} \alpha_i A_i\Big) \times \Big(\sum_{j=1}^{l} \beta_j B_j\Big) \tag{3.2}$$

In practical terms, to store ΔW for a specific task, we only need to retain the seed (a single scalar) and the coefficient vectors α and β. Remarkably, this approach allows for a small number of basis matrices $(k + l)$ to be chosen, irrespective of the rank of the $A \times B$ factorization and the shape of ΔW, thereby enhancing the compression ratio to go beyond $\frac{mn}{m+n}$.

Quantization of coefficients α and β: We are mainly interested in reducing the storage for a new task, assuming the pre-trained LLM is already available. Hence, to further reduce the storage, we quantize the α and β coefficients to lower precision (e.g., 4 bits) while the random basis and the pre-trained LLM weights have standard FP16 floating point precision. Note that one can also quantize A and B matrices in LoRA; however, our method does not force A and B themselves to be of low precision. One can quantize α and β after the optimization (post-training quantization) or while optimizing them (quantization-aware training). We expect the latter to perform better. For quantization-aware learning, we use the method in [210, 363] where we use the quantized α and β in the forward pass and update the FP16 versions of α and β in the backward pass. Moreover, we use the Straight-Through Estimator (STE) trick [37] to estimate the gradient.

A few recent works have shown that it is possible to quantize the weights of LLMs for each task, which reduces both computation and storage. However, these methods are not suitable for a large number of tasks since the quantized LLM is still task-specific and large.

Memory Efficiency: Note that depending on the number of basis matrices, the random basis may be large, requiring a large memory. Interestingly, generating random matrices in the GPU itself is very fast, so similar to PRANC, at each iteration, we generate chunks of the basis matrices at a time, multiply them by the corresponding coeficents, and discard them. Generating a basis on the fly at the inference time can drastically reduce the communication cost between CPU and GPU since α and β vectors for several tasks can be stored in the GPU memory.

Efficiency of NOLA compared to PRANC: PRANC [322] reshapes the whole model parameters or each layer of it into a long vector and reparameterizes that by a linear combination of random vectors. However, as mentioned in [322], this method involves multiplication of the coefficients with the big random matrix twice at each iteration (once in forward and once in backward passes), which is very expensive. For instance, the size of the random matrix for ResNet18 with 1000 coefficients will be almost $11M \times 1K$. NOLA reduces this computation while keeping the same number of parameters by reshaping the long vector to be a 2D matrix and constraining its rank. Assuming d^2 weights and k random basis, the basis matrix

size for PRANC will be kd^2 while NOLA with rank r reduces that to kdr assuming that each component in $A \times B$ has $\frac{k}{2}$ basis matrices to keep the number of parameters equal to PRANC. Then, the total compute for PRANC will be $kd^2 + d^2 \approx kd^2$ while for NOLA, it will be $kdr + 2dr \approx kdr$. Hence, assuming a small rank r, NOLA can reduce the training time of PRANC by a large factor $\frac{d}{r}$ due to the reduction of the computation at forward and backward passes. Moreover, in the appendix, we empirically show that NOLA offers a better coverage of the weight space compared to PRANC.

Structure of the parameters: Note that one can apply NOLA to model architectures other than transformer by simply reshaping the weight tensor to be a 2D matrix (preferably close to square) and then compressing it. We do this in our ResNet experiments in the Appendix, where the weight matrices are 4D tensors of convolutional filters.

3.1.2 Experiments

Here, we evaluate NOLA in transfer learning tasks in both NLP and vision. Moreover, in the appendix, we evaluate NOLA in training from scratch.

3.1.2.1 NOLA on GPT-2:

We adapt the parameters of pre-trained GPT-2 to three different Natural Language Generation (NLG) datasets by finetuning the parameters using NOLA. We use GPT-2-Large and GPT-2-Medium in our experiments. We follow the [197, 270] for our adaptation setup.

Datasets: We utilize the following datasets for our Natural Language Generation (NLG) task: E2E NLG Challenge [326] serves as a commonly used benchmark for evaluating NLG models. It encompasses of 51,200 samples, distributed as follows: 42,200 for training, 4,600 for validation, and an additional 4,600 for testing. DART [316] is yet another significant dataset employed for evaluating text-to-data generation. This dataset is rich with 82,191 examples drawn from various domains. WebNLG [147] is a text-to-data dataset, boasting 22,000 examples spanning 14 distinct categories. Notably, the WebNLG test set introduces five new categories, prompting us to present results across all categories within this dataset. These datasets collectively provide a comprehensive foundation for our NLG evaluation and experimentation.

LoRA: In our experiments, we apply LoRA on both query and value projection layer in each attention block. Since number of parameters is tied to the rank, we adjust the rank to reduce number of parameters. We compare to LoRA with both rank four and rank one.

Other Baselines: Moreover, we compared NOLA to a few other baselines, including finetuning all parameters, Adapters [193, 281, 343, 380], and Prefix-layer tuning (PreLayer) [270].

TABLE 3.1. **E2E NLG Challenge:** We compare NOLA to LoRA with two different architectures: GPT-2 medium (M) and large (L). QV refers to Query and Value projection, while MLP denotes the MLP layer within transformers. Adapter^{L} and Adapter^{H} are two Adapter baselines reported in [197]. The best and second best performing methods are in bold. To reduce number of parameters in LoRA, we use lower rank (LoRA rank=1). We don't see drop in performance by reducing number of trainable parameters to $\frac{1}{20}$ of LoRA with rank 4 in GPT-L. Note that in LoRA, one cannot reduce the number of parameters below rank one.

Method	Adapted Layers	Adapter Rank	# Trainable Parameters	E2E NLG Challenge BLEU	NIST	MET	ROUGE-L	CIDEr
GPT-2 M								
Finetune	All Layers	-	354.920M	68.2	8.62	46.2	71.0	2.47
Adapter^{L}	Extra Layers	-	0.370M	66.3	8.41	45.0	69.8	2.40
Adapter^{L}	Extra Layers	-	11.090M	68.9	8.71	46.1	71.3	2.47
Adapter^{H}	Extra Layers	-	11.090M	67.3	8.50	46.0	70.7	2.44
Finetune^{Top2}	Last 2 Layers	-	25.190M	68.1	8.59	46.0	70.8	2.41
PreLayer	Extra Tokens	-	0.350M	69.7	8.81	46.1	71.4	2.49
LoRA	QV	4	0.350M	**70.4**	**8.85**	**46.8**	**71.8**	**2.53**
LoRA	QV	1	0.098M	68.7	8.72	45.6	70.5	2.43
NOLA (Ours)	QV	8	0.350M	70.1	8.80	**46.8**	**71.7**	**2.53**
NOLA (Ours)	QV	8	0.096M	70.0	**8.82**	46.7	71.6	2.51
NOLA (Ours)	MLP	8	0.096M	**70.2**	8.79	46.7	**71.8**	2.51
NOLA (Ours)	QV	8	0.048M	70.1	**8.82**	46.4	71.4	2.52
NOLA (Ours)	MLP	8	0.048M	69.4	8.71	46.5	71.5	2.47
GPT-2 L								
Finetune	All Layers	-	774.030M	68.5	8.78	46.0	69.9	2.45
Adapter^{L}	Extra Layers	-	0.880M	69.1	8.68	46.3	71.4	2.49
Adapter^{L}	Extra Layers	-	23.000M	68.9	8.70	46.1	71.3	2.45
PreLayer	Extra Tokens	-	0.770M	70.3	**8.85**	46.2	**71.7**	2.47
LoRA	QV	4	0.770M	**70.4**	**8.89**	**46.8**	**72.0**	2.47
LoRA	QV	1	0.184M	69.9	8.81	46.7	71.6	**2.53**
NOLA (Ours)	QV	8	0.144M	**70.5**	**8.85**	**46.8**	**71.7**	**2.54**
NOLA (Ours)	MLP	8	0.144M	70.1	8.80	46.5	71.2	2.52
NOLA (Ours)	QV	8	0.072M	69.8	8.80	46.4	71.3	2.51
NOLA (Ours)	MLP	8	0.072M	69.4	8.71	46.6	71.5	2.48
NOLA (Ours)	QV	8	0.036M	70.1	8.80	46.7	71.7	**2.53**
NOLA (Ours)	MLP	8	0.036M	70.0	8.81	46.4	71.5	**2.53**

NOLA: We evaluate NOLA with two different variations: 1. Adapting MLP layers. 2. Adapting query and value projection matrices. Note that, unlike LoRA, we can use any number of parameters while applying NOLA to any weight structure since the number of parameters is not tied to the shape of the weight tensor. We allocate an equal number of parameters to A and B in each NOLA layer (i.e., $k = l$). Using $k = l = 1000$ results in $0.096M$ parameters in GPT-M and $0.144M$ parameters in GPT-L. Also, we use half ($k = l = 500$) and quarter ($k = l = 250$) number of parameters per layer to get smaller checkpoints.

Implementation Details: We trained our models using a single NVIDIA RTX 6000 Ada Generation GPU. For all hyperparameters except learning rate, we use the same values as LoRA for training and evaluation of GPT-2. We train our models for 5 epochs with a learning rate of 0.1 and no weight decay. We use a

TABLE 3.2. **Training time and memory:** We compare the training memory and running time of NOLA to LoRA. Since generating a random basis on each layer has a small overhead, we can generate random basis once and share the basis across layers to save time. This version of NOLA has a similar runtime to LoRA and on-par accuracy to NOLA with a non-shared basis.

Model & Method	Random Basis	Training Memory	Training Time (ms/batch)	# Trainable Parameters	E2E NLG Challenge				
					BLEU	NIST	MET	ROUGE-L	CIDEr
GPT-2 L (LoRA r=1)	-	33.35GB	776	184K	69.89	8.81	46.70	71.64	2.53
GPT-2 L (NOLA QV)	Non-shared	33.37GB	834	144K	70.46	8.85	46.77	71.68	2.54
GPT-2 L (NOLA QV)	Shared	33.40GB	779	144K	70.32	8.85	46.74	71.71	2.54

batch size of 8. We use a rank of 8 for NOLA in our experiments. Like LoRA, we scale $A \times B$ with $\frac{c}{r}$, where c is a hyperparameter and r is the rank. We use the default value of $c = 1$.

Results: We compare to LoRA and other baselines in Table 3.1 and Table A.10 in the Appendix. NOLA is on par or better compared to other methods with the same number of parameters. In the E2E task, NOLA with $0.036M$ parameters archives a BLEU score of 70.12, which is 20 times more compact compared to LoRA with rank 4 that has $0.77M$ parameters and archives a BLEU score of 70.4. This NOLA model uses a rank of 8, which does not affect the number of parameters and increases the run time slightly (negligible compared to that of the actual LLM model). Note that our goal is to reduce the number of parameters without reducing the accuracy, which is shown in Tables 3.1 and A.10. We are not claiming that NOLA improves the accuracy compared to baselines. We show various variants of NOLA (MLP, QV, etc) to emphasize that NOLA is not very sensitive to the choice of the variation.

Training Time and Memory of NOLA: Similar to LoRA, in the inference time, we can calculate $A \times B$ offline and merge it with W. Therefore, NOLA does not have any overhead compared to the original model. In training time, NOLA has a small overhead due to the multiplication of coefficients to basis weights. We measure the running time and memory footprint of NOLA during training and compare it to LoRA in Table 3.2. Since generating a random basis for each layer adds a small overhead, we can share the random basis across all layers and generate and store them only once to improve the running time. We measure time and memory with a batch size of 8. NOLA, with a unique random basis for each layer, is slightly slower than LoRA. However, NOLA with a shared random basis has on-par accuracy with the unique random basis and has a similar running time to LoRA.

Ablation Study on the rank of NOLA: Since the number of parameters is decoupled from the rank of the matrix, we can solely evaluate the effect of rank without changing the number of parameters. We report the effect of rank in Table 3.3. We vary the rank from 1 to 8 and use $c = 1.0$ for ranks 4 and 8, and $c = 0.5$ for lower ranks. As also noted by [197], understanding the effect of rank needs more rigorous study as future work.

TABLE 3.3. **Effect of rank in NOLA:** We vary the rank from 1 to 8. Note that we can use the same number of parameters in all ranks since the number of parameters is not tied to the rank in NOLA.

# Train Params	Rank	E2E NLG Challenge BLEU	NIST	MET	ROUGE-L	CIDEr
GPT-2 M (NOLA QV)						
96K	8	70.03	8.82	46.74	71.64	2.51
96K	4	69.69	8.76	46.56	71.44	2.51
96K	2	70.47	8.86	46.71	71.79	2.53
96K	1	69.09	8.78	45.87	70.15	2.45
96K	8	70.03	8.82	46.74	71.64	2.51
48K	8	70.09	8.82	46.44	71.36	2.52
24K	8	68.30	8.67	45.13	68.91	2.40
12K	8	67.18	8.60	43.99	68.38	2.26
GPT-2 L (NOLA QV)						
144K	8	70.46	8.85	46.77	71.68	2.54
144K	4	70.25	8.85	46.84	71.81	2.54
144K	2	69.69	8.78	46.55	71.25	2.51
144K	1	69.71	8.82	46.47	70.96	2.51

TABLE 3.4. **Quantization of coefficients:** Post-training quantization of parameters does not degrade the performance up to the 4 bit quantization. In quantization-aware training, NOLA is more robust to quantization compared to LoRA.

Model & Method	# Quant Bits	E2E NLG Challenge BLEU	NIST	MET	ROUGE-L	CIDEr
Post Training Quantization						
GPT-2 L (LoRA r=1)	16-bit	69.89	8.81	46.70	71.64	2.53
	8-bit	69.91	8.81	46.69	71.75	2.53
	4-bit	69.63	8.75	46.32	71.24	2.48
	3-bit	62.01	8.02	42.01	67.23	2.07
GPT-2 L (NOLA QV)	16-bit	70.46	8.85	46.77	71.68	2.54
	8-bit	70.43	8.84	46.78	71.72	2.54
	4-bit	70.29	8.82	46.74	71.82	2.52
	3-bit	65.14	8.58	44.38	67.56	2.23
Quantization Aware Training						
GPT-2 L (LoRA r=1)	3-bit	67.08	8.86	44.67	68.76	2.36
	2-bit	56.13	4.70	35.38	63.68	1.40
GPT-2 L (NOLA QV)	3-bit	70.14	8.82	46.58	71.61	2.53
	2-bit	68.69	8.72	46.06	70.61	2.48

3.1.2.2 NOLA with Quantized Coefficients, α and β:

We evaluate the performance of NOLA with rank 4 with quantized α and β parameters on the E2E dataset in Table 3.4 in two different setups. First, we do Post Training Quantization (PTQ) by quantizing the parameters to q bits after the training. We observe no significant drop in both LoRA and NOLA in 4 bits PTQ experiments. Second, we evaluate models with Quantization Aware Training (QAT) where we quantize the coefficients in the forward pass and update them in the non-quantized form. In QAT with 3 bits, NOLA has a slight drop of 0.3 points while LoRA has a drop of 2.8 points in the BLEU metric. Note that in NOLA, although α and β are quantized, A and B in Eq 3.1 are not quantized since the basis matrices, A_i and B_j, are non-quantized. This is in contrast to LoRA where A and B are quantized.

3.1.2.3 NOLA on LLaMA-2:

Finetuning with LoRA for larger LLMs is challenging due to compute demand, so we need to resort to QLoRA where the base model is quantized. Our NOLA framework is agnostic to the quantization of base model, so for LLaMA-2 [436] experiments, we use NOLA with base model quantized to 8-bits while the NOLA coefficients still use 16-bit. We fine-tune the pretrained LLaMA-2 model using 8-bit QLoRA on the Alpaca dataset [417], reporting both training and validation loss metrics specific to the Alpaca dataset. Additionally, we employ the MMLU (Massively Multitask Language Understanding) benchmark [184] to assess performance across a diverse set of language understanding tasks. This benchmark comprises 57

TABLE 3.5. **Instruction finetuning for quantized LLaMA-2:** NOLA fine-tunes LLaMA-2 70B (8-bit) with $0.57M$ parameters, a 95% reduction compared to rank-one LoRA. We use quantized version of LLaMA-2 for both LoRA and NOLA, so our LoRA baseline is equivalent to QLoRA.

	LLaMA-2 - 7B (8-bit)			LLaMA-2 - 13B (8-bit)			LLaMA-2 - 70B (8-bit)		
	w/o Finetuning	LoRA	NOLA	w/o Finetuning	LoRA	NOLA	w/o Finetuning	LoRA	NOLA
Adapter Rank	-	1	16	-	1	16	-	1	16
Trainable Parameters	-	2.50M	0.06M (↓97%)	-	3.91M	0.14M (↓96%)	-	12.94M	0.57M (↓95%)
Train Loss	1.53	0.97	1.05	1.43	0.94	0.95	1.42	0.87	0.90
Val Loss	1.74	1.04	1.01	1.59	0.96	0.97	1.53	0.92	0.90
MMLU Acc	45.3	46.5	46.5	54.8	55.3	55.3	68.9	69.5	69.4

tasks spanning areas such as mathematics, history, computer science, and law. On MMLU, we focus on 5-shot evaluation (providing 5 examples in the context), following the standard practice.

Implementation Details: We apply LoRA and NOLA across all layers (Query, Key, Value and Output projections and MLP layers) of the transformer on three different model sizes of LLaMA-2: 7B, 13B, and 70B. For our LoRA experiments, we set the rank $r = 1$ since LoRA paper (its Table 15) shows that rank one model performs as well as higher ranks for larger LLMs (e.g., GPT-3). For NOLA, we use $r = 16$ and adjust the number of optimized parameters for each LLaMA-2 model size using the following settings: $k = l = 128$ for LLaMA-2 7B, $k = l = 256$ for LLaMA-2 13B, and $k = l = 512$ for LLaMA-2 70B. During fine-tuning LoRA or NOLA, we adhere to the hyperparameters reported in QLoRA, [102]. We optimize for one epoch on the Alpaca dataset with a batch size of 256 using four RTX 3090 GPUs. The learning rate is 0.0002 for LoRA and 0.001 for NOLA. Similar to LoRA, we scale $A \times B$ with $\frac{c}{r}$, where c is a hyperparameter and r is the rank. We use $c = 16$ for LoRA and $c = 4$ for NOLA.

Results: Results are presented in Table 3.5. Remarkably, NOLA is capable of fine-tuning LLaMA-2 70B with fewer than $0.6M$ parameters, representing an average reduction of parameters by 95% compared to LoRA with rank one.

3.1.2.4 NOLA on Vision Transformers

We perform experiments on the image classification task on ViT-B and ViT-L architectures with both supervised and self-supervised (MAE) initializations.

Implementation details: All pre-trained networks are obtained from Timm library [472]. All approaches are trained for 50 epochs, and the top-1 accuracy at the final epoch is reported. We use a batch-size of 64 and tune the initial learning rate for each dataset and architecture for all approaches. Since our focus is on finetuning on small datasets, we use 5 and 10 labeled examples per class for finetuning. Since there is a high variance in performance due to the small training sets, we sample four different sets of training samples

per k-shot and three different initializations for LoRA/NOLA and report the mean accuracy and standard deviation. All approaches are trained with cross-entropy loss. Additional details are in the appendix.

Datasets: ImageNet-21k and ImageNet-1k are used to pretrain the backbone models. We use CIFAR10 [248], CIFAR100 [247], CUB-200-2011 [471], Caltech-101 [130], Aircraft [301], Food101 [41], Pets [333] and SUN397 [481] datasets for finetuning.

Baselines: We compare NOLA with three baseline approaches: Linear, Full-FT (full fine-tuning) and LoRA [197]. In Linear, only the final classifier head is optimized, while in Full-FT, the entire backbone network is optimized too. No additional parameters are used during finetuning for either of these approaches. We apply LoRA on Query, Key, and Value projection matrices with rank set 4 for ViT-B to 1 or 4 for ViT-L. In our preliminary experiments, LoRA with rank one sees a big drop in accuracy. This is aligned with our GPT-2 M experiments where smaller models require higher rank. We apply NOLA on the MLP layers and use rank of 4 for ViT-B and 1 for ViT-L. We report the number of trainable parameters for each approach excluding the classifier head parameter count which is common to all approaches. We also report nearest-neighbor (1-NN) evaluation for zero-shot classification on downstream tasks using ImageNet pretrained features.

Results: Results on finetuning on image classification tasks are presented in Table 3.6. A naive Nearest Neighbor approach performs competitively on the CUB and Caltech datasets. LoRA and NOLA are significantly better than Linear and Full-FT on several settings (e.g. CIFAR-100 5 shot on ViT-B-MAE). This might be due to the overfitting of the Linear and Full-FT approaches on the limited number of train samples. When using a similar number of training parameters, NOLA outperforms LoRA in most of the settings across architectures and datasets. It also achieves comparable performance to LoRA with just half or one-third of the training parameters of LoRA. This is consistent with our observations on the NLG tasks. The difference between the two methods is particularly noticeable when the number of training examples is small - either in 5 shot setup or when the number of classes is small, as in CIFAR-10. This suggests that the improvement obtained by NOLA could be due to the reduction in the number of training parameters. Both LoRA and NOLA consistently and significantly outperform Linear and Full-FT approaches. NOLA can easily be employed in MLP layers since the number of training parameters is decoupled from the weight matrix dimensions. A similar application of LoRA would require 8× more training parameters due to the large hidden layer dimensionality of the MLP module. We empirically observe that NOLA-MLP slightly outperforms NOLA on attention block (see Table A.9 in appendix). We provide results on four additional datasets used for benchmarking transfer learning in Table 3.7. Aircraft, Food101 and Pets are finegrained

TABLE 3.6. **Results on vision transformers.** We finetune ImageNet pre-trained ViT models on multiple small datasets with 5 and 10 training samples. The mean accuracy and standard deviation across 12 runs are reported. The number of train parameters for Linear classifier depends on the number of classes in the dataset (0.01, 0.1, 0.2, 0.1M parameters for CIFAR-10, CIFAR-100, CUB and Caltech respectively). We do not count the linear layer in trainable parameters. The best performing method is in bold while all methods within one point of the best are underlined. NOLA outperforms LoRA with comparable parameters across datasets and architectures, particularly in the low training data regime. The performance of NOLA with half or one-third of the training parameters is comparable to that of LoRA. Note that LoRA with rank one is the most compact LoRA.

Base Model		# Train Params	CIFAR-10 5	CIFAR-10 10	CIFAR-100 5	CIFAR-100 10	CUB-200-2011 5	CUB-200-2011 10	Caltech-101 5	Caltech-101 10
ViT-B	Nearest Neighbor		79.6	80.8	52.4	59.2	71.9	78.0	84.1	87.5
	Linear	0	80.8 (1.1)	85.1 (1.0)	58.9 (0.9)	64.5 (0.7)	72.7 (0.4)	79.2 (0.2)	85.8 (0.8)	88.5 (0.4)
	Full-FT	5.3M	73.9 (6.5)	87.6 (2.7)	61.4 (2.4)	78.2 (1.1)	59.7 (1.9)	76.6 (0.2)	87.9 (0.8)	**91.1** (0.5)
	LoRA (r=4)	141K	87.3 (2.3)	**93.1** (0.5)	**76.3** (0.5)	**81.6** (0.9)	**75.7** (0.5)	**82.4** (0.3)	**88.4** (1.1)	90.8 (0.5)
	NOLA-MLP	**47K**	**87.9** (1.3)	92.2 (0.5)	75.1 (0.6)	81.3 (0.8)	75.5 (0.6)	81.7 (0.4)	88.0 (1.2)	90.6 (0.5)
ViT-B-MAE	Nearest Neighbor		18.2	19.8	5.8	9.8	13.2	25.3	28.2	40.7
	Linear	0	27.4 (1.9)	34.5 (1.4)	15.7 (0.7)	22.2 (0.2)	12.7 (0.3)	18.4 (0.3)	66.9 (1.1)	76.9 (0.6)
	Full-FT	5.3M	41.1 (4.4)	58.4 (3.6)	19.7 (4.8)	24.2 (11.1)	23.0 (3.8)	51.9 (2.8)	76.4 (2.3)	86.5 (0.5)
	LoRA (r=4)	141K	54.7 (1.6)	70.1 (2.2)	39.3 (3.1)	52.4 (1.3)	35.7 (1.5)	**54.0** (0.6)	82.4 (0.6)	87.7 (0.5)
	NOLA-MLP	**47K**	**55.1** (2.6)	**72.1** (2.7)	**42.1** (1.4)	**53.5** (1.0)	**35.8** (1.5)	53.9 (0.6)	**88.0** (1.2)	**90.6** (0.5)
ViT-L	Nearest Neighbor		88.7	89.9	68.9	74.0	77.4	82.3	88.4	90.1
	Linear	0	84.1 (1.8)	88.4 (1.1)	63.7 (1.3)	70.6 (0.9)	73.7 (0.6)	79.2 (0.3)	87.6 (0.9)	89.9 (0.4)
	Full-FT	289M	77.2 (2.7)	90.2 (2.8)	74.0 (2.3)	86.2 (0.6)	73.3 (0.9)	83.9 (0.2)	88.7 (1.0)	91.3 (0.7)
	LoRA (r=4)	375K	86.5 (2.0)	93.8 (1.0)	82.9 (0.9)	87.6 (0.6)	**81.2** (0.4)	**85.3** (0.3)	89.3 (0.7)	91.3 (0.3)
	LoRA (r=1)	94K	86.3 (1.3)	92.8 (0.8)	82.2 (0.8)	85.6 (0.9)	80.6 (0.3)	85.2 (0.3)	89.9 (1.0)	91.6 (0.4)
	NOLA-MLP	94K	**89.0** (3.6)	**96.0** (0.5)	**83.6** (0.9)	**87.8** (0.6)	80.8 (0.6)	85.2 (0.2)	**90.0** (0.7)	**91.7** (0.3)
	NOLA-MLP	**47K**	83.9 (1.8)	93.0 (1.7)	81.2 (1.0)	87.1 (0.6)	80.7 (0.5)	85.0 (0.3)	89.8 (0.8)	91.5 (0.4)
ViT-L-MAE	Nearest Neighbor		33.5	39.2	15.2	21.9	16.9	29.2	57.4	67.6
	Linear	0	40.2 (2.3)	49.2 (2.6)	22.6 (0.9)	31.3 (0.5)	15.2 (0.3)	21.9 (0.4)	75.2 (0.5)	83.2 (0.6)
	Full-FT	289M	60.6 (4.5)	68.3 (4.0)	37.9 (11.1)	52.0 (16.1)	42.2 (2.3)	67.1 (1.1)	87.2 (0.8)	90.8 (0.7)
	LoRA (r=4)	375K	63.5 (3.0)	82.4 (2.3)	50.2 (6.8)	62.6 (5.2)	35.2 (2.9)	60.8 (1.2)	87.0 (0.9)	90.7 (0.4)
	LoRA (r=1)	94K	67.7 (3.8)	83.8 (1.2)	50.4 (1.0)	62.5 (0.6)	32.9 (1.8)	56.6 (1.7)	87.0 (0.6)	90.8 (0.4)
	NOLA-MLP	94K	**70.6** (3.8)	**86.0** (1.4)	**51.7** (1.1)	**63.8** (0.8)	**36.9** (5.6)	**61.6** (1.0)	**87.4** (0.4)	**90.9** (0.5)
	NOLA-MLP	**47K**	69.6 (3.8)	84.8 (1.1)	49.9 (0.8)	62.8 (0.7)	36.1 (0.8)	58.8 (1.2)	87.1 (0.6)	**90.9** (0.4)

TABLE 3.7. **More results on vision tasks.** Using ViT-B, NOLA achieves comparable performance as LoRA (r=4) with just one-third the number of parameters on four challenging datasets. The linear layer sizes are: 0.03M, 0.2M, 0.1M, 0.3M for Aircraft, Food101, Pets and SUN397 respectively.

Method	# Train Params	Aircraft 5 Shot	Aircraft 10 Shot	Food101 5 Shot	Food101 10 Shot	Pets 5 Shot	Pets 10 Shot	SUN397 5 Shot	SUN397 10 Shot
Nearest Neighbor		24.6	27.1	48.4	54.2	82.3	86.2	44.4	51.5
Linear	0	29.7 (2.3)	36.9 (2.1)	53.2 (0.3)	61.5 (0.1)	**88.4** (0.4)	91.3 (0.3)	38.0 (0.8)	42.7 (0.4)
Full-FT	82M	31.4 (2.4)	43.2 (2.0)	48.6 (5.1)	65.8 (2.7)	82.7 (1.1)	91.1 (0.5)	45.0 (3.3)	52.6 (0.3)
LoRA (r=4)	0.141M	32.4 (1.4)	**43.8** (1.5)	60.8 (1.6)	**73.1** (0.6)	85.5 (0.8)	91.6 (0.5)	**51.6** (0.4)	**55.6** (0.3)
NOLA-MLP	0.047M	**33.7** (2.2)	43.3 (1.4)	**64.5** (0.8)	72.6 (0.4)	88.0 (0.6)	**92.2** (0.3)	50.5 (0.4)	55.5 (0.3)

datasets while SUN397 is a large dataset with 397 classes. There is a bigger difference in the performance of Nearest Neighbor and Linear approaches on most of these datasets compared to those in Table 3.6, suggesting that it is harder to adapt to these datasets. In line with our prior observations in Table 3.6, NOLA with just one-third the number of parameters performs comparably to LoRA.

3.1.3 Related Works

Vision and Language Transformer Models: Transformer networks, introduced by [446], emerged as a sequence-to-sequence model in Natural Language Processing (NLP). Their success soon extended to the computer vision community, with subsequent works [113, 433] introducing the Vision Transformer (ViT) network as a strong alternative to the Convolutional Neural Network (CNN) backbones. Transformers accept a sequence of tokens as input. These tokens can be, for instance, word embeddings in language or image patches in vision. BERT [103] and GPT-2 [354] in NLP, and MAE [174] and DINO [59] in computer vision train transformer networks via self-supervision on large amounts of unlabeled data. These studies demonstrate that large transformer networks when trained on massive corpora, generalize well to downstream tasks even when finetuning on very few task-specific examples. For example, [46] show that GPT-3 with 175B parameters is a good few-shot learner. Lastly, the scaling law presented by [224] indicates that a simultaneous increase in training data and model parameters can lead to significant performance gains and emergent capabilities previously unavailable to smaller models.

Parameter Efficient Fine-Tuning: Owing to their unprecedented few-shot generalization performance, large neural networks, such as foundation models and LLMs have gained immense popularity in recent years. An increasing number of users are customizing these models to adapt them to their specific tasks. However, given the colossal size of these models, fine-tuning and storing the entire set of model parameters [103, 354] for each task is impractical. This challenge is exacerbated as the number of tasks increases. In addition to storage concerns, the overhead involved in loading task-specific models and transferring weights from CPU to GPU often becomes a computational bottleneck in many applications. Parameter Efficient Fine-Tuning (PEFT) approaches aim to address these issues by identifying the minimum number of parameters needed to adapt a large model. Adapters [193, 281, 299, 366] are PEFT approaches that achieve adaptation by adding small modules to the intermediate layers of the model. A major drawback of Adapters is the extra latency they introduce in inference. BitFit [517] only adapt bias of the network. Ladder tuning [411] reduce memory footprint in training by avoiding back-propagation through the main backbone. IA3 [284] trains extra parameters in the attention module. Another widely adopted PEFT approach is prompt-tuning for LLMs that involves optimizing a new set of input tokens, or prompts, for each task [167, 257, 270, 286]. While reminiscent of prompt engineering, the distinction lies in training a specific set of prompt tokens in prompt-tuning which might also increase inference latency.

[197] introduced LoRA, demonstrating that a low-rank modification of the original weights is sufficient to adapt an LLM to a new task. Unlike adapters and prompt-tuning, these low-rank modifications can be

integrated into the original weights, thus avoiding additional overhead during inference. However, LoRA has two main limitations: 1) the rank-one decomposition sets a lower bound on the parameters needed for fine-tuning, and 2) the number of required parameters is contingent upon the architecture and the rank choice. Our work, termed NOLA, addresses these challenges by decoupling the trainable parameters from both the rank choice and the network architecture. Several recent studies have sought to enhance LoRA by quantizing its parameters [102, 153, 253, 492], optimizing the design choice of LoRA through neural architecture search [531], or dynamically allocating parameters based on the required rank for each weight matrix [525]. Most of these enhancements are also compatible with our proposed method. In fact, we demonstrate that NOLA can be quantized to 4-bit without any performance degradation, thereby emphasizing that the concept of quantization is distinct from and complementary to, NOLA.

Compact Deep Learning Models: A closely related area to PEFT is model compression. Pruning [172, 233, 272, 277, 396, 427, 457] and quantization [255, 364] stand as the principal methods for compressing neural networks. Techniques such as those in [208, 252] can achieve a high pruning rate, leading to significant compression.

CHAPTER 4

Training Time Compute Efficiency

4.1 ISD: Self-Supervised Learning by Iterative Similarity Distillation

Recently, contrastive learning has achieved great results in self-supervised learning, where the main idea is to pull two augmentations of an image (positive pairs) closer compared to other random images (negative pairs). We argue that not all negative images are equally negative. Hence, we introduce a self-supervised learning algorithm where we use a soft similarity for the negative images rather than a binary distinction between positive and negative pairs. We iteratively distill a slowly evolving teacher model to the student model by capturing the similarity of a query image to some random images and transferring that knowledge to the student. Specifically, our method should handle unbalanced and unlabeled data better than existing contrastive learning methods, because the randomly chosen negative set might include many samples that are semantically similar to the query image. In this case, our method labels them as highly similar while standard contrastive methods label them as negatives. Our method achieves comparable results to the state-of-the-art models. Our code is available here: https://github.com/UMBCvision/ISD.

4.1.1 Introduction

We can view the recent crop of SSL methods as iterative self-distillation where there is a teacher and a student. Both teacher and student improve simultaneously while the teacher is evolving more slowly (running average) compared to the student: (1) In the case of contrastive methods e.g. MoCo [175], we classify images to positive and negative pairs in the binary form. (2) In the case of clustering methods (DC-v2 [57], SwAV [57], SeLA [502]), the student predicts the quantized representations from the teacher. (3) In the case of BYOL [160], the student simply regresses the teacher's embeddings vector. Here, we introduce a novel method using similarity based distillation to transfer the knowledge from the teacher to the student. We argue that our method is more regularized compared to prior work and improves the quality of the features in transfer learning.

In the standard contrastive setting, e.g. MoCo [175], there is a binary distinction between positive and negative pairs, but in practice, many negative pairs may be from the same category as the positive one.

Thus, forcing the model to classify them as negative is misleading. This can be more important when the unlabeled training data is unbalanced, for example, when a large portion of images are from a small number of categories. Such scenario can happen in applications like self-driving cars, where most of the data is just repetitive data captured from a high-way scene with a couple of cars in it. In such cases, the standard contrastive learning methods will try to learn features to distinguish two instances of the most frequent category that are in a negative pair, which may not be helpful for the down-stream task of understanding rare cases.

We are interested in relaxing the binary classification of contrastive learning with soft labeling, where the teacher network calculates the similarity of the query image with respect to a set of anchor points in the memory bank, converts that into a probability distribution over neighboring examples, and then transfers that knowledge to the student, so that the student also mimics the same neighborhood similarity. In the experiments, we show that our method is competitive with SOTA self-supervised methods on ImageNet and show an improved accuracy when trained on unbalanced, unlabeled data (for which we use a subset of ImageNet).

Our method is different from BYOL [160] in that we are comparing the query image with other random images rather than only with a different augmentation of the same query image. We believe our method can be seen as a more relaxed version of BYOL. Instead of imposing that the embedding of the query image should not change at all due to an augmentation (as done in BYOL), we are allowing the embedding to vary as long as its neighborhood similarity does not change. In other words, the augmentation should not change the similarity of the image compared to its neighboring images. This relaxation lets self-supervised learning focus on what matters most in learning rich features rather than forcing an unnecessary constraint of no change at all, which is difficult to achieve.

Our distillation method is inspired by the CompRess method [240], which introduces an analogous similarity-based distillation method to compress a deeper self-supervised model to a smaller one and get better results compared to training the small model from scratch. Our method is different from [240] in that in our case, both teacher and student share the same architecture, we do self-supervised learning from scratch rather than compressing from another deeper model, and also the teacher evolves over time as a running average of the student rather than being frozen as in [240].

4.1.2 Related Work

Self-supervised learning: The task of learning representations by solving a pretext task without using any supervised annotations is called self-supervised learning. Various pretext tasks like solving jigsaw puzzles [323], predicting rotations [149], counting the visual primitives [324], filling up a missing patch [336], predicting missing channels of input [526, 527], and contrastive learning [165, 175] are explored in the literature. We are proposing a novel pretext task based on iterative similarity based distillation.

Contrastive learning: The task of learning unsupervised representations by contrasting the representations of an image with other images is called contrastive learning [165]. Contrastive learning is essentially positive/negative classification where the positive and negative pairs of embeddings can be defined in various ways. In [70, 175, 480], the positive pairs are augmented views of the same image while the negative pairs are those of different images. In [183], the positive pairs are a patch and context embeddings from the same image while the negatives pairs are a patch and context embedding from different images. In [31, 189], the positives pairs are global and local features from the same image while the negative pairs are global and local features from different images. In [56, 57, 502], the positive pairs are members of the same cluster while the negative pairs are member of different clusters. We are different from these contrastive methods in that we do not consider all negatives equally: we calculate a soft labeling for the negatives using similarity of the data points. Also, we do not consider positive pairs directly: the comparison for the positive pairs is done through the similarity distillation. A few methods have attempted to fix false negative problem of contrastive learning by debiasing the loss [89] and by sampling the local neighborhood as positives [453]. We're different as we simply make the contrastive learning soft. [207] is a concurrent work that identifies some wrong negatives and cancels them in constrictive leaning.

Knowledge Distillation: The task of transferring the knowledge from one model to the other is called knowledge distillation [29, 188]. The knowledge from the teacher can be extracted and transferred in various ways. The knowledge in the activations of intermediate layers can be transferred through regression [373, 500, 516]. While in most works the teacher is a deeper model and the student is a shallower model, in [32, 141] both teacher and the student use the same architecture. Techniques from knowledge distillation can also be used in an unsupervised way to improve self-supervised learning [240, 325, 495]. Instead of using knowledge distillation for model compression or reducing the generalization gap, we use it iteratively to evolve the teacher and student together to learn rich representations from scratch.

Similarity based knowledge distillation: While the above methods [29, 188, 373, 500, 516] only extract the information about a single data point from the teacher, similarity based distillation methods [20, 124,

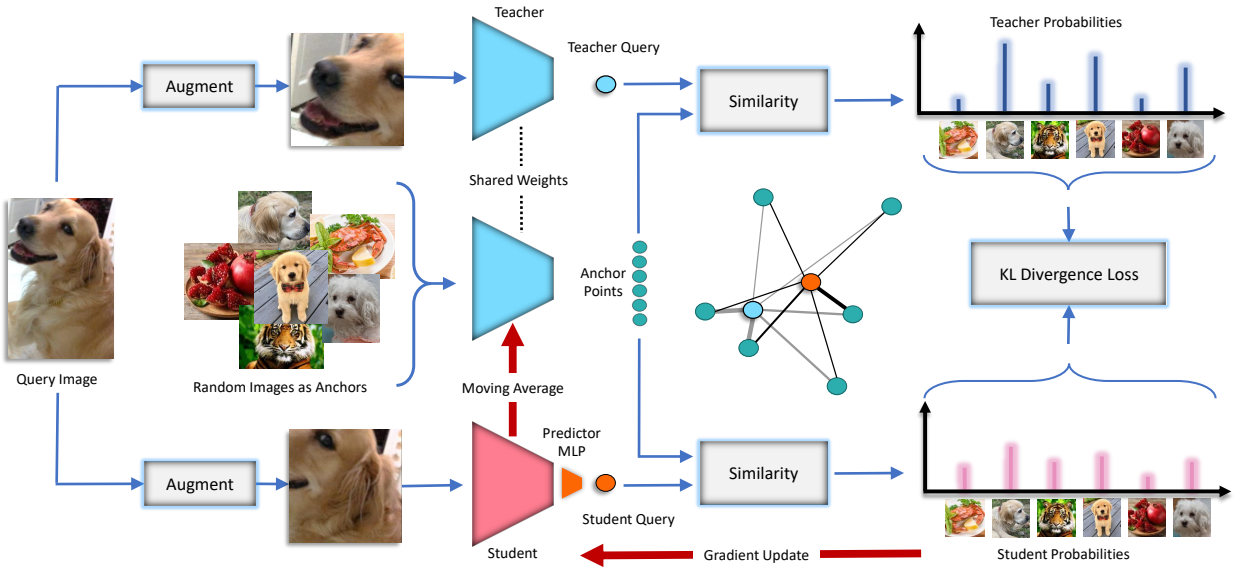


FIGURE 4.1. **Our method:** We initialize both teacher and student networks from scratch and update the teacher as running average of the student. We feed some random images to the teacher, and feed two different augmentations of a query image to both teacher and student. We capture the similarity of the query to the anchor points in the teacher's embedding space and transfer that knowledge to the student. We update the student based on KL divergence loss and update the teacher to be a slow moving average of the student. This can be seen as a soft version of MoCo [175] which can handle negative images that are similar to the query image. Note that unlike contrastive learning and BYOL [160], we never compare two augmentations of the query images directly (positive pair).

240, 332, 334, 340, 425, 444] represent the knowledge from the teacher in terms of similarities between data points. CompRess [240] and SEED [124] are the closest related work to our method that uses similarity based distillation to compresses a large self-supervised model to a smaller one. Our method is different as we use similarity based distillation to iteratively distill an evolving teacher to a student.

Consistency regularization: Consistency regularization is a method of regularization that seeks to make the output of a model consistent across small perturbations in either the input [314] or the model parameters [418]. Recently, BYOL [160] applied a variant of Mean Teachers [418] for self-supervised learning. Our method uses a variation of this idea through similarity based loss rather than regression loss defined on data points individually. [468] is probably the closet to ours that uses a loss similar to our as a regularizer in addition to the MoCo loss. Our method is different as we optimize our loss from scratch as the main objective without adding it to another method. Also, our method benefits from different temperatures for the teacher and student networks which is inspired by [124].

4.1.3 Method

We are interested in learning rich representations from unlabeled data. We have a teacher network and a student network. We initialize both models from scratch and update the teacher to be a slower version of the student: we use the momentum idea from MoCo in updating the teacher so that it is running average of the student. The method is described in Figure 4.1. Following the notation in [240], at each iteration, we pick a random query image and a bunch of random other images that we call anchor points. We augment those images and feed them to the teacher model to get their embeddings. Then, we augment the query again independent of the earlier augmentation and feed it to the student model only. We calculate the similarity of the query point compared to the anchor points in the teacher's embedding space and then optimize the student to mimic the same similarity for the anchor point at the student's embedding space. Finally, we update the teacher with a momentum to be the running average of the student similar to MoCo and BYOL. Note that our method is closely related to ComPress method [240] which uses the similarity distillation for compressing a frozen larger model to a smaller one.

More formally, we assume a teacher model t and a student model s. Given a query image q, we augment it twice independently to get q_t and q_s. We also assume a set of n augmented random images $\{x_i\}_{i=1}^n$. We feed $\{x_i\}_{i=1}^n$ to the teacher model to get their embeddings $\{t(x_i)\}_{i=1}^n$ and call them anchor points. We also feed q_t to the teacher and q_s to the student to get $t(q_t)$ and $s(q_s)$ respectively. Then, we calculate the similarity of the query embedding $t(q_t)$ compared to all anchor points, divide by a temperature, and convert to a probability distribution using a SoftMax operator to get:

$$p_t(i) = -\log \frac{\exp\big(\text{sim}\big(t(q_t), t(x_i)\big)/\tau_t\big)}{\sum_{j=1}^n \exp\big(\text{sim}\big(t(q_t), t(x_j)\big)/\tau_t\big)}$$

where τ_t is the teacher's temperature parameter and $\text{sim}(.,.)$ refers to the similarity between two vectors. In our experiments, we use cosine similarity which is standard in most recent contrastive learning methods.

Then, we calculate a similar probability distribution for the student's query embedding to get:

$$p_s(i) = -\log \frac{\exp\big(\text{sim}\big(s(q_s), t(x_i)\big)/\tau_s\big)}{\sum_{j=1}^n \exp\big(\text{sim}\big(s(q_s), t(x_j)\big)/\tau_s\big)}$$

Where, τ_s is the student's temperature parameter. Finally, we optimize the student only by minimizing the following loss:

$$L = KL(p_t || p_s)$$

and the teacher is updated using the following rule:

$$\theta_t \leftarrow m\theta_t + (1-m)\theta_s$$

where θ refers to the parameters of a model and m is a the momentum hyperparameter that is set be close to one (0.99 in our ResNet50 experiments) as in MoCo. Since the teacher is not optimized by the loss directly, the loss can be simplified as cross entropy loss instead of KL divergence.

Note that unlike the positive pair in contrastive learning methods and BYOL, the query image is never compared to its own augmentation as it is not included in the anchor points. We do this since when the features are mature, the similarity of the query to itself will be very large and will dominate the whole probability distribution. Note that one can convert our method to MoCo by including the query in anchor points and replacing the probability of the teacher p_t with a one-hot encoding vector in which the positive pair (query) corresponds to one and all other anchor points correspond to zero. Figure 4.2 shows some example teacher probabilities for both ISD and MoCo.

Our method can benefit from a large number of anchor points to cover the neighborhood of any query image, and also the anchor points are fed to the teacher only that evolves slowly. Hence, we use MoCo's trick of a large memory bank (queue) for the anchor points. The queue size is 128K in our experiments which uses only 1.6% of the total memory and less than 1% of the total computation.

Different Temperature for student and teacher: Since the student is learning from the teacher, we can use a lower temperature for the teacher compared to the student to make the teacher more confident. In the extreme case, when the teacher uses zero temperature, its output will be a one-hot encoding over the anchor points which is a very sharp distribution. In the experiments we observe best results when the teacher has 10 times smaller temperature.

4.1.4 Experiments

We describe various experiments and their results in this section. We compare our proposed self-supervised method with other state-of-the-art methods on ImageNet and transfer learning. We also demonstrate the advantage of our method compared to MoCo on unbalanced, unlabeled dataset.

Implementation details: For all experiments, we use PyTorch with SGD optimizer (momentum = 0.9, weight decay = $1e{-}4$, batch size = 256) except when stated otherwise. Details about the specific architecture, epochs for training, and learning rate are described for each experiment in its respective section. We follow the evaluation protocols in [240] for nearest neighbor (NN) and linear layer (Linear) evaluation. We use the ImageNet labels only in the setting of evaluating the learned features. To evaluate how SSL features

FIGURE 4.2. **Positives vs. negatives:** We sample some query images randomly (left column), calculate their teacher probability distribution over all anchor points in the memory bank (size=128K) and rank them in descending order (right columns). The second left column is another augmented version of the query image that contrastive learning methods use for the positive pair. Our students learns to mimic the probability number written below each anchor image while contrastive learning method (e.g., MoCo) learn to predict the one-hot encoding written below the images. Note that there are lots of images in the top anchor points that are semantically similar to the query point that MoCo tries to discriminate them from the query while our method does not.

transfer to new tasks, we perform Linear layer evaluation on different datasets including Food101 [41], SUN397 [481], CIFAR10 [245], CIFAR100 [245], Cars [244], Flowers [321], Pets [333], Caltech-101 [130] and DTD [90]. More details about the datasets and training can be found in the appendix. We follow [160] setting for transfer learning and reproduce BYOL results for fairness.

4.1.4.1 Self-supervised learning

BYOL-asym (baseline). ResNet-50 is recently used as a benchmark in the community. Unfortunately, we cannot run it for 1000 epochs because of resource constraints. Some methods [57, 160] are even slower as they forward the mini-batch through the model more than once. For instance, BYOL method forwards the images twice to calculate the symmetric loss, so 100 epochs of symmetric BYOL is equivalent to almost 200 epochs of asymmetric BYOL in terms of running time. As shown in [77], given a constant budget, there is no big difference between symmetric and asymmetric losses. Thus, for a fair comparison with our method and MoCo, we use asymmetric loss, a small batch size (256), momentum for the teacher is 0.99, and

train for 200 epochs. We implement BYOL in PyTorch following [160]. We call this baseline as BYOL-asym since it's asymmetric version of BYOL. For ResNet18, hidden units in the MLP for projection and prediction layers is 1024, and output embedding dimension is 128. For ResNet50, hidden units in the MLP for projection and prediction layers is 4096, and output embedding dimension is 512. We use cos learning rate scheduler with initial learning rate of 0.05.

ResNet-18 experiments: Following CompRess [240], we train our self-supervised ResNet-18 model with the initial learning rate set to 0.01 and multiplied by 0.2 at epochs 140 and 180. We follow [160] and add a prediction layer for the student. The hidden and output dimensions of the prediction MLP layer are set to 512. We do not have any projection layer for ResNet18. We use the same set of augmentations used in [70, 76, 160]. We use the same temperature for teacher and student $\tau_s = \tau_t = 0.02$. The memory bank size is 128K and momentum m for teacher encoder is 0.999. We choose these parameters based on ablations which can be found in the appendix. The results are shown in Tables 4.1 and 4.2. Our model outperforms both baselines on full ImageNet linear and transfer linear benchmarks.

It is important to note that our method can be seen as a soft version of MoCo. So, ISD outperforming MoCo, empirically supports our main motivation of improving representations by smoothing the contrastive learning: not considering all negatives equally negative.

ResNet-50 experiments: We train ResNet-50 with different settings than ResNet-18. We use same architecture and settings as ResNet-50 BYOL-asym. We use cos learning rate scheduler with initial learning rate of 0.05. We use temperature of $\tau_s = 0.1$ for the student and $\tau_t = 0.01$ for the teacher. Memory bank size is 128K, and momentum m for teacher encoder is 0.99. We study the effect of memory bank size in Figure 4.3 which shows that memory bank size of even 16K is on-par with 128K and above. Additionally, inspired by [403], we train our model with two different augmentation sets which we call it weak (random cropping and random horizontal flipping) and strong (same as [70, 76, 160]). The teacher view uses weak augmentation while the student view uses the strong augmentation. We evaluate the effect of different augmentation in Table 4.4. ResNet50 results are shown in Tables 4.1 and 4.2.

Tables 4.1 and 4.2 show the results on ImageNet and transfer learning settings respectively. Our method is comparable to SOTA SSL methods including BYOL in Linear and nearest neighbor evaluation on ImageNet. As mentioned earlier, we believe our method is more relaxed compared to BYOL as our method lets the embeddings of augmented images move as long as their similarity relationship with neighbors has not changed. Table 4.3 shows our results when only limited labels are available in ImageNet dataset. Figure 4.5

TABLE 4.1. **Evaluation on full ImageNet:** We compare our method with other state-of-the-art SSL methods by evaluating the learned features on the full ImageNet. A single linear layer is trained on top of a frozen backbone. Note that methods using symmetric losses use 2× computation per mini-batch. Thus, it is not fair to compare them with the asymmetric loss methods. Further, we find that given a similar computational budget, both asymmetric MoCo-V2 (400 epochs) and symmetric MoCo-V2 (800 epochs) have similar accuracies (71.0 vs 71.1). Under similar resource constraints, our method performs competitively with other state-of-the-art methods. * is compressed from ResNet-50x4. †: SwAV is not comparable as it uses multiple crops together. ‡: is our concurrent work.

Method	Ref	Batch Size	Epochs	Sym. Loss 2x FLOPS	Top-1 Linear	NN	20-NN
ResNet-50							
Supervised	-	256	100	-	76.2	71.4	74.8
SwAV [57]	[77]	4096	200	✓	69.1	-	-
SimCLR [70]	[70]	4096	1000	✓	69.3	-	-
MoCo-V2 [175]	[77]	256	200	✓	69.9	-	-
SimSiam [77]	[77]	256	200	✓	70.0	-	-
BYOL [160]	[77]	4096	200	✓	70.6	-	-
MoCo-V2 [76]	[76]	256	400	✓	71.0	-	-
MoCo-V2 [175]	[175]	256	800	✗	71.1	57.3	61.0
CompRess* [240]	[240]	256	1K+130	✗	71.9	63.3	66.8
BYOL [160]	[160]	4096	1000	✓	74.3	62.8	66.9
SwAV † [57]	[57]	4096	800	✓	**75.3**	-	-
MoCo-V2 [175]	[77]	256	200	✗	67.5	-	-
CO2 [468]	[468]	256	200	✗	68.0	-	-
BYOL-asym	-	256	200	✗	69.3	55.0	59.2
MSF ‡ [242]	[242]	256	200	✗	**72.4**	**62.0**	**64.9**
ISD	-	256	200	✗	69.8	59.2	62.0
ResNet-18							
Supervised	-	256	100	-	69.8	63.0	67.6
MoCo-V2 [175]	[77]	256	200	✗	51.0	37.7	42.1
BYOL-asym	-	256	200	✗	52.6	40.0	44.8
ISD	-	256	200	✗	**53.8**	**41.5**	**46.6**

shows random image samples from random clusters where each row corresponds to a cluster. Note that each row contains almost semantically similar images.

Evolution of teacher and student models: In Figure 4.4, for every 10 epoch of ResNet-18, we evaluate both teacher and student models for BYOL, MoCo, and ISD methods using nearest neighbor. For all methods, the teacher performs usually better than the student in the initial epochs when the learning rate is relatively large and then is very close to the student when it shrinks. This is interesting as we have not seen previous papers comparing the teacher with the student. This might happen since the teacher is a running average of the student so can be seen as an ensemble over many student networks similar to [418]. We believe this deserves more investigation as future work.

TABLE 4.2. **Linear transfer evaluation:** We linear classifiers on top of frozen features for various downstream datasets. Hyperparameters are tuned individually for each method and the results are reported on the hold-out test sets. Our ResNet-18 is significantly better than other state-of-the-art SSL methods. "rep." refers to the reproduction with our framework for a fair comparison. ‡: our concurrent work.

Method	Ref.	Epochs	Food 101	CIFAR 10	CIFAR 100	SUN 397	Cars 196	DTD	Pets	Caltech 101	Flowers 102	Mean
ResNet-50												
Sup-IN	[160]		72.3	93.6	78.3	61.9	66.7	74.9	91.5	94.5	94.7	80.9
SimCLR [70]	[160]	1000	72.8	90.5	74.4	60.6	49.3	75.7	84.6	89.3	92.6	68.6
MoCo v2 [76]	-	800	72.5	92.2	74.6	59.6	50.5	74.4	84.6	90.0	90.5	76.5
BYOL [160]	rep.	1000	**75.4**	**92.7**	78.1	62.1	67.1	**76.8**	89.8	92.2	95.5	81.1
BYOL [160]	[160]	1000	75.3	91.3	**78.4**	**62.2**	**67.8**	75.5	**90.4**	**94.2**	**96.1**	**81.2**
BYOL-asym [160]	-	200	70.2	91.5	74.2	59.0	54.0	73.4	86.2	90.4	**92.1**	76.8
MoCo v2 [76]	-	200	70.4	91.0	73.5	57.5	47.7	**73.9**	81.3	88.7	91.1	75.0
MSF ‡ [242]	[242]	200	**71.2**	**92.6**	**76.3**	**59.2**	**55.6**	73.2	88.7	**92.7**	92.0	**77.9**
ISD	-	200	68.6	90.8	72.0	55.8	45.8	68.6	**89.1**	90.3	87.4	74.3
ResNet-18												
BYOL-asym [160]	-	200	55.0	**83.4**	59.3	48.2	26.6	65.4	74.1	82.7	82.3	64.1
MoCo v2 [76]	-	200	56.7	83.0	59.7	48.8	30.4	64.4	70.1	80.5	83.1	64.1
ISD	-	200	**58.3**	83.3	**62.7**	**49.6**	**36.1**	**65.6**	**76.4**	**84.5**	**87.4**	**67.1**

TABLE 4.3. **Evaluation on limited labels ImageNet for ResNet-50:** We evaluate our model for the 1% and 10% ImageNet linear evaluation. Unlike other methods, we only train a single linear layer on top of the frozen backbone. We observe that our method is better than other state-of-the-art methods given similar computational budgets. * is compressed from ResNet-50x4

Method	Epochs	Top-1		Top-5	
		1%	10%	1%	10%
Entire network is fine-tuned.					
Supervised		25.4	56.4	48.4	80.4
PIRL [313]	800	-	-	57.2	83.8
CO2 [468]	200	-	-	71.0	85.7
SimCLR [70]	1000	48.3	65.6	75.5	87.8
InvP [453]	800	-	-	78.2	88.7
BYOL [160]	1000	53.2	68.8	78.4	89.0
SwAV† [57]	800	**53.9**	**70.2**	**78.5**	**89.9**
Only the linear layer is trained.					
BYOL‡ [160]	1000	55.7	**68.6**	80.0	**88.6**
CompRess* [240]	1K+130	**59.7**	67.0	**82.3**	87.5
MoCo v2 [76]	200	43.6	58.4	71.2	82.9
BYOL-asym [160]	200	47.9	61.3	74.6	84.7
ISD	200	**53.4**	**63.0**	**78.8**	**85.9**

Ablation study: We varied the temperature for our method on ResNet-18 with 130 epochs and reported the results in Table 4.5. Here, *LR* = 0.01 and it is multiplied by 0.2 at 90 and 120 epochs. Also, for more fair comparison with BYOL on ResNet18, we varied the learning rate and chose the best one for BYOL. Table 4.6 shows the results of this experiment.

TABLE 4.4. **Effect of augmentation strategies**: Effect of using weak or strong augmentations for ResNet-50 trained with 200 epochs.

Method	Student Aug.	Teacher Aug.	NN	20-NN
ISD	weak	weak	40.4	43.5
ISD	strong	weak	22.9	26.3
ISD	strong	strong	58.0	61.2
ISD	weak	strong	**59.2**	**62.0**

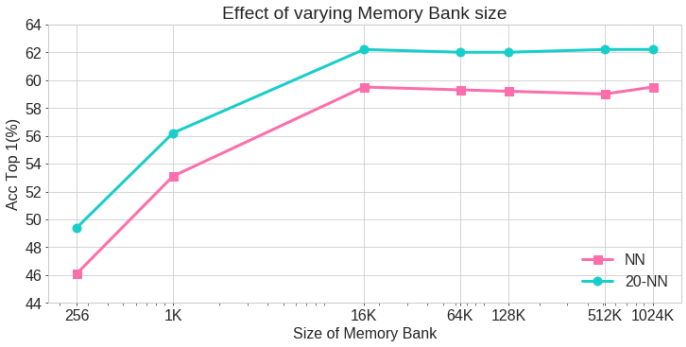


FIGURE 4.3. **Effect of Memory Bank Size:** We study the effect of memory bank size by varying from 256 to 1024K for ISD on ResNet-50 model.

TABLE 4.5. **Effect of temperature:** Effect of changing temperature for our method ISD on ResNet-18 model.

τ	0.003	0.007	0.01	0.02	0.04	0.06
NN	37.2	37.8	37.7	**39.7**	35.3	32.5

TABLE 4.6. **Effect of learning rates for BYOL:** Comparison of different learning rates for BYOL on ResNet-18 with 200 epochs and cosine learning rate scheduler.

LR	0.01	0.05	0.10	0.20
NN	37.3	**40.0**	38.6	37.3

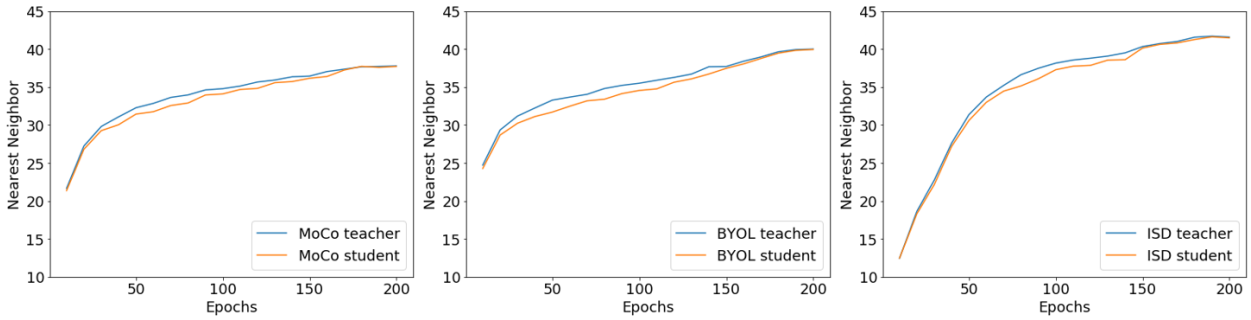


FIGURE 4.4. **Evolution of teacher and student models:** Comparing the teacher and student ResNet18 models using Nearest Neighbor while training for MoCo, BYOL, and ISD methods. Interestingly, the teacher performs better than the student before shrinking the learning rate. Most previous works use the student as the final model which seems to be sub-optimal. We believe this is due to ensembling effect similar to [418] and needs more investigation.

4.1.4.2 Self-Supervised Learning on Unbalanced Dataset

Most recent self-supervised learning methods are benchmarked by training on unlabeled ImageNet. However, we know that ImageNet has a particular bias of having an almost uniform distribution over the number of samples per category. We believe this bias does not exist in many real-world applications where the data is unbalanced: a few categories have a large number of samples while the rest of the data have a small number of samples. For instance, in self-driving car applications, it is really important to learn features

FIGURE 4.5. **Random Clusters:** We cluster ImageNet dataset into 1000 clusters using k-means and show random samples from random clusters. We did not do cherry-picking for this visualization. Each row corresponds to a cluster. Note that semantically similar images are clustered together. More results can be found in the appendix.

for understanding rare scenes while most of the data is captured from repetitive safe highway scenes. Hence, we believe it is important to design and evaluate self-supervised learning methods for such unbalanced data.

As mentioned earlier, since standard contrastive learning methods e.g., MoCo, consider all negative examples equally negative, when the query image is from a large category, it is possible to have multiple samples from the same category in the memory bank. Then, the contrastive loss in MoCo pushes their embeddings to be far apart as negative pairs. However, our method can handle such cases since our teacher assigns a soft label to the negative samples, so if an anchor example is very similar to the query, it will have high similarity and the student is optimized to reproduce such similarity.

To study our method on unbalanced data, we design a controlled setting to introduce the unbalanced data in the SSL training only and factor out its effect in the feature evaluation step. Hence, we sub-sample ImageNet data with 38 random categories where 8 categories are large (use all of almost 1300 images per category) and 30 categories are small (use only 100 images per category.) We train our SSL method and then evaluate by nearest neighbor (NN) classifier on the balanced validation data. To make sure that the feature evaluation is not affected by the unbalanced data, we keep both evaluation and the training data of NN search balanced, so for NN search, we use all ImageNet training images (almost 1300×38 images) for those 38 categories.

We repeat the sampling of 38 categories 10 times to come up with 10 datasets and report the results for our method and also MoCo in Table 4.7. To measure the effect of the unbalanced data. we report the accuracy on all 38 categories and also on those 30 small categories only separately. Our method performs

TABLE 4.7. **Unbalanced dataset:** Nearest Neighbor (NN) results with ResNet-18 model for the unbalanced data when we consider all 38 categories and 30 small categories separately. We repeat the experiment 10 times with different random sets of 38 categories. NN is done on the validation set of ImageNet (which has uniform distribution) by searching the nearest neighbors among all ImageNet training data of those 38 categories (so the training data of NN also has uniform distribution). Hence, the whole evaluation is on balanced data to make sure we observe the effect of the unbalanced, "unlabeled" data only. "Diff" shows the improvement of our method over MoCo. Interestingly the improvement is bigger in the rare categories. This is aligned with out hypothesis that our method can handle unbalanced, unlabeled data better since it does not consider all negative images equally negative.

Method	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	D_{10}	Mean
Evaluation On All 38 Categories											
MoCo	48.9	57.8	59.4	56.7	62.0	54.6	57.8	57.2	62.4	54.4	57.12
ISD	49.6	57.9	61.9	58.6	62.3	56.3	57.8	58.1	62.5	55.3	58.03
Diff	+0.7	+0.1	+2.5	+1.9	+0.3	+1.7	0	+0.9	+0.1	+0.9	+0.91
Evaluation Only on 30 Rare Categories											
MoCo	44.7	52.3	57.3	53.1	57.7	50.7	51.1	51.9	58.9	59.8	53.75
ISD	46.5	53.9	60.8	56.8	60.5	54.5	53.1	55.0	60.7	61.5	56.33
Diff	+1.7	+1.6	+3.5	+3.7	+2.8	+3.8	+2.0	+3.1	+1.8	+1.7	+2.57

consistently better than MoCo, but more interestingly, the gap the improvement is larger when we evaluate on the 30 small categories only. We believe this empirically proves our hypothesis that our method may be able to handle unbalanced data more effectively. For a fair comparison, we train both our model and MoCo for 400 epochs with a memory bank size of 8192 and cosine learning schedule.

4.1.5 Conclusion

We introduce ISD, a novel self-supervised learning method. It is a variation of contrastive learning (e.g., MoCo) in which negative samples are not all treated equally. The similarity between images in the teacher's embedding space determines how much each anchor image should be contrasted with. Our extensive experiments show that our method performs comparable to the state-of-the-art SSL methods on ImageNet, transfer learning tasks, and when the unlabeled data is unbalanced.

4.2 Mean Shift for Self-Supervised Learning

Most recent self-supervised learning (SSL) algorithms learn features by contrasting between instances of images or by clustering the images and then contrasting between the image clusters. We introduce a simple mean-shift algorithm that learns representations by grouping images together without contrasting between them or adopting much of prior on the structure or number of the clusters. We simply "shift" the embedding of each image to be close to the "mean" of the neighbors of its augmentation. Since the closest neighbor is always another augmentation of the same image, our model will be identical to BYOL when using only one nearest neighbor instead of 5 used in our experiments. Our model achieves 72.4% on ImageNet linear evaluation with ResNet50 at 200 epochs outperforming BYOL. Also, our method outperforms the SOTA by a large margin when using weak augmentations only, facilitating adoption of SSL for other modalities. Our code is available here: https://github.com/UMBCvision/MSF

4.2.1 Introduction

Most current visual recognition algorithms are supervised, meaning that they learn from large scale annotated images or videos. However, in many applications, the annotation process may be expensive, biased, ambiguous, or involve privacy concerns. Self-supervised learning (SSL) algorithms aim to learn rich representations from unlabeled images or videos. Such learned representations can be used along with small annotated data to provide an accurate visual recognition model. We are interested in developing better SSL models using unlabeled images.

Some recent SSL models learn by contrasting between instances of images. They pull different augmentations of an image instance together while pushing them away from other image instances [70, 175]. Some other SSL methods cluster the unlabeled images to a set of clusters with the hope that each cluster will contain semantically similar images. Then, a model that predicts those clusters learns rich representations similar to supervised learning with labels [56, 57, 502].

These clustering methods also can be considered as contrastive learning since they contrast between different clusters of images. For instance, the SoftMax layer in deep clustering method [56] encourages an image to be assigned to the correct single cluster and not the other clusters.

Also, most clustering algorithms have strong priors on the overall structure of the clusters. For instance, deep clustering (k-means) using Euclidean distance encourages spherical cluster shapes which we believe is unnecessary for the purpose of SSL methods.

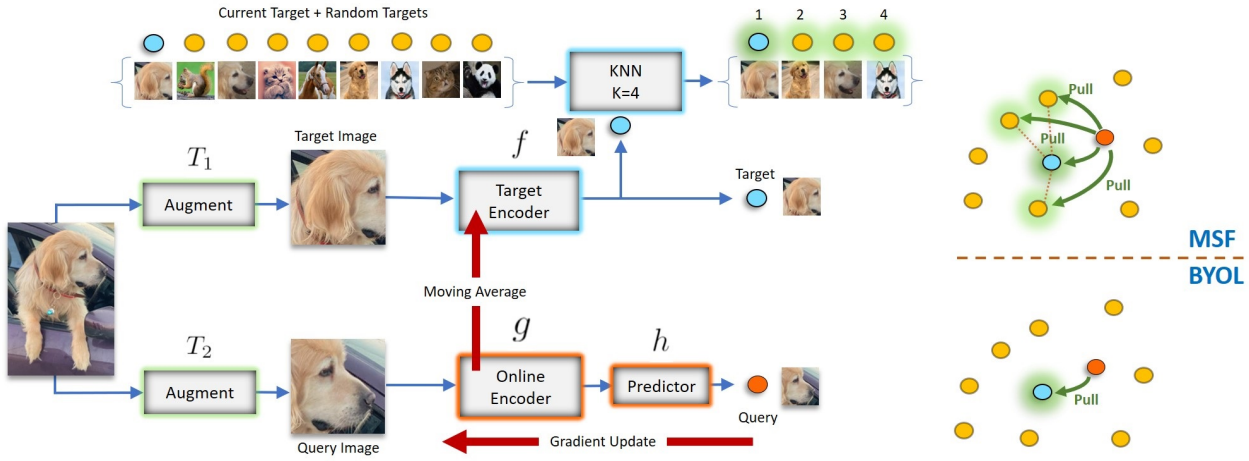


FIGURE 4.6. **MSF method:** Similar to BYOL, we maintain two encoders ("target" and "online"). The online encoder is updated with gradient descent while the target encoder is the moving average of the online encoder. We augment an image twice and feed to both encoders. We add the target embedding to the memory bank and look for its nearest neighbors in the memory bank. Obviously target embedding itself will be the first nearest neighbor. We want to shift the embedding of the input image towards the mean of its nearest neighbors, so we minimize the summation of those distances. Note that our method using only one nearest neighbor is identical to BYOL which pulls different augmentations together without grouping different instances of images. To our knowledge, our method is the first in grouping different instances of images without contrasting between image instances or clusters.

Recently, BYOL [160] showed that it is possible to learn rich representations without contrasting between image instances. BYOL [160] works by simply pulling the two views of an image closer without any contrast with other images. The better performance of BYOL [160] compared to MoCo hints that contrasting with other images may be a limiting constraint. For instance, in MoCo [175], since the negative images are sampled randomly, they may be from the same category as the query, resulting in degraded representations. [420] tries to resolve this issue by not treating all negatives equally negative.

Inspired by mean shift clustering, we generalize BYOL to a simple yet effective SSL method where a data point is pulled closer to not only its other augmentations but also the nearest neighbors (NNs) of its augmentation. Unlike DeepCluster [56], SwAV [57], and SeLA [502] that use explicit, mutually exclusive cluster assignment, our method uses mean-shift algorithm that groups similar images together locally without explicit cluster assignment. Moreover, unlike k-means clustering, mean-shift does not have strong priors on the shape, size, or number of the clusters. This makes mean-shift suitable for SSL where such priors are not known. Compared to MoCo [175], SimCLR [70], SwAV [57] and few others, our method never contrasts between different images or even cluster centers.

Since we need a large set of embeddings to search for nearest neighbors, we adopt the memory bank idea [175] to maintain a random set of embeddings. Also, since the model is evolving over time in the

FIGURE 4.7. **Nearest neighbors (NN) of the model at each epoch:** For a random query images, we show how the nearest neighbors evolve at the learning time. Initially, NNs are not semantically quite related, but are close in low-level features. The accuracy of 1-NN classifier in the initialization is 1.5% which is 15 times larger than random chance (0.1%). This little signal is bootstrapped in our learning method and results in NNs of the late epochs which are mostly semantically related to the query image. More visualizations can be found in the appendix.

learning process, the old elements in the memory bank will not be valid, so we adopt the momentum idea from [175] to maintain two encoders ("target" and "online") instead of only one. The online model is updated by the loss and the target model is updated as a moving average of the online model. We feed two different augmentations of an image to these two encoders, then we push the online embedding of the image to be close to the average of nearest neighbors of the target encoding of the image in the target embedding space. Hence, similar to most recent SSL methods, our method also uses the inductive bias that the augmentation should not move the embedding much.

Our experiments show that our method outperforms state-of-the-art methods on various settings. For instance, when trained on unlabeled ImageNet for 200 epochs, it achieves 72.4% linear ImageNet accuracy which is better than BYOL at 200 epochs.

Most recent SSL methods use strong augmentations to improve the accuracy, leading to "augmentation engineering" to improve SSL. However, in many applications, e.g., medical domain, designing such augmentations is not easy and needs extensive domain knowledge. Hence, designing SSL methods that do not rely heavily on large variations of augmentations is interesting. We show that when using only weak augmentations, our method (MSF w/w) outperforms BYOL by a large margin. We hypothesize that NNs act as a proxy for strong augmentations of the query image, so there is no need for engineering strong augmentations.

4.2.2 Related Work

Self-supervised learning: The aim of self-supervised learning is to learn representations directly from the data without using any manual data annotation. Specifically, a pretext task is designed based on the inherent structure in the data and a model is trained to solve it. Various pretext tasks have been designed that exploit different structural cues in the data. Here, we focus on following pretext tasks for images: treating each data point as a single class to perform instance level classification [114], predicting the relative location of patches [108, 323], filling up a missing patch in an image [336], predicting a colored image from its grayscale version [526, 527], counting objects in an image [324], predicting the rotation of an image [149], and predicting the pseudo-labels obtained from clustering [56, 502]. Note that designing the pretext task or the augmentations itself is still manual and needs domain knowledge.

Instance discrimination: Recently, the task of instance discrimination [114] has shown great promise. The basic idea is to treat each image as single class. This is also referred as contrastive learning where positive samples (augmented views of the same instance) are pushed close and away from the negative samples (all other instances). While [114] took a parametric approach for this classification, [480] took a non-parametric approach. The non-parametric approach has seen broad adoption with great results [57, 70, 175, 313, 424, 545]. Two important components of these methods are: memory bank (source of negative samples) and augmentation (constructing positive samples). A simple but effective technique using a momentum encoder to populate the memory bank is proposed in [175]. A rigorous study of the impact of different augmentations and hyperparameters is conducted in [70]. Improved augmentation strategies are proposed in [57, 70, 426]. Instance discrimination can also be viewed from an information theoretic perspective as the task of maximizing the information between different views of a single image [31, 189, 445].

Consistency regularization: Although negative pairs were thought to be central in preventing the collapse of representations for instance discrimination, [160] proposed a method that does not collapse despite not using any negatives. The objective in [77, 160] simply pulls augmented views of the same image close without any contrast with negative samples. This is also referred to as consistency regularization in the semi-supervised learning framework [418]. Inspired by [160], we propose a more general form of it where the positives can also come from the close neighborhood of a sample grouping similar images together.

Clustering methods: Another class of methods based on clustering have shown promise. The basic idea is to alternate between clustering and learning the representations [483, 496]. This approach was first scaled to large scale pre-training in [108]. A big concern in these methods is to prevent the collapse of all

representations into a single cluster. To that end, an optimal transport based formulation of clustering is proposed in [502]. An online clustering algorithm based on the formulation of [502] is proposed in [57].

Clustering and instance discrimination: Clustering methods can be seen as the generalizing the instance discrimination framework. Only the views from the same sample can be positives in instance discrimination [114, 480], but all the members (and their views) of a cluster are positives in clustering based methods [35, 57]. A more flexible middle ground is where the set of positives are based on the local neighborhood of a sample: top-k nearest neighbors of a sample in [204], nearest neighbors of a sample that are also the members of the same cluster in [545], and top-k graph distance based neighbors in [453]. Our method shares the motivation behind these works: embeddings should be locally clustered around high density regions. We also use top-k nearest neighbors as positives [204,453], but our method is fundamentally different as there is no notion of negatives in our method. Intuitively, we enforce a more simpler and flexible constraint: move each sample closer to its nearest neighbors in each iteration. This idea is inspired from Mean-Shift Clustering [82, 91] where the cluster assignment for each sample is iteratively updated to be the mean of its nearest neighbors. In contrast to the k-means clustering, Mean-Shift does not make strong assumptions about the shape of clusters.

4.2.3 Method

We are interested in mean-shift clustering, so at each iteration we want to encourage the model to shift the embedding of an image to be closer to the average of its nearest neighbors on a large random set of samples.

Following the notation of BYOL [160], we assume a target encoder f and an online encoder g. Both encoders have the same backbone architecture followed by a projection layer and are initialized equally. The online encoder g is followed by an additional prediction layer h on top of it. The online encoder g and the prediction layer h are updated by back-propagating the loss while the target encoder f is updated by momentum update to be a running average of the online encoder g. Since nearest neighbor needs a large pool of examples, we maintain a first-in-first-out (FIFO) memory bank [175] that includes recent embeddings from the slowly evolving target encoder f.

Given an unlabeled image x, we augment it randomly twice to get $T_1(x)$ and $T_2(x)$. We feed them to encoders and then normalize them with ℓ_2 norm to get $u = \frac{f(T_1(x))}{\|f(T_1(x))\|_2}$ and $v = \frac{h(g(T_2(x)))}{\|h(g(T_2(x)))\|_2}$. We first add u to the memory bank and then, find the k nearest neighbors of u in the memory bank to get a set of embeddings $\{z_j\}_{j=1}^{k}$. Note that this set includes u itself. Since we know it is another augmentation of the same input

image, it should be a good target for v. Finally, we minimize the following loss:

$$L = \frac{1}{k} \sum_{j=1}^{k} dist(v, z_j)$$

where, $dist(.,.)$ is the distance metric between two embeddings. We use MSE loss ($dist(a,b) = ||a-b||_2^2$) as the distance in our experiments. Minimizing this loss is equivalent to maximizing Cosine similarity as the vectors are already ℓ_2 normalized. The final loss is the summation of the above loss for all input images.

Ideally, we can average the set of nearest neighbors to come up with a single target embedding, but since averaging is dependent on the choice of loss function, we simply minimize the summation of distances. Note that for Euclidean distance, both methods result in identical gradients.

Since u itself is included in our NN search, it will be always the best nearest neighbor. Hence, our method with $k = 1$ will be identical to BYOL which minimizes $||v-u||_2^2$ for each image without using a memory bank.

Moreover, in the initial stages of the learning, v may be far from u and the other $k-1$ nearest neighbors may be semantically different from the query image. Since those wrong neighbors are still close to u, the loss will still pull v closer to the neighborhood of u (another augmentation of v).

In later stages of learning, when the representation is more mature, the other $k-1$ neighbors will be semantically related and will contribute to learning since u and v are already closer to each other. Table 4.8-right and Figure 4.7 show that the representation improves as the learning progresses.

Strength of augmentation: In most exemplar-based SSL methods, augmentation plays an important role since the main supervision signal is that the augmentation should not change the embedding much. Hence, recent methods, e.g., MoCo v2, SimCLR, and BYOL, use strong augmentations. We believe such aggressive augmentations on the target embeddings u may add randomness to the learning process as some of those augmentation do not look natural, so the nearest neighbors will not be semantically close to the query image. Hence, we use weaker augmentations for the target model to make u and z less noisy while still using strong augmentations for the online model. We refer to this as the weak/strong ("w/s") variation. This is inspired by [403] which uses weak augmentations in semi-supervised learning. This variation results in almost one point improvement over the regular variation where both encoders use strong augmentations. As shown in Fig. 4.8 (right), the nearest neighbors are more pure in the "weak only" setting which is consistent with our above intuition. Our experiments show that BYOL also benefits from w/s augmentation to some extent. This is probably due to more robust target encoding. Finally, we explore a weak/weak "w/w" variation where both views are augmented with a weak augmentation.

4.2.4 Experiments

We report the results of our self-supervised learning and transfer learning in this section.

4.2.4.1 Self-supervised Learning

Mean Shift (MSF): We use 0.99 for the the momentum of the target encoder, top-k = 5, and 1.024M for the memory bank size (which is roughly the same as the size of ImageNet dataset). Our ablation study shows that a memory bank of 128K does not degrade the results. We observe that the added computational cost of NN search is small compared to the overall forward and backward passes. We find that MSF with 128K memory bank size and 512 dimensions for the embedding, uses less than 0.5GB of extra GPU memory for the memory bank and less than 1% of extra computation for finding 5 NNs (see Table 4.13).

BYOL-asym (baseline): Training SSL methods for more than 200 epochs is not easy due to resource constraints. For instance, training BYOL with ResNet50 for 200 epochs takes roughly 7 days on four RTX 2080-Ti GPUs. Thus, for a fair comparison, we re-implement BYOL in our own framework and call it BYOL-asym. We note and justify the major differences between BYOL-asym and BYOL here. First, we use an asymmetric loss. The original BYOL paper [160] uses symmetric loss which passes each view of the image through both encoders. As a result, the gradient is calculated over $2 \times B$ instances where B is the batch size, so each epoch needs twice computation compared to asymmetric loss. Hence, 200 epochs of BYOL-asym should be compared with 100 epochs of regular BYOL. Second, we use a small batch size of 256 instead of 4096. [160] shows that BYOL works well even with the batch size of 256. Third, we use SGD optimizer instead of LARS. Despite these differences, our implementation works reasonable well compared to reported results in prior work. Our MSF uses the same setup for fairness.

Augmentation: In all of our experiments, "strong" augmentation refers to the augmentation in MoCo v2 [76]. The strong augmentation involves the following stochastic operations: grayscale, color jitter, horizontal flip, and Gaussian blur. The "weak" augmentation is simply a random crop of size 224×224 with area ratio between 0.2 and 1.0 followed by random horizontal flipping with probability 0.5. **MSF w/s** refers to our "weak/strong" variation where the target encoder view is augmented with the weak augmentation and online encoder view is augmented with the strong augmentation. **MSF w/w** refers to our "weak/weak" variation where both teacher and student views use weak augmentation. **BYOL-asym** and **MSF** use the standard SSL practice of augmenting both views with the strong augmentation.

Architecture: We generally follow [160] for the architectures of both BYOL-asym and MSF. We use the ResNet50 [178] model as backbone in all our experiments. A projection layer (2 layer MLP) is added

on top of the backbone. The first layer expands the feature channels from 2048 to 4096. It is followed by BatchNorm and ReLU layers. The final linear layer reduces the feature channels from 4096 to 512. The prediction layer architecture is the same as projection layer except its first layer expands the channels from 512 to 4096. After the pre-training step, online encoder's backbone is evaluated by removing the projection and prediction layers.

Training: For both BYOL-asym and MSF, we use the SGD (lr=0.05, momentum=0.9, and weight decay=1e-4) optimizer and train for 200 epochs. Learning rate uses cosine scheduler.

4.2.4.2 Evaluation on ImageNet

Evaluation on full ImageNet. We evaluate the representations of the pre-trained model by training linear and nearest neighbor (NN) classifiers. We use the code provided by [240] for training both classifiers. A single linear layer is trained on top of a frozen backbone. The features from the backbone are normalized to have unit ℓ_2 norm and then scaled and shifted to have zero mean and unit variance for each dimension. The linear layer is trained with SGD (lr=0.01, epochs=40, batch size=256, weight decay=1e-4, and momentum=0.9). Learning rate is multiplied by 0.1 at 15 and 30 epochs. We use standard supervised ImageNet augmentations [11] during training. For nearest neighbor classification, we pre-process the ImageNet training and validation sets with center crop augmentation (size 256) and compute ℓ_2-normalized embeddings by forwarding throught the backbone. We report Top-1 accuracies on ImageNet val set for linear, 1-NN, and 20-NN classifiers in Table 4.8.

Evaluation on smaller ImageNet: Similar to [70, 160, 183, 240], we evaluate the pre-trained models on the task of classification with limited ImageNet labels. The training details are the same as above except the training dataset sizes are reduced to 1% and 10% of the train set of ImageNet [378]. The results are reported in Table 4.10.

4.2.4.3 Evaluation on Transfer Learning

Linear classification: Following the procedure outlined in [70, 160], we evaluate the self-supervised pre-trained models for linear classification task on following datasets: Food101 [41], SUN397 [481], CIFAR10 [245], CIFAR100 [245], Cars [244], Aircraft [301], Flowers [321], Pets [333], Caltech-101 [130] and DTD [90]. The appendix includes more details on the datasets and training. The results are reported in Table 4.9. To verify our implementation, we evaluate the official 1000-epoch BYOL weights provided in [1] and compare with the results from [160] in Table 4.9.

TABLE 4.8. **Left: Evaluation on full ImageNet:** We compare our model on the full ImageNet linear and nearest neighbor benchmarks using ResNet50. We find that given similar computational budget, our models are better than other state-of-the-art methods. Our w/s variation works slightly better than the regular MSF. Interestingly, when using weak augmentations only, our method (MSF w/w) outperforms BYOL and SimCLR with a large margin. We believe this is important in some applications,e g.g. medical domain, where augmentation engineering is not easy. Note that methods with symmetric loss are not directly comparable with the other ones as they need to feed each image twice though each encoder. This results in twice the computation for each mini-batch. One may argue that a non-symmetric BYOL with 200 epochs should be compared with symmetric BYOL with 100 epochs only as they use similar amount of computation. Note that symmetric MoCo v2 with 400 epochs is almost the same as asymmetric MoCo v2 with 800 epochs (71.0 vs. 71.1). Note that the accuracy of Random-init for ResNet50 is much lower than AlexNet (14.1% on conv5 layer from [325]). †: CompRess is not directly comparable as it uses ResNet50 distilled from a larger SSL teacher model (SimCLR-ResNet50x4). ‡: SwAV is not comparable as it uses multiple crops together. **Right: Epochwise t-SNE for MSF:** We visualize the ℓ_2 normalized features for 10 random ImageNet classes at certain epochs of MSF training. We find that over the period of training, semantic clusters are formed in the feature space.

Method	Ref.	Batch Size	Epochs	Sym. Loss 2x FLOPS	Top-1 Linear	NN	20-NN
Supervised	[16]	256	100	-	76.2	71.4	74.8
Random-init	-	-	-	-	5.1	1.5	2.0
SeLa-v2 [502]	[57]	4096	400	✓	67.2	-	-
SimCLR [70]	[70]	4096	1000	✓	69.3	-	-
SwAV [57]	[57]	4096	400	✓	70.1	-	-
DeepCluster-v2 [56]	[57]	4096	400	✓	70.2	-	-
SimSiam [77]	[77]	256	400	✓	70.8	-	-
MoCo v2 [175]	[77]	256	400	✓	71.0	-	-
MoCo v2 [175]	[76]	256	800	✗	71.1	57.3	61.0
CompRess † [240]	[240]	256	1K+130	✗	71.9	63.3	66.8
InvP	[453]	256	800	✗	71.3	-	-
BYOL [160]	[160]	256	300	✓	71.8	-	-
BYOL [160]	[160]	4096	1000	✓	74.3	62.8	66.9
SwAV ‡ [57]	[57]	4096	800	✓	**75.3**	-	-
SimCLR [70]	[77]	4096	200	✓	68.3	-	-
SwAV [57]	[77]	4096	200	✓	69.1	-	-
MoCo v2 [175]	[77]	256	200	✓	69.9	-	-
SimSiam [77]	[77]	256	200	✓	70.0	-	-
BYOL [160]	[77]	4096	200	✓	**70.6**	-	-
MoCo v2 [175]	[76]	256	200	✗	67.5	50.9	54.3
CO2 [468]	[468]	256	200	✗	68.0	-	-
BYOL-asym [160]	-	256	200	✗	69.3	55.0	59.2
ISD [420]	[420]	256	200	✗	69.8	59.2	62.0
MSF	-	256	200	✗	71.4	60.6	64.0
MSF w/s	-	256	200	✗	**72.4**	**62.0**	64.9
MSF w/s (128K)	-	256	200	✗	72.1	**62.0**	**65.2**
SimCLR w/w [70]	[160]	4096	300	✓	40.2	-	-
BYOL w/w [160]	[160]	4096	300	✓	60.1	-	-
MSF w/w	-	256	200	✗	**66.3**	54.6	57.4

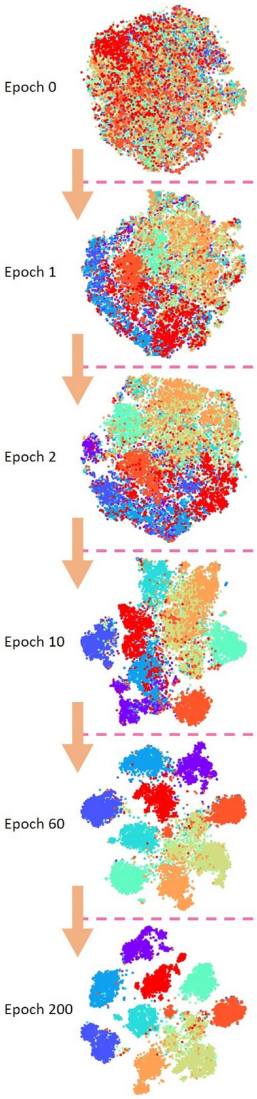


TABLE 4.9. **Linear layer transfer learning evaluation:** We compare various SSL methods on transfer tasks by training linear layers. Under similar computational budgets, we show that our models are consistently better or on par with other state-of-the-art methods. Only a single linear layer is trained on top of features. No train time augmentations are used. "rep." means we have reproduced the results using our evaluation framework for better comparison.

Method	Ref.	Epochs	Food 101	CIFAR 10	CIFAR 100	SUN 397	Cars 196	Aircraft	DTD	Pets	Caltech 101	Flowers 102	Mean
Supervised	[160]		72.3	93.6	78.3	61.9	66.7	61.0	74.9	91.5	94.5	94.7	78.9
SimCLR [70]	[160]	1000	72.8	90.5	74.4	60.6	49.3	49.8	75.7	84.6	89.3	92.6	74.0
MoCo v2 [76]	-	800	72.5	92.2	74.6	59.6	50.5	53.2	74.4	84.6	90.0	90.5	74.2
BYOL [160]	[160]	1000	75.3	91.3	**78.4**	**62.2**	**67.8**	60.6	75.5	**90.4**	**94.2**	**96.1**	**79.2**
BYOL [160]	rep.	1000	**75.4**	**92.7**	78.1	62.1	67.1	**62.0**	**76.8**	89.8	92.2	95.5	**79.2**
BYOL-asym [160]	-	200	70.2	91.5	74.2	59.0	54.0	52.1	**73.4**	86.2	90.4	92.1	74.3
MoCo v2 [76]	-	200	70.4	91.0	73.5	57.5	47.7	51.2	73.9	81.3	88.7	91.1	72.6
MSF	-	200	70.7	92.0	76.1	59.0	**60.9**	53.5	72.1	89.2	92.1	92.4	75.8
MSF-w/s	-	200	71.2	92.6	**76.3**	59.2	55.6	53.7	73.2	88.7	**92.7**	92.0	75.5
MSF-w/s (128K)	-	200	**72.3**	**92.7**	**76.3**	**60.2**	59.4	**56.3**	71.7	**89.8**	90.9	**93.7**	**76.3**

TABLE 4.10. **Evaluation on small labeled ImageNet:** We compare our model on the ImageNet 1% and 10% linear evaluation benchmarks for ResNet50. The column "Fine-tuned" refers to whether the full network was fine-tuned or a single linear layer was trained. Given similar computational budgets, both of our models are better than other state-of-the-art methods. We evaluate BYOL and MoCo v2 with our evaluation framework and interestingly, realize that BYOL performs better in linear evaluation compared to finetuning the whole network on the 1% split. We report these numbers for fair comparison. * is ResNet50 compressed from SimCLR-R50x4. † uses a different augmentation strategy than others. ‡ is our evaluation with the official weights [1].

Method	Fine-tuned	Epochs	Top-1 1%	Top-1 10%	Top-5 1%	Top-5 10%
Supervised	✓		25.4	56.4	48.4	80.4
PIRL [313]	✓	800	-	-	57.2	83.8
CO2 [468]	✓	200	-	-	71.0	85.7
SimCLR [70]	✓	1000	48.3	65.6	75.5	87.8
InvP [453]	✓	800	-	-	78.2	88.7
BYOL [160]	✓	1000	53.2	68.8	78.4	89.0
SwAV† [57]	✓	800	**53.9**	**70.2**	**78.5**	**89.9**
MoCo v2 [76]	✗	800	51.5	63.6	77.6	86.1
BYOL‡ [160]	✗	1000	55.7	**68.6**	80.0	**88.6**
CompRess* [240]	✗	1K+130	**59.7**	67.0	**82.3**	87.5
MoCo v2 [76]	✗	200	43.6	58.4	71.2	82.9
BYOL-asym	✗	200	47.9	61.3	74.6	84.7
ISD [420]	✗	200	53.4	63.0	78.8	85.9
MSF	✗	200	53.5	65.2	78.1	86.4
MSF w/s	✗	200	**55.5**	**66.5**	**79.9**	**87.6**

TABLE 4.11. **Transfer learning to PASCAL VOC object detection:** We compare our models on the transfer task of object detection. We find that given a similar computational budget, our method is better than BYOL. The models are trained on the VOC `trainval07+12` set and evaluated on the `test07` set. We report average over 5 runs.

Method	Ref.	Epochs	AP_{50}	AP	AP_{75}
Sup. IN	[77]	-	81.3	53.5	58.8
Scratch	[77]	-	60.2	33.8	33.1
Symmetric loss. 2x FLOPS					
SimCLR	[77]	200	81.8	55.5	61.4
MoCo v2	[77]	200	82.3	57.0	63.3
BYOL	[77]	200	81.4	55.3	61.1
SwAV	[77]	200	81.5	55.4	61.4
SimSiam	[77]	200	82.4	57.0	63.7
Asymmetric loss.					
MoCo v2	[76]	800	82.5	57.4	64.0
InvP	[453]	800	81.8	56.2	61.5
MoCo v2	[76]	200	82.4	57.0	63.6
CO2	[468]	200	**82.7**	**57.2**	**64.1**
BYOL-asym	-	200	81.9	56.8	63.5
MSF	-	200	82.2	56.7	63.4
MSF w/s	-	200	82.2	56.6	63.1

TABLE 4.12. **Effect of k in top-k:** Our study shows that MSF is not very sensitive to k. While k = 10 performs the best, we report the main results for k = 5. Note that setting k = 1 makes our method identical to BYOL-asym.

	k=1	k=2	k=5	k=10	k=20	k=50
NN	55.8	61.0	62.0	62.5	62.0	61.5
20-NN	59.1	64.2	64.9	65.7	65.4	64.9

TABLE 4.13. **Additional computational cost of finding NNs:** Forwarding through each ResNet50 encoder needs 4.14 GFLOPS, so finding NNs adds a small cost. Note that any method like MoCo that uses a memory bank needs this additional cost.

Mem. Size	kNN Time	kNN GFLOPS	NN	20-NN	Transfer Mean
1M	6.78%	1.05	**62.0**	64.9	75.5
128K	0.72%	0.13	**62.0**	**65.2**	**76.3**

Object detection: Following the procedure outlined in [175], we use Faster-RCNN [370] for the task of object detection on PASCAL-VOC [121]. We use the code provided at [12] with default parameters. All the weights are finetuned on the `trainval07+12` set and evaluated on the `test07` set. We report an average over 5 runs in Table 4.11.

4.2.4.4 Ablation study

Here, we study the effect of MSF hyperparameters and design choices like augmentation strategies, top-k, and memory bank size. We use ResNet50 and train it with ImageNet. In all experiments, we use the default MSF w/s variant and only vary the parameter of interest.

Same view of an instance to both encoders: One may argue that the mean-shift grouping and using different views of the same instance for different encoders are orthogonal ideas, and mean-shift alone might work. We did an experiment by feeding the same augmented view to both encoders ($T1 = T2$) and realized that the model does not learn. It collapses in the first epoch. Hence, we believe using different views is still an important inductive bias.

Effect of k in top-k: This section shows the effect of sampling different top-k nearest neighbors. We use k values from set $\{2, 5, 10, 20, 50\}$. We use k = 5 for main experiments, but k = 10 improves NN by 0.5 point. Note that setting k = 1, makes MSF identical to BYOL. Results are in the Table 4.12. Additionally, we plot the purity for each experiment in Figure 4.8. Purity for a single query is the percentage of the samples 2 to k in the top-k nearest neighbors (excluding u itself) which have the same class as the query. Final purity is calculated by averaging the purities of all samples. One may study the effect of increasing k gradually during iterations as a future extension.

Size of memory bank: CompRess [240] shows that a large memory bank is important to accurately capture the neighborhood of a random sample in the embedding space. Thus, we vary the size of the memory bank from 256 to 1M to evaluate if larger memory bank can help with more accurate nearest

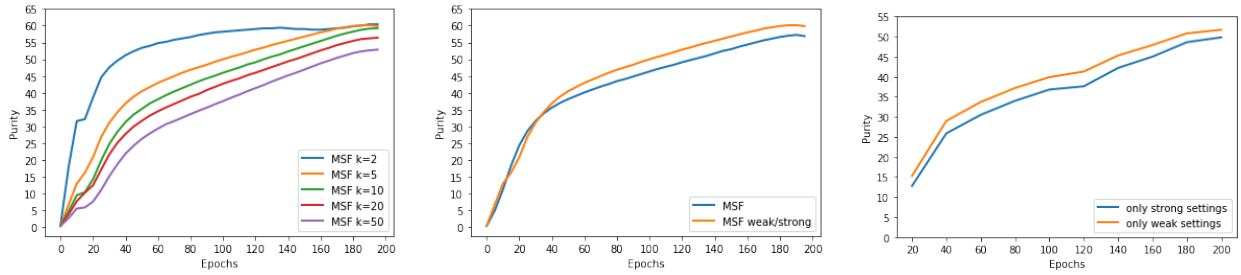


FIGURE 4.8. **Comparison of purity:** Ideally, we want all k nearest neighbors to be from the same category as the input image, so we calculate the percentage of correct neighbors for each input, average it over all images, and call it purity. We exclude the first NN which is identical to u, so the accuracy is over $k-1$ neighbors. We find this metric to be very handy in evaluating the model at the training time since it comes almost for free. **(left)** shows the purity for different k values with respect to epoch number while **(middle)** compares the purity of "w/s" variation with the regular "s/s" variation for $k = 5$. Purity is higher for "w/s" variation which is consistent with our intuition. In **(right)**, at each epoch of our MSF w/s (top-k = 10) model, we calculate purity for the target encoder using either only weak (orange) or only strong (blue) augmentations. We see that strong setting has lower purity. This suggests that the stronger augmentation makes the nearest neighbors more noisy. This is aligned with our intuition for using weak augmentation for the target model in w/s variation.

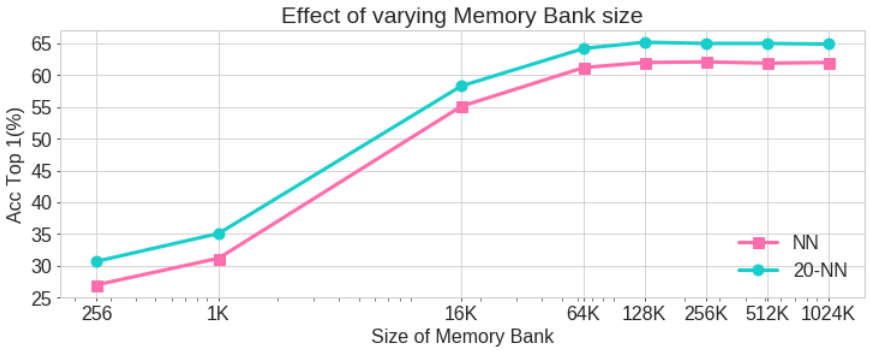


FIGURE 4.9. **Memory Bank Size:** On ImageNet, we do not see an improvement in increasing the memory bank size beyond 128K which needs only 0.5GB of GPU memory.

neighbors. Results are in Figure 4.9. Although our main experiment uses 1M sized memory bank, we find that 128K works equally well. Note that the size of memory bank also depends on the training dataset size.

Comparison of different augmentation strategies: Table 4.14 shows results for BYOL and MSF with different augmentation strategies. Comparing "s/s" variants with "w/s", we find that BYOL receives a very small boost from the "w/s" variant while MSF improves consistently by ≈ 1 point on all three benchmarks. We believe this is due to better purity of the nearest neighbors while training (also shown in Fig. 4.8 (right)). Further, we observe that MSF w/w is significantly better as compared to BYOL w/w. This can be attributed to the nearest neighbors serving as a proxy for strong augmentation.

TABLE 4.14. **Comparison augmentations strategies:** In s/s, both views are strongly augmented while in w/w they are weakly augmented. w/s refers to weak augmentation for target view and strong augmentation for the online view. w/s improves our method more compared to BYOL. This may be due to more pure nearest neighbors. In w/w setting, MSF is significantly better than BYOL as the nearest neighbors can be a good substitute for strong augmentation. †: uses 4096 batch size, 300 epochs, and symmetric loss.

Method	Aug.	Top-1	NN	20-NN
BYOL-asym	s/s	69.3	55.0	59.2
BYOL-asym	w/s	69.5	55.8	59.1
BYOL † [160]	w/w	60.1	-	-
MSF	s/s	71.4	60.6	64.0
MSF	w/s	72.4	62.0	64.9
MSF	w/w	66.3	54.6	57.4

4.2.5 Conclusion

We introduce a simple but effective SSL method based on grouping similar images together in an online fashion. We simply shift the embedding of an image towards the mean of its nearest neighbors. MSF with $k = 1$ is identical to BYOL so MSF can be seen as a generalized form of BYOL. Our extensive experiments show that MSF performs better or on-par compared to state-of-the-art SSL methods on various tasks including ImageNet linear evaluation.

4.3 Constrained Mean Shift for Representation Learning

We are interested in representation learning in self-supervised, supervised, and semi-supervised settings. Some recent self-supervised learning methods like mean-shift (MSF) cluster images by pulling the embedding of a query image to be closer to its nearest neighbors (NNs). Since most NNs are close to the query by design, the averaging may not affect the embedding of the query much. On the other hand, far away NNs may not be semantically related to the query. We generalize the mean-shift idea by constraining the search space of NNs using another source of knowledge so that NNs are far from the query while still being semantically related. We show that our method (1) outperforms MSF in SSL setting when the constraint utilizes a different augmentation of an image from the previous epoch, and (2) outperforms PAWS in semi-supervised setting with less training resources when the constraint ensures that the NNs have the same pseudo-label as the query. Our code is available here: https://github.com/UCDvision/CMSF

4.3.1 Introduction

Recently, we have seen great progress in self-supervised learning (SSL) methods that learn rich representations from unlabeled data. Such methods are important since they do not rely on manual annotation of data, which can be costly, biased, or ambiguous. Hence, SSL representations may perform better than supervised ones in transferring to downstream visual recognition tasks.

Most recent SSL methods, *e.g.*, MoCo [175] and BYOL [160], pull the embedding of a query image to be closer to its own augmentation compared to some other random images. Follow-up works have focused on improving the positive pairs through generating better augmentations [256, 367, 426] and the negative set by increasing the set size [175] or mining effective samples [207, 221, 454], but have largely ignored possibility of utilizing additional positive images. More recently, [27, 119, 242] expand the positive set using nearest neighbors. Inspired by classic mean-shift algorithm, MSF [242] generalizes BYOL to group similar images together. MSF pulls a query image to be close to not only its augmentation, but also the top-k nearest neighbors (NNs) of its augmentation.

We argue that the top-k neighbors are close to the query image by construction, and thus may not provide a strong supervision signal. We are interested in choosing far away (non-top) neighbors that are still semantically related to the query image. This cannot be trivially achieved by increasing the number of NNs since the *purity* of retrieved neighbors decreases with increasing k (See Fig. 4.13 and Fig. 4.14). Purity is defined as the percentage of the NNs belonging to the same category as the query image.

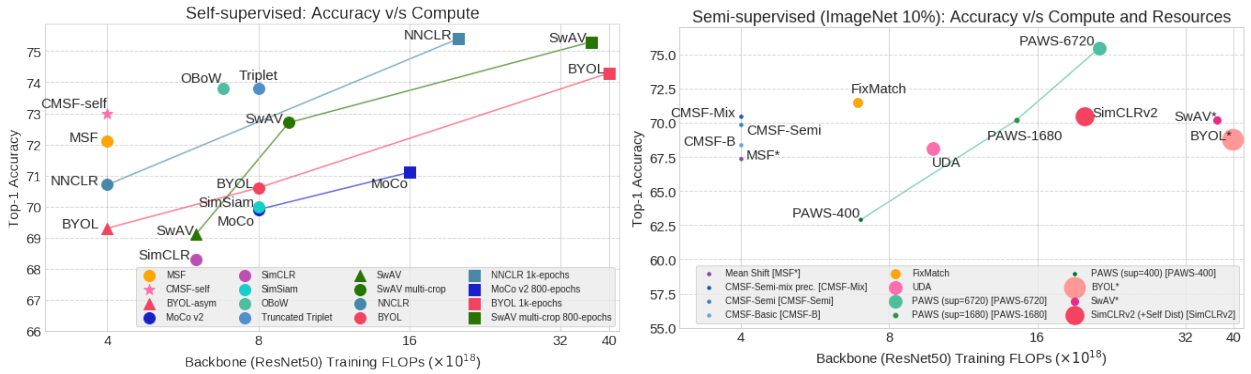


FIGURE 4.10. **Accuracy vs. training compute on ImageNet with ResNet50:** We report the total training FLOPs for forward and backward passes through the CNN backbone. **(Left) Self-supervised:** All methods are for 200 epochs. $\text{CMSF}_{\text{self}}$ achieves competitive accuracy with considerably lower compute. **(Right) Semi-supervised:** Circle radius is proportional to the number of GPUs/TPUs used. The results are on ImageNet with 10% labels. In addition to being compute efficient, CMSF is trained with an order of magnitude lower resources, making it more practical and accessible. * methods use self-supervised pre-training and finetuning on the labeled set.

We generalize MSF [242] method by simply limiting the NN search to a smaller subset that we believe is reasonably far from the query but still semantically related to it. We define this constraint to be (1) the nearest neighbors of another augmentation of the query in SSL setting and (2) images sharing the same label or pseudo-label as the query in supervised and semi-supervised settings. While we aim to obtain distant samples of the same category, note that we group only a few neighbors (k in our method) from the constrained subset instead of grouping the whole subset together. This is in contrast to cross-entropy supervised learning, where we pull all images of a category to form a cluster or be on the same side of a hyper-plane. Our method can benefit from this relaxation by preserving the latent structure of the categories and also being robust to noisy labels.

Our experiments show that the method outperforms the various baselines in all three settings with same or less amount of computation in training (refer Fig. 4.10). It outperforms MSF [242] in SSL, cross-entropy in supervised (with clean or noisy labels), and PAWS [26] in semi-supervised settings. Our main novelty is in developing a simple but effective method for searching for far away but semantically related NNs and in generalizing it to work across the board from self-supervised to semi-supervised and fully supervised settings. To summarize,

(1) We propose constrained mean-shift (CMSF), a generalization of MSF [242], to utilize additional sources of knowledge to constrain the NN search space.

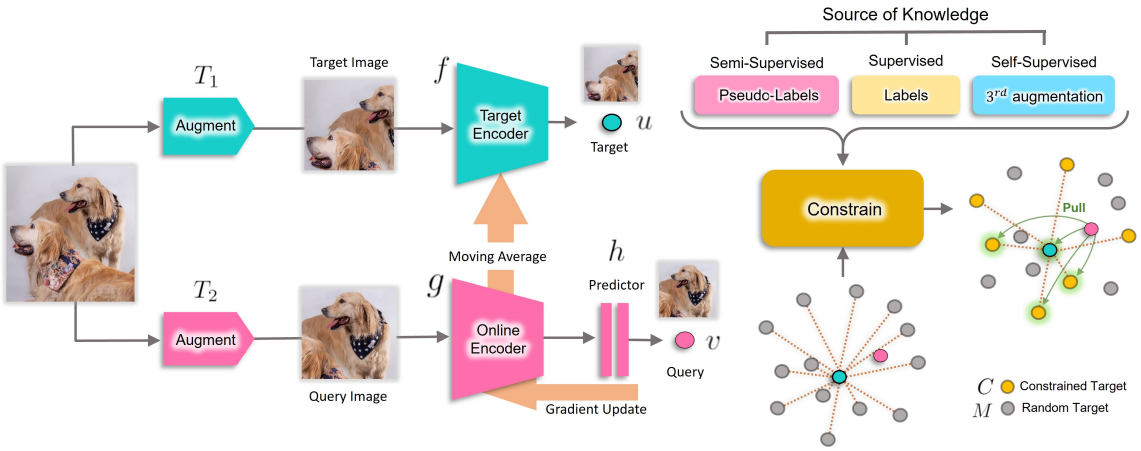


FIGURE 4.11. **Our method (CMSF):** We augment an image twice and pass them through online and target encoders followed by ℓ_2 normalization to get u and v. Mean-shift [242] encourages v to be close to both u and its nearest neighbors (NN). To make NNs diverse, we constrain the NN search space based on additional knowledge in the form of NNs of the previous augmentation in self-supervised setting or the labels or pseudo-labels in semi or fully supervised settings. These constraints encourages the query to be pulled towards semantically related NNs that are farther away from the target embedding. See Fig 4.12 for constructing the constrained set.

(2) We develop methods to select the constraint set in self-, semi- and fully supervised settings. The retrieved samples are empirically shown to be far away in the embedding space but semantically related to the query image, providing a stronger training signal compared to MSF.

(3) CMSF achieves non-trivial gains in performance over self-supervised MSF and a direct extension of MSF to semi-supervised version. CMSF outperforms SOTA methods with comparable compute in self- and semi-supervised settings.

4.3.2 Related Work

Self-supervised learning (SSL): Earlier works on SSL focused on solving a pretext task that does not require additional labeling. Examples of pretext tasks include colorization [526], jigsaw puzzle [323], counting [324], and rotation prediction [149]. Another class of SSL methods is based on instance discrimination [114]. The idea is to classify each image as its own class. Some methods adopt the idea of contrastive learning for instance discrimination [56,57,59,70,175]. BYOL [160] proposes a non-contrastive approach by removing the negative set and simply regressing one view of an image from another.

Several recent works aim to find a larger positive sample set to improve learning. In LA [545], samples are clustered using k-means and samples within a cluster are brought closer together compared to cross-cluster samples. MSF [242] and MYOW [27] generalize BYOL by regressing target view and its NNs.

NNCLR [119] extends SimCLR to use NNs as positives. CLD [465] integrates grouping using instance-group discrimination. Affinity diffusion [205] uses strongly connected nodes in a graph constructed using embeddings to find positive samples. Unlike these methods, we focus on grouping together far away neighbors that are semantically similar. We show quantitatively and qualitatively the diversity and purity of retrieved neighbors and improved performance over MSF. We generalize the idea in MSF [242] to use an additional source of knowledge to constrain the NN search space for the target view. CoCLR [169] and Cl-InfoNCE [443] also use additional information sources in the form of additional modality and auxiliary labels respectively to improve performance. However, we focus on self- and semi-supervised classification settings and design methods to obtain and use the additional information as a constraint in NN search space.

Supervised learning: A drawback of Cross-entropy is its lack of robustness to noisy labels [408, 532]. [315, 413, 437, 491] address the issue of hard labeling, *e.g.*, (one-hot labels) with label smoothing, [32, 142, 188] replace hard labels with prediction of pre-trained teacher, and [512, 520] propose an augmentation strategy to train on combination of instances and their labels. Another line of work [151, 384] is to learn representations with good kNN performance. SupCon [230] and [479] improve upon [151] by changing the distance to inner product on ℓ_2 normalized embeddings. We include the supervised setting to better understand the effect of using constrained NNs, particularly in the noisy label setting.

Semi-supervised learning: Several methods combine self-supervised and supervised learning to form semi-supervised methods. S4L [518] uses rotation prediction based loss on the unlabeled set along with cross-entropy loss on the labeled set. Similarly, SuNCEt [25] combines SimCLR [70] and SwAV [57] methods with supervised contrastive loss. Pseudo-labeling is frequently used in semi-supervised learning. In Pseudo-Label [254], the network is trained with cross-entropy loss using supervised data on the labeled examples and pseudo-labels on the unlabeled ones. In SimCLR-v2 [72], a teacher network is pre-trained using SimCLR [70] and fine-tuned with supervised labels. The teacher is then distilled to a student network using pseudo-labels on the unlabeled set. FixMatch [403] uses pseudo-labels obtained using a weakly augmented image to train a strongly augmented version of the same image. UDA [484] leverages strong data augmentation techniques in enforcing this consistency in pseudo-labels across augmentations. MPL [344] optimizes a student network using pseudo-labels from a teacher network, while the teacher is optimized to maximize the student's performance on the labeled set. PAWS [26] uses consistency based loss on soft pseudo-labels obtained in a non-parametric manner. Our method too uses pseudo-labels to train the unlabeled samples. However, we use the labels as a constraint in MSF [242] and do not directly optimize samples using cross-entropy loss.

Metric learning: The goal of metric learning is to train a representation that puts two instances close in the embedding space if they are semantically close. Two important methods in metric learning are: triplet loss [85, 389, 470] and contrastive loss [42, 402]. Metric learning methods perform well on tasks like image retrieval [475] and few-shot learning [401, 451]. Prototypical networks [401] is similar to a contrastive version of our method with top-*all*.

4.3.3 Method

Similar to MSF [242], given a query image, we are interested in pulling its embedding closer to the mean of the embeddings of its nearest neighbors (NNs). However, since top NNs are close to the target itself, they may not provide a strong supervision signal. On the other hand, far away (non-top) NNs may not be semantically similar to the target image. Hence, we constrain the NN search space to include mostly far away points with high purity. The purity is defined as the percentage of the selected NNs being from the same ground truth category as the query image. We use different constraint selection techniques to analyze our method in supervised, self- and semi-supervised settings.

Following MSF and BYOL, we use two embedding networks: a target encoder $f(.)$ with parameters θ_f and an online encoder $g(.)$ with parameters θ_g. The online encoder is directly updated using backpropagation while the target encoder is updated as a slowly moving average of the online encoder: $\theta_f \leftarrow m\theta_f + (1-m)\theta_g$ where m is close to 1. We add a predictor head $h(.)$ [160] to the end of the online encoder so that pulling the embeddings together encourages one embedding to be predictable by the other one and not necessarily encouraging the two embeddings to be equal. In the experiments, we use a two-layer MLP for $h(.)$.

Given a query image x_i, we augment it twice with transformations $T_1(.)$ and $T_2(.)$, feed them to the two encoders, and normalize them with their ℓ_2 norm to get $u_i = \frac{f(T_1(x_i))}{||f(T_1(x_i))||_2}$ and $v_i = \frac{h(g(T_2(x_i)))}{||h(g(T_2(x_i)))||_2}$. We add u_i to the memory bank M and remove the oldest entries to maintain a fixed size M. We select the constraint set C_i as a subset of M. Constraint set selection is explained in detail in Sections 4.3.3.1, 4.3.3.2, and 4.3.3.3. We then find the set S_i of top-k nearest neighbors of u_i in C_i including u_i itself. Finally, we update $g(.)$ by minimizing:

$$L = \sum_{i=1}^{n} \frac{1}{|S_i|} \sum_{z \in S_i} v_i^T z$$

where n is the size of mini-batch and $|S_i|$ is the size of set S_i, *e.g.*, k in top-k. Finally, we update $f(.)$ with the momentum update. In the top-*all* variation of our method, number of neighbors k is set equal to the size of C_i, *i.e.*, $S_i = C_i$. Note that since u_i itself is included in the nearest neighbor search, the method will be identical to BYOL [160] when $k = 1$ and to self-supervised mean-shift [242] when the constraint is fully

relaxed ($C_i = M$). Our method covers a larger spectrum of algorithms by defining the constrained set. Below we discuss the selection of constrained set in various settings.

4.3.3.1 Self-Supervised Setting

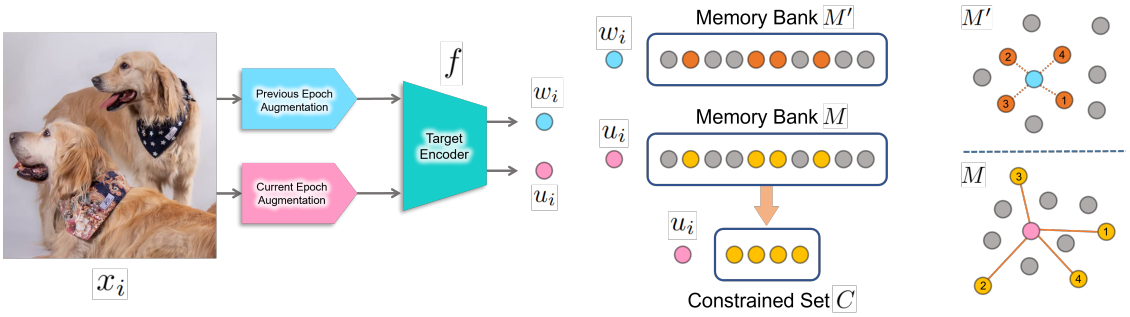


FIGURE 4.12. **CMSF$_{\text{self}}$:** The indices of the NNs of the previous epoch's memory bank M' are used to construct the constrained set C from the current memory bank M.

In addition to M, we maintain a second memory bank M' that is exactly the same as M but contains features from a different (3^{rd}) augmentation of the image x_i fed through target encoder $f(.)$. We assume $w_i \in M'$ and $u_i \in M$ are two embeddings corresponding to the same image x_i. Then, we find NNs of w_i in M' and use their indices to construct the search space C_i from M (See Fig. 4.12). Note that although the NNs of w_i in M' are already close to each other, their corresponding elements in M may not be close to each other since M contains different augmentations u_i of the same images. As a result, C_i will maintain good purity while containing distant NNs (refer to Table 4.15-Right and Fig. 4.14).

Since it is expensive to embed a 3rd augmentation of each image, we embed only two augmentations as in MSF and BYOL and cache the embeddings from the previous epoch, keeping the most recent embedding for each image. The cached embedding will be still valid after one epoch since the target encoder is updated slowly using the momentum update rule (similar to MoCo). Since cache size is equal to the dataset size, we store it in the CPU memory and maintain the auxiliary memory bank M' by loading the corresponding part of it to the GPU memory for each mini-batch. Caching of features is not essential for CMSF to work and is only used to reduce computational cost. We performed experiments with an actual 3rd augmentation instead and found the results to be similar to our method except that it was nearly 30% slower due to forwarding an additional augmentation. Table 4.15-Right shows that in the intermediate stages of learning, the top elements of C_i are spread apart in M with higher median ranks, and get closer to the top elements of M as the learning progresses. Note that we use w_i instead of u_i in finding the NNs in M' since both w_i and M' use an older target model, so are more comparable.

Since CMSF adds farther NNs only for stronger supervision, we additionally employ MSF loss calculated on the unconstrained M. Then, in the self-supervised setting, the total loss is an equally weighted sum of MSF and CMSF losses.

Our method can be extended to cross-modal self-supervised setting where the constraint can use NNs in a different modality rather than the 3rd augmentation of the same modality. We report the details and some preliminary experiments on this setting in the supplementary.

4.3.3.2 Supervised Setting

While supervised setting is not our primary novelty or motivation, we study it to provide more insights into our constrained mean-shift framework. With access to the labels of each image, we can simply construct C_i as the subset of M that shares the same label as the query x_i. This guarantees 100% purity for NNs.

Note that most supervised methods, including cross-entropy loss, try to group all examples of a category together on the same side of a hyper-plane while remaining categories are on the other side. However, our method pulls the target to be close to only those examples of the same category that are already close to the target. This results in a supervised algorithm that may keep the latent structure of each category which can be useful for pre-training on coarse-grained labels. Moreover, as shown in the experiments (Fig. 4.16), our method is more robust to label noise since most mis-labeled images will be far from the target embedding, thus ignored in learning. This motivates applying our method to semi-supervised setting where the limited supervision provides noisy labels.

4.3.3.3 Semi-Supervised Setting

In this setting, we assume access to a dataset with a small labeled and a large unlabeled subset. We train a simple classifier using the current embeddings of the labeled data and use the classifier to pseudo-label the unlabeled data. Then, similar to the supervised setting, we construct C_i to be the elements of M that share the pseudo-label with the target embedding. Again, this method increases the diversity of C_i while maintaining high purity. To keep the purity high, we enforce the constraint only when the pseudo-label is very confident (the probability is above a threshold.) For the samples with non-confident pseudo-label, we relax the constraint resulting in regular MSF loss (*i.e.,* $C_i = M$.) Moreover to reduce the computational overhead of pseudo-labeling, we cache the embeddings of labeled examples throughout the epoch and train a 2-layer MLP classifier using the frozen cached features and their groundtruth labels in the middle and end of each epoch.

4.3.4 Experiments

Implementation details: We use PyTorch for all our experiments. Unless specified, we use the same hyper-parameter values in self-, semi- and fully supervised settings. All models are trained on ImageNet-1k (IN-1k) for 200 epochs with ResNet-50 [178] backbone and SGD optimizer (learning rate=0.05, batch size=256, momentum=0.9, and weight decay=1e-4) with cosine scheduling for learning rate. The momentum value of CMSF for the moving average key encoder is 0.99. The 2-layer MLP architecture for $\text{CMSF}_{\text{semi}}$ is as follows: (linear (2048x4096), batch norm, ReLU, linear (4096x512)). The default memory bank size is 128k. Top-k is set to 10 in the semi- and fully supervised settings and 5 in the self-supervised setting. Additional details are provided in the supplementary. Our main CMSF experiment with 200 epochs takes nearly 6 days on four NVIDIA-2080TI GPUs. The overhead in training time due to NN search is negligible compared to the forward and backward passes through the network (that is also done in BYOL): the increase in time is 0.7% for MSF [242] and 2.1% for $\text{CMSF}_{\text{self}}$.

Recent SSL methods are usually computationally expensive leading to worse environmental impact and exclusion of smaller research labs. While our experiments are more efficient and accessible than most SOTA methods, *e.g.*, PAWS, we limit our training length to 200 epochs due to resource constraints. We do not empirically verify whether the improvements observed over SOTA approaches at lower epochs (200) are persistent with longer training (*e.g.*, 800 or 1000 epochs).

Evaluation: We evaluate the pre-trained models using linear evaluation (*Linear IN-1k*) in both ImageNet classification and transfer settings. The model backbone parameters are fixed and a single linear layer is trained atop them following the setting in CompRess [240]. Additionally, we report k-nearest neighbor ($k = 1, 20$) evaluation for the SSL setting as in [240]. The transfer performance is evaluated on the following datasets: Food101 [41], SUN397 [481], CIFAR10 [245], CIFAR100 [245], Cars196 [244], Aircraft [301], Flowers (Flwrs102) [321], Pets [333], Caltech-101 (Calt101) [130], and DTD [90] (additional details in supplementary material.)

4.3.4.1 Self-Supervised Learning ($\text{CMSF}_{\text{self}}$)

To reduce the GPU memory footprint, we cache the previous augmentation embedding of each sample in the dataset in the CPU. The cached features corresponding to the current mini-batch are retrieved from CPU memory to maintain memory bank M' with previous augmentations. This cache is updated using the oldest features in M that we remove from M after each iteration.

FIGURE 4.13. **Nearest neighbor selection on constrained memory bank:** First row shows top-5 NNs of target in constrained set C and their corresponding rank in the unconstrained memory bank M obtained using an intermediate checkpoint (epoch 100). While they are not the closest samples to the target (higher rank index), they are semantically similar to the target. This shows that the constraint can capture far away samples with similar semantic as the target. The second row depicts images from memory bank with one rank lower than the corresponding image in the first row. These images contain incorrect category retrievals. Distant neighbors cannot be trivially obtained by increasing the number of NNs. Examples are chosen randomly.

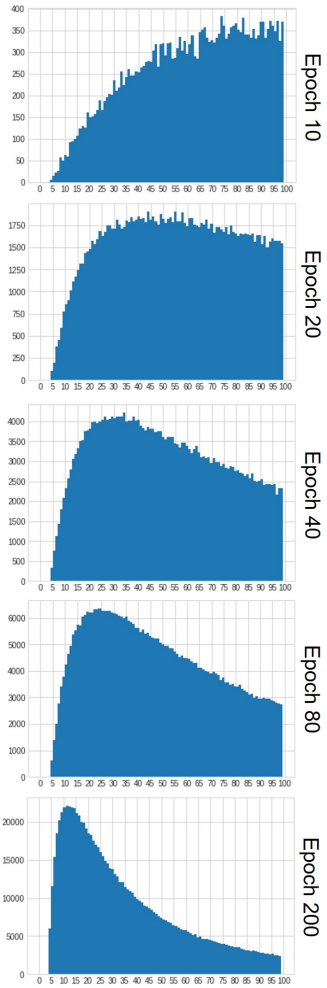
Results on ImageNet: Results of $\text{CMSF}_{\text{self}}$ are shown in Table 4.15. $\text{CMSF}_{\text{self}}$ outperforms MSF baseline with a larger memory bank, which we believe is due to pulling together far yet semantically similar samples (Fig. 4.13). We use MSF with $2x$ larger memory bank for fair comparison. $\text{CMSF}_{\text{self}}$ also achieves state-of-the-art performance on both NN and Linear metrics when compared with approaches with similar computational budget. We compare our method to other state-of-the-art approaches with 200 epochs of training in Fig. 4.10. We observe a good trade-off in terms of accuracy and compute for $\text{CMSF}_{\text{self}}$. Note that we train without symmetrical loss and multi-crop strategy which are known to generally improve performance [57] at the cost of significant increase in compute. **Evaluation on ImageNet subsets:** Following [70, 183], we evaluate the pre-trained models on the ImageNet classification task with limited labels. We report results with 1% and 10% labeled subsets of ImageNet (Table 4.15). $\text{CMSF}_{\text{self}}$ outperforms MSF on top-1 accuracy in both 1% and 10% settings and is comparable to existing approaches that require significantly higher training time.

Transfer learning: We follow the procedure in [70, 160] for transfer evaluation (refer to Table 4.16). Hyper-parameters for each dataset are tuned independently based on the validation set accuracy and final accuracy is reported on the held-out test set (more details in supplementary). $\text{CMSF}_{\text{self}}$ achieves SOTA average performance among methods trained for 200 epochs.

Purity of constrained samples: In $\text{CMSF}_{\text{self}}$, we depend on information from previous augmentations to constrain NN search in the current memory bank. Our goal is to improve learning by using distant samples with a good purity. We observe that the top-k samples from constrained memory bank C have higher rank in M, so are far neighbors of the target (see Table 4.15-Right and Fig. 4.14). Also, as shown in Fig. 4.14, those samples maintain almost the same purity as the top-k samples from unconstrained memory bank M. As a result, C maintains good purity while being diverse.

TABLE 4.15. **Left: Evaluation on full ImageNet:** We compare our model with other SOTA methods in Linear (Top-1 Linear) and Nearest Neighbor (1-NN,20-NN) evaluation. We use a memory bank of size 128K for CMSF and provide comparison with both 256K and 1M memory bank versions of MSF. Since $\text{CMSF}_{\text{self}}$ uses NNs from two memory banks, it is comparable to MSF (256K) in memory and computation overhead. Both single crop and multi-crop versions of our method outperform other SOTA methods, including MSF, with similar compute. **Right: Histogram of constrained sample ranks:** We consider the 5^{th} NN in the constrained set C and obtain its rank in the unconstrained memory bank M. The histogram of these ranks are shown up to rank 100 for different train stages of $\text{CMSF}_{\text{self}}$. Also, the median of these ranks are shown in Figure 4.14. A large number of distant neighbors are included in the constrained set in the early stages of training while there is a higher overlap between constrained and unconstrained NN sets towards the end of training.

Method	Ref.	Batch Size	Epochs	Sym. Loss 2x FLOPS	Multi-Crop Training	Top-1 Linear	NN	20-NN
Supervised	[16]	256	100	-	-	76.2	71.4	74.8
Random-init	-	-	-	-	-	5.1	1.5	2.0
SeLa-v2 [502]	[57]	4096	400	✓	✗	67.2	-	-
SimCLR [70]	[70]	4096	1000	✓	✗	69.3	-	-
SwAV [57]	[57]	4096	400	✓	✗	70.1	-	-
DeepCluster-v2 [56]	[57]	4096	400	✓	✗	70.2	-	-
SimSiam [77]	[77]	256	400	✓	✗	70.8	-	-
MoCo v2 [175]	[76]	256	800	✗	✗	71.1	57.3	61.0
CompRess [240]	[240]	256	1K+130	✗	✗	71.9	63.3	66.8
InvP	[453]	256	800	✗	✗	71.3	-	-
BYOL [160]	[160]	4096	1000	✓	✗	74.3	62.8	66.9
SwAV [57]	[57]	4096	800	✓	✓	75.3	-	-
NNCLR	[119]	4096	1000	✗	✗	**75.4**	-	-
SimCLR [70]	[77]	4096	200	✓	✗	68.3	-	-
SwAV [57]	[77]	4096	200	✓	✗	69.1	-	-
MoCo v2 [175]	[77]	256	200	✓	✗	69.9	-	-
SimSiam [77]	[77]	256	200	✓	✗	70.0	-	-
NNCLR [119]	[119]	4096	200	✗	✗	70.7	-	-
BYOL [160]	[77]	4096	200	✓	✗	70.6	-	-
SwAV [57]	[77]	256	200	✓	✓	72.7	-	-
Truncated Triplet [454]	[454]	832	200	✓	✗	**73.8**	-	-
OBoW [148]	[148]	256	200	✗	✓	**73.8**	-	-
MoCo v2 [175]	[76]	256	200	✗	✗	67.5	50.9	54.3
CO2 [468]	[468]	256	200	✗	✗	68.0	-	-
BYOL-asym [160]	[242]	256	200	✗	✗	69.3	55.0	59.2
ISD [420]	[420]	256	200	✗	✗	69.8	59.2	62.0
MSF (1M) [242]	[242]	256	200	✗	✗	72.4	62.0	64.9
MSF (256K) [242]	[242]	256	200	✗	✗	72.2	62.1	65.1
$\text{CMSF}_{\text{self}}$ (128K)	-	256	200	✗	✗	**73.0**	**63.2**	**66.4**



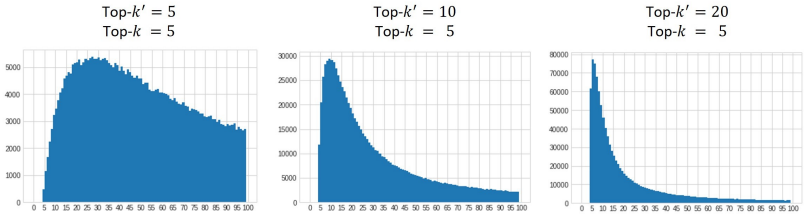
Effect of k': In $\text{CMSF}_{\text{self}}$, we first calculate top-k' samples (the first k' NNs of the target) from the secondary memory bank M'. We then use those indices to constrain NN search space in the primary memory bank M and select top-k for optimization. We varied the value of k' in $\text{CMSF}_{\text{self}}$ to explore its effect, keeping k fixed to 5. We observe that increasing k' (relaxing the constraint) will decrease the accuracy of the model. As observed in Table 4.17-right, the overlap between constrained and unconstrained NN set increases with increasing value of k'. Note that in a case where $k' = \infty$, $\text{CMSF}_{\text{self}}$ will be identical to the MSF baseline.

TABLE 4.16. **Transfer learning evaluation:** Our supervised CMSF model at just 200 epochs outperforms all supervised baselines on transfer learning evaluation. Our SSL model outperforms MSF, the comparable state-of-the-art approach, by 1.2 points on average over 10 datasets. We get the results for MoCo v2, MSF, and BYOL-asym from [242], SimCLR and Xent (1000 epoch) from [70], and BYOL from [160].

Method	Epoch	Food 101	CIFAR 10	CIFAR 100	SUN 397	Cars 196	Air-craft	DTD	Pets	Calt. 101	Flwr 102	**Mean Trans**	**Linear IN-1k**
Supervised Models													
Xent	200	67.7	89.8	72.5	57.5	43.7	39.8	67.9	91.8	91.1	88.0	71.0	77.2
Xent	90	72.8	91.0	74.0	59.5	56.8	48.4	70.7	92.0	90.8	93.0	74.9	76.2
ProtoNW	200	73.3	93.2	78.3	61.5	65.0	57.6	73.7	92.2	94.3	93.7	78.3	76.0
SupCon	200	72.5	93.8	77.7	61.5	64.8	58.6	74.6	**92.5**	93.6	94.1	78.4	**77.5**
Xent	1000	72.3	93.6	78.3	61.9	66.7	61.0	**74.9**	91.5	94.5	94.7	78.9	76.3
CMSF_{sup} top-*all*	200	73.7	94.2	**78.7**	62.1	**71.7**	**64.1**	73.4	**92.5**	94.5	**95.8**	**80.1**	75.7
CMSF_{sup} top-10	200	**74.9**	**94.4**	**78.7**	**62.7**	70.8	63.4	73.8	92.2	**94.9**	95.6	**80.1**	76.4
Self-Supervised Models													
SimCLR	1000	72.8	90.5	74.4	60.6	49.3	49.8	**75.7**	84.6	89.3	92.6	74.0	69.3
MoCo v2	800	72.5	**92.2**	74.6	59.6	50.5	53.2	74.4	84.6	90.0	90.5	74.2	71.1
BYOL	1000	**75.3**	91.3	**78.4**	**62.2**	**67.8**	**60.6**	75.5	**90.4**	**94.2**	**96.1**	**79.2**	**74.3**
MoCo v2	200	70.4	91.0	73.5	57.5	47.7	51.2	73.9	81.3	88.7	91.1	72.6	67.5
BYOL-asym	200	70.2	91.5	74.2	59.0	54.0	52.1	73.4	86.2	90.4	92.1	74.3	69.3
MSF	200	72.3	**92.7**	76.3	60.2	59.4	56.3	71.7	89.8	90.9	93.7	76.3	72.1
$\text{CMSF}_{\text{self}}$	200	**73.0**	92.2	**77.2**	**61.0**	**60.6**	**58.4**	**74.1**	**91.1**	**92.0**	**94.5**	**77.4**	**73.0**

TABLE 4.17. **Effect of k' in sampling NN from M':** In $\text{CMSF}_{\text{self}}$, we constrain top-$k$ NN search space in M with top-k' samples from M'. (**Left**) Increasing k' results in a drop in accuracy. The k in top-k is set to 5 for all values of k'. (**Right**) Histogram of the constrained sample ranks at epoch 50. As k' increases, the histogram shifts left, indicating greater overlap between constrained and unconstrained NN sets.

k'	5	10	20	40	80
NN	63.2	62.9	62.7	62.3	61.7
20-NN	66.4	66.1	65.9	65.6	65.0



4.3.4.2 Supervised Learning

Evaluation: Unlike cross-entropy (Xent [34, 258, 376]) baseline, SupCon [230], ProtoNW [401] and CMSF do not train a linear classifier during the pre-training stage. Thus, we use the pre-training dataset ImageNet-1k (IN-1k) for linear evaluation of the frozen features as done in SSL. For Xent, we use the linear classifier trained during pre-training. We use the same settings and datasets as self-supervised for transfer learning evaluation.

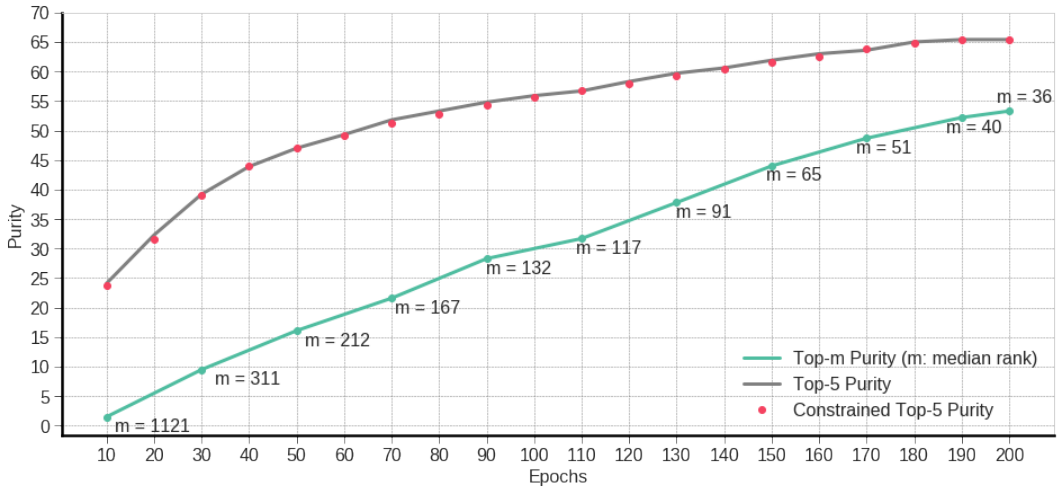


FIGURE 4.14. **Purity of constrained samples:** During training of $\text{CMSF}_{\text{self}}$, we plot purity of the top-5 samples in unconstrained set M (in black) and that of the top-5 samples in constrained set C (in red). The red curve is not significantly below the black one suggesting that the purity is not dropped by increasing the distance of the NNs. To show that elements in C may be far from the target u, we choose the 5^{th} element in C and find its rank in the set M. We calculate the median of this rank as m. The purity of the top-m elements of set M (green curve) is consistently lower than that of top-5 elements of the constrained set C (red curve). This suggests that one cannot maintain high purity by simply considering more NNs using a larger k.

FIGURE 4.15. **Evaluation on small labeled ImageNet** : We compare our model to MSF and other baselines on ImageNet 1% and 10% linear evaluation benchmarks. "Fine-tuned" refers to fine-tuning the entire backbone network instead of a single linear layer. $\text{CMSF}_{\text{self}}$ outperforms MSF on top-1 metric in both 1% and 10% settings.

Method	Fine-tuned	Epochs	Top-1		Top-5	
			1%	10%	1%	10%
Supervised	✓		25.4	56.4	48.4	80.4
PIRL [313]	✓	800	-	-	57.2	83.8
CO2 [468]	✓	200	-	-	71.0	85.7
SimCLR [70]	✓	1000	48.3	65.6	75.5	87.8
InvP [453]	✓	800	-	-	78.2	88.7
BYOL [160]	✓	1000	53.2	68.8	78.4	89.0
SwAV [57]	✓	800	**53.9**	**70.2**	**78.5**	**89.9**
MoCo v2 [76]	✗	800	51.5	63.6	77.6	86.1
BYOL [160]	✗	1000	55.7	**68.6**	80.0	**88.6**
CompRess [240]	✗	1K+130	**59.7**	67.0	**82.3**	87.5
MoCo v2 [76]	✗	200	43.6	58.4	71.2	82.9
BYOL-asym	✗	200	47.9	61.3	74.6	84.7
ISD [420]	✗	200	53.4	63.0	78.8	85.9
MSF [242]	✗	200	55.5	66.5	**79.9**	87.6
$\text{CMSF}_{\text{self}}$	✗	200	**56.4**	**67.5**	79.8	**87.7**

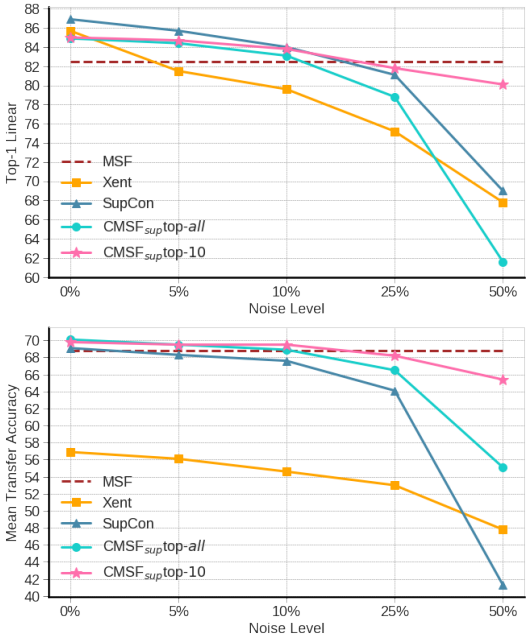


FIGURE 4.16. **Noisy supervised setting on ImageNet-100:** Our method is more robust to noisy annotation compared to Xent and SupCon. Also, using top-*all* degrades the results since all images from a single category are not guaranteed to be semantically related due to noisy labels. Mean Transfer Accuracy is the average over 10 transfer datasets.

Results: Results on IN-1k dataset are shown in Table 4.16. In top-*all* variation of our method, k is equal to the total size of C. SSL inspired methods like CMSF and SupCon significantly outperform Xent when trained for similar number of epochs. We observe that improvements in ImageNet performance do not always translate to transfer performance. Interestingly, CMSF performs the best on transfer evaluation, particularly on fine-grained datasets like Cars196 and Aircraft. We believe that the absence of explicit cross-entropy based optimization using the supervised labels preserves the multi-modal distribution of categories improving fine-grained performance. Supervised CMSF uses class labels only as a constraint for MSF during pre-training and does not explicitly optimize on the classification task. Superior performance of CMSF_{sup} top-10 demonstrates the importance of using distant yet semantically related neighbors as positives.

Noisy Labels: In the noisy setting, we use random i.i.d. noise to corrupt the labels (change the label randomly) of a percentage of images. We consider, 5%, 10%, 25%, and 50% label corruption (noise) rates. For faster experiments, we report results on the ImageNet-100 dataset [424] (Fig. 4.16). We observe a significantly higher degradation in performance of Xent baseline and CMSF_{sup} top-*all* compared to CMSF_{sup} top-10 at high noise levels. The gap between the approaches is larger on transfer learning. These observations indicate that NN based methods like CMSF are better suited for noisy constraint settings compared to approaches utilizing all samples of a class as positives. This robustness to label noise motivates our application of CMSF to self- and semi-supervised settings where pseudo-labels or the NNs of previous augmentations may be noisy.

Coarse-grained ImageNet: CMSF groups together only top-k neighbors and thus can help in preserving the latent structure of the data compared to top-*all*. To verify this, we consider a dataset with coarse-grained labels where this difference is pronounced. Based on the WordNet hierarchy, we merge each category in the ImageNet dataset to its parent class. We further ensure that no two classes are in the same path in the graph by merging the descendant into the ancestor class. The total number of classes is thus reduced from 1000 in ImageNet-1k to 93 in our ImageNet-coarse. We train CMSF and the baseline approaches in a supervised manner using the coarse labels and then evaluate on the fine-grained / original labels on ImageNet-1k validation set. In Table 4.18 we compare the top-*all* and top-k variants on the coarse grained version of ImageNet. CMSF_{sup} top-k sees a minor drop in performance compared to training on ImageNet-1k. However, methods in which all samples in a class are explicitly brought closer - CMSF_{sup} top-*all*, cross-entropy and supervised contrastive - see a huge drop in accuracy. More details on coarse-grained ImageNet are in the supplementary.

TABLE 4.18. **Supervised learning on coarse grained ImageNet:** We train on the coarse grained version of ImageNet (93 super categories) and perform linear evaluation on the original ImageNet-1k validation set with fine-grained labels (1000 categories).

Train Dataset	ImageNet-1k Validation Set			
	Xent	SupCon	CMSF_{sup} top-*all*	CMSF_{sup} top-10
ImageNet-1k	77.2	**77.5**	75.7	76.4
ImageNet-coarse	61.4	58.7	67.0	**74.2**

4.3.4.3 Semi-Supervised Learning

Implementation Details: We train a 2-layer MLP atop the cached target features of supervised set for pseudo-labeling. The pseudo-label training is performed twice per epoch (takes 40 seconds per training) and the label assignment is done in an online fashion for each mini-batch. The confidence threshold for pseudo-labeling is set to 0.85. We use the same optimizer settings as in self-supervised CMSF for the pre-training stage. Similar to S4L [518], we perform two stages of fine-tuning with supervised and pseudo-labels. We fine-tune the backbone network with two MLPs (as in PAWS [26]) on the 10% labeled set for 20 epochs and pseudo-label the train set. Samples above confidence threshold (nearly 30% of dataset) are combined with supervised set to fine-tune again for 20 epochs (more details in suppl.). The second fine-tuning is equivalent to 5 epochs with full data and is a small increase in our total compute. This is needed since we do not directly optimize cross-entropy loss in pre-training as in [344, 403, 484].

Evaluation: The final epoch parameters are used to perform evaluation. We report top-1 accuracy on the ImageNet validation set. We additionally report the total number of FLOPs for forward and backward passes (backward is 2× forward) through ResNet-50 backbone and the number of GPUs/TPUs used by each method in the pre-training stage (more details in suppl.).

Baselines: We compare the proposed approach ($CMSF_{semi}$) with self- and semi-supervised approaches. $CMSF_{semi}$-*basic* minimizes unconstrained MSF loss on the unlabeled examples (no pseudo-labeling) and CMSF loss on the labeled examples only. We provide comparison of PAWS method with different support set sizes. We train PAWS on 4x 16GB GPUs with maximum possible support set size (200 classes, 2 images/class) using code provided by the authors. We also report results using mixed precision training ($CMSF_{semi}$-*mix precision*) as in PAWS [26] with a higher batch size of 768 since it has lower memory requirement.

Results: $\text{CMSF}_{\text{semi}}$-mix precision achieves comparable performance to most methods with significantly less training and without the use of stronger augmentation schemes like RandAugment [94] (Table 4.19, Fig. 4.10). PAWS with a support set size of 6720 outperforms other approaches. However, this requires

TABLE 4.19. **Semi-supervised learning on ImageNet dataset with 10% labels:** FLOPs denotes the total number of FLOPS for forward and backward passes through ResNet-50 backbone while batch size denotes the sum of labeled and unlabeled samples in a batch. $\text{CMSF}_{\text{semi}}$-mix precision is compute and resource efficient, achieving SOTA performance at comparable compute. PAWS requires large number of GPUs to be compute efficient and its performance drastically drops with 4/8 GPUs. † Trained with stronger augmentations like RandAugment [94]. * TPUs are used.

Method	Epochs	Batch Size	GPUs	FLOPs ($\text{x}10^{18}$)	Top-1
Self-supervised Pre-training					
Mean Shift [242]	200	256	4	4	67.4
BYOL [160]	1000	4096	512*	40	68.8
SwAV [57]	800	4096	64	37	70.2
SimCLRv2 [72]	800	4096	128*	16	68.4
Semi-supervised Pre-training					
SimCLRv2 (+Self Dist) [72]	1200	4096	128*	20	70.5
UDA† [484]	800	15872	64*	10	68.1
FixMatch† [403]	300	6144	32*	7	71.5
MPL† [344]	800	2048	-	30	73.9
PAWS (support=6720) [26]	300	4096	64	21	75.5
PAWS (support=1680) [26]	100	256	8	15	70.2
PAWS (support=400) [26]	100	256	4	7	62.9
$\text{CMSF}_{\text{semi}}$-basic	200	256	4	4	68.6
$\text{CMSF}_{\text{semi}}$	200	256	4	4	69.9
$\text{CMSF}_{\text{semi}}$-mix precision	200	768	4	4	70.5

significantly higher compute (4.8× FLOPs) and resources (64 GPUs) compared to $\text{CMSF}_{\text{semi}}$-mix precision (4 GPUs). Since PAWS requires a large support set, it does not scale well to lower resource (4/8 GPUs) settings even if the total compute remains the same. When trained on only 4 GPUs, CMSF outperforms PAWS by **7.6%** points. Additional ablations and results on ImageNet-100 dataset are in supplementary.

4.3.5 Conclusion

MSF is a recent SSL method that pulls an image towards its nearest neighbors. We argue that the model can benefit from more diverse yet pure neighbors. Hence, we generalize MSF method by constraining the NN search. This opens the door to using the mean-shift idea to various settings of self-supervised, supervised, and semi-supervised. To construct the constraint, our SSL method uses cached augmentations from the previous epoch while the supervised and semi-supervised settings use labels or pseudo-labels. We show that our method outperforms SOTA approaches like MSF in SSL, PAWS in semi-supervised, and supervised contrastive in transfer-learning evaluation of supervised settings.

CHAPTER 5

Robustness of Efficient Models

5.1 Adversarial Attack on Compute of Efficient Vision Transformers

Recently, there has been a lot of progress in reducing the computation of deep models at inference time. These methods can reduce both the computational needs and power usage of deep models. Some of these approaches adaptively scale the compute based on the input instance. We show that such models can be vulnerable to a universal adversarial patch attack, where the attacker optimizes for a patch that when pasted on any image, can increase the compute and power consumption of the model. We run experiments with three different efficient vision transformer methods showing that in some cases, the attacker can increase the computation to the maximum possible level by simply pasting a patch that occupies only 8% of the image area. We also show that a standard adversarial training defense method can reduce some of the attack's success. We believe adaptive efficient methods will be necessary in the future to lower the power usage of expensive deep models, so we hope our paper encourages the community to study the robustness of these methods and develop better defense methods for the proposed attack. Code is available at:
https://github.com/UCDvision/SlowFormer

5.1.1 Introduction

The field of deep learning has recently made significant progress in improving model efficiency for inference. Two broad categories of methods can be distinguished: 1) those that reduce computation regardless of input, and 2) those that reduce the computation depending on the input (adaptively). Most methods, such as weight pruning or model quantization, belong to the first category which reduces computation by a constant factor regardless of the input. However, in many applications, the complexity of the perception task may differ depending on the input. For example, when a self-driving car is driving between lanes in an empty street, the perception may be simpler and require less computation when compared to driving in a busy city street scene. Interestingly, in some applications, simple scenes such as highway driving may account for the majority of the time. Therefore, we believe that adaptive computation reduction will become

an increasingly important research area in the future, especially when non-adaptive methods reach the lower bound of computation.

We argue that reduction of compute usually reduces power usage, which is crucial, particularly in mobile devices that run on battery, e.g., AR/VR headsets, humanoid robots, and drones. For instance, increasing the size of the battery for a drone may lead to a drastic reduction in its range due to the increased battery weight. This is important since the improvement in battery technology is much slower than compute technology. For instance, the battery capacity from iPhone [7] (1st generation) in 2007 to iPhone 15 Pro Max in 2023 improved from 5.18 watt-hour to 17.32 watt-hour (less than 4 times) while the compute has increased by a much larger factor. As an example, a delivery robot like Starship uses a 1,200Wh battery and can run for 12 hours [15], so it uses almost 100 watts for compute and mobility. Hence, an adversary increasing the power consumption of the perception unit by 20 watts, will reduce the battery life by almost 20%, which can be significant. Note that 20 watts increase in power is realistic assuming that it uses two NVIDIA Jetson Xavier NX cards (almost 20 watts each).

Key idea: Assuming that a perception method is reducing the computation adaptively with the input, an adversary can attack the model by modifying the input to increase the computation and power consumption. **Our goal** is to design a universal adversarial patch that when pasted on any input image, it will increase the computation of the model leading to increased power consumption. We believe this is an important vulnerability, particularly for safety-critical mobile systems that run on battery.

Please note that in this paper, we do not experiment with real hardware to measure the power consumption. Instead, we report the change in FLOPs of the inference time assuming that the power consumption is positively correlated with the number of FLOPs, as studied in [385].

We design our attack, SlowFormer, for three different methods (A-ViT [501], ATS [127], and Ada-VIT [310]) that reduce the computation of vision transformers. These methods generally identify the importance of each token for the final task and drop the insignificant ones to reduce the computation. We show that in all three cases, our attack can increase the computation by a large margin, returning it to the full-compute level (non-efficient baseline) for all images in some settings. Our threat model is agnostic to the accuracy of the model and attacks the computation and power consumption only. Figure 5.1 shows our attack.

There are some prior works that design a pixel-level perturbation attack to increase the compute of the model. We believe universal patch-based attacks that do not change with the input image (generalize from training data to test data) are much more practical in real applications. Note that to modify the pixel values

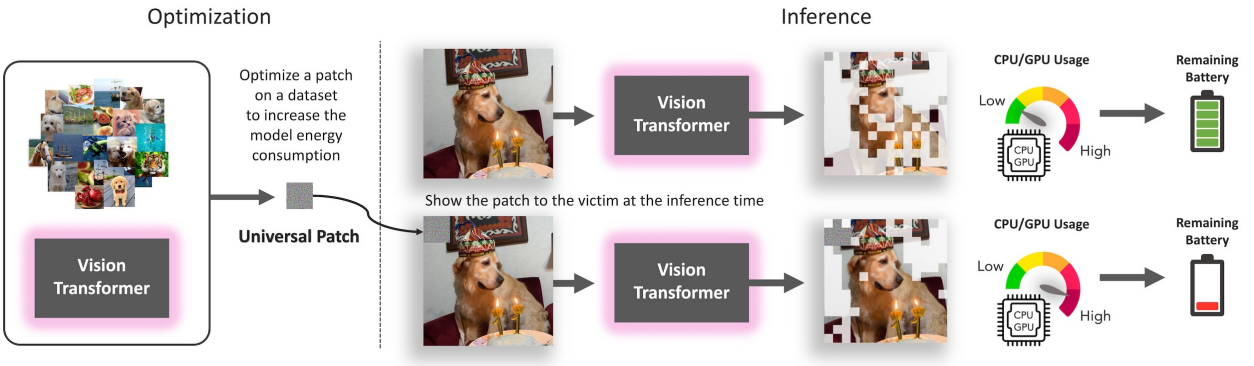


FIGURE 5.1. **Computation and Energy Attack on Vision Transformers:** Given a pre-trained input-dependent computation efficient model, the adversary first attaches an adversarial patch to all images in a dataset and optimizes this patch with our method such that it maximizes the model's computation for each sample. During inference, the adversary modifies the input of the victim's model by applying the learnt patch to it. This results in an increase in compute in the victim's model. The attack will thus potentially slowdown and also lead to increased energy consumption and CPU/GPU usage on the victim's device.

on a real robot, the attacker needs to access and manipulate the image between the camera and compute modules, which is impossible in many applications.

Contributions: We show that efficient vision transformer methods are vulnerable to a universal patch attack that can increase their compute and power usage. We demonstrate this through experiments on three different efficient transformer methods. We show that an adversarial training defense can reduce attack success to some extent.

5.1.2 Related Work

Vision Transformers: The popularity of transformers [446] in vision has grown rapidly since the introduction of the first vision transformer [110, 431]. Recent works demonstrate the strength of vision transformers on a variety of computer vision tasks [55, 81, 110, 288, 362, 430, 507, 533, 536, 541]. Moreover, transformers are the backbone of recent Self-Supervised Learning (SSL) models [59, 174], and vision-language models [353]. In our work, we design an attack to target the computation and energy efficiency of vision transformers.

Efficient Vision Transformers: Due to the recent importance and popularity of vision transformers, many works have started to study the efficiency of vision transformers [44, 227, 504]. To accomplish this, some lines of work study token pruning with the goal of removing uninformative tokens in each layer [127, 304, 310, 360, 501]. ToMe [40] merges similar tokens in each layer to decrease the computation. Some works address quadratic computation of self-attention module by introducing linear attention [23, 225, 237,

293, 392]. Efficient architectures [190, 289] that limit the attention span of each token have been proposed to improve efficiency. In our paper, we attack token pruning based efficient transformers where the computation varies based on the input samples [127, 310, 501].

Dynamic Computation: There are different approaches to reducing the computation of vision models, including knowledge distillation to lighter network [188, 294], model quantization [285, 363] and model pruning [263]. In these methods, the computation is fixed during inference. In contrast to the above models, some works address efficiency by having variable computation based on the input. The intuition behind this direction is that not all samples require the same amount of computation. Several recent works have developed models that dynamically exit early or skip layers [39, 120, 136, 158, 161, 201, 419, 449, 467] and selectively activate neurons, channels or branches for dynamic width [36, 51, 63, 129, 145, 187, 200, 510] depending on the complexity of the input sample. Zhou et al. show that not all locations in an image contribute equally to the predictions of a CNN model [538], encouraging a new line of work to make CNNs more efficient through spatially dynamic computation. Pixel-Wise dynamic architectures [53, 64, 123, 234, 368, 450, 487] learn to focus on the significant pixels for the required task while Region-Level dynamic architectures perform adaptive inference on the regions or patches of the input [139, 273]. Finally, lowering the resolution of inputs decreases computation, but at the cost of performance. Conventional CNNs process all regions of an image equally, however, this can be inefficient if some regions are “easier" to process than others [195]. Correspondingly, [497, 498] develop methods to adaptively scale the resolution of images.

Transformers have recently become extremely popular for vision tasks, resulting in the release of a few input-dynamic transformer architectures [126, 310, 501]. Fayyaz et al. [126] introduce a differentiable parameter-free Adaptive Token Sampler (ATS) module which scores and adaptively samples significant tokens. ATS can be plugged into any existing vision transformer architecture. A-ViT [501] reduces the number of tokens in vision transformers by discarding redundant spatial tokens. Meng et al. [310] propose AdaViT, which trains a decision network to dynamically choose which patch, head, and block to keep/activate throughout the backbone.

Adversarial Attack: Adversarial attacks are designed to fool models by applying a targeted perturbation or patch on an image sample during inference [154, 251, 414]. These methods can be incorporated into the training set and optimized to fool the model. Correspondingly, defenses have been proposed to mitigate the effects of these attacks [133, 269, 331, 482]. Patch-Fool [140] considers adversarial patch-based attacks on transformers. Some recent works [300, 469, 522] also study and design methods for the transferability of

adversarial attacks on vision transformers. However, most prior adversarial attacks target model accuracy, ignoring model efficiency.

Energy Attack: Very recently, there have been a few works on energy adversarial attacks on neural networks. In ILFO [170], Haque et al. attack two CNN-based input-dynamic methods: SkipNet [467] and SACT [136] using image specific perturbation. DeepSloth [191] attack focuses on slowing down early-exit methods, reducing their energy efficiency by 90-100%. GradAuto [329] successfully attacks methods that are both dynamic width and dynamic depth. NICGSlowDown and TransSlowDown [68, 69] attack neural image caption generation and neural machine translation methods, respectively. All these methods primarily employ image specific perturbation based adversarial attack. SlothBomb injects efficiency backdoors to input-adaptive dynamic neural networks [66] and NodeAttack [171] attacks Neural Ordinary Differential Equation models, which use ordinary differential equation solving to dynamically predict the output of a neural network. Our work is closely related to ILFO [170], DeepSloth [191] and GradAuto [329] in that we attack the computational efficiency of networks. However, unlike these methods, we focus on designing an adversarial patch-based attack that is universal and on vision transformers. We additionally provide a potential defense for our attack. We use a patch that generalizes from train to test set and thus we do not optimize per sample during inference. Our patch-based attack is especially suited for transformer architectures [140].

5.1.3 Computation and Energy Attack

5.1.3.1 Threat Model:

We consider a scenario where the adversary has access to the victim's trained deep model and modifies its input such that the energy consumption and computational demand of the model is increased. The attack is agnostic to model accuracy. To make the setting more practical, instead of perturbing the entire image, we assume that the adversary can modify the input image by only pasting a patch [45, 381] on it and that the patch is universal, that is, image independent. During inference, a pretrained patch is pasted on the test image before propagating it through the network.

In this paper, we attack three state-of-the-art efficient transformers. Since the attacker manipulates only the input image and not the network parameters, the attacked model must have dynamic computation that depends on the input image. As stated earlier, several recent works have developed such adaptive efficient models and we believe that they will be more popular in the future due to the limits of non-adaptive efficiency improvement.

5.1.3.2 Attack on Efficient Vision Transformers:

Universal Adversarial Patch: We use an adversarial patch to attack the computational efficiency of transforms. The learned patch is universal, that is, a single patch is trained and is used during inference on all test images. The patch generalizes across images but not across models. The patch optimization is performed only on the train set. The patch is pasted on an image by replacing the image pixels using the patch. We assume the patch location does not change from train to test. The patch pixels are initialized using i.i.d. samples from a uniform distribution over $[0, 255]$. During each training iteration, the patch is pasted on the mini-batch samples and is updated to increase the computation of the attacked network. The patch values are projected onto $[0, 255]$ and quantized to 256 uniform levels after each iteration. Note that we use a pretrained network and do not update its parameters either in the training or in the evaluation of our attack. During inference, the trained patch is pasted on the test images and the computational efficiency of the network on the adversarial image is measured.

Here, we focus on three methods employing vision transformers for the task of image classification. All these methods modify the computational flow of the network based on the input image for faster inference. A pretrained model is used for the attack and is not modified during our adversarial patch training. We first provide a brief background of each method before describing our attack.

Attacking A-ViT :

Background: A-ViT [501] adaptively prunes image tokens to achieve speed-up in inference with minimal loss in accuracy. For a given image, a dropped token will not be used again in the succeeding layers of the network. Let x be the input image and $\{t^l\}_{1:K}$ be the corresponding K tokens at layer l. An input-dependent halting score h_k^l for a token k at layer l is calculated and the token is dropped at layer N_k where its cumulative halting score exceeds a fixed threshold value $1-\epsilon$ for the first time. The token is propagated until the final layer if its score never exceeds the threshold. Instead of introducing a new parameter for h_k^l, the first dimension of each token is used to predict the halting score for the corresponding token. The network is trained to maximize the cumulative halting score at each layer and thus drop the tokens earlier. The loss, termed ponder loss, is given by:

$$L_{\text{ponder}} = \frac{1}{K}\sum_{k=1}^{K}(N_k + r_k), \qquad r_k = 1 - \sum_{l=1}^{N_{k-1}} h_k^l \tag{5.1}$$

Additionally, A-ViT enforces a Gaussian prior on the expected halting scores of all tokens via KL-divergence based distribution loss, $L_{\text{distr.}}$. These loss terms are minimized along with the task-specific loss L_{task}. Thus, the overall training objective is $L = L_{\text{task}} + \alpha_d L_{\text{distr.}} + \alpha_p L_{\text{ponder}}$ where α_d and α_p are hyperparameters.

Attack: Here, we train the patch to increase the inference compute of a trained A-ViT model. Since we are interested in the compute and not task-specific performance, we simply use $-(\alpha_d L_{\text{distr.}} + \alpha_p L_{\text{ponder}})$ as our loss. It is possible to preserve (or hurt) the task performance by additionally using $+L_{\text{task}}$ (or $-L_{\text{task}}$) in the loss formulation.

Attacking AdaViT:

Background: To improve the inference efficiency of vision transformers, AdaViT [310] inserts and trains a decision network before each transformer block to dynamically decide which patches, self-attention heads, and transformer blocks to keep/activate throughout the backbone. The l^{th} block's decision network consists of three linear layers with parameters $W_l = W_l^p, W_l^h, W_l^b$ which are then multiplied by each block's input Z_l to get m.

$$(m_l^p, m_l^h, m_l^b) = (W_l^p, W_l^h, W_l^b) Z_l \tag{5.2}$$

The value m is then passed to sigmoid function to convert it to a probability value used to make the binary decision of keep/discard. Gumbel-Softmax trick [298] is used to make this decision differentiable during training. Let M be the keep/discard mask after applying Gumbel-Softmax on m. The loss on computation is given by:

$$\begin{aligned} L_{usage} = &\Big(\frac{1}{D_p}\sum_{d=1}^{D_p} M_d^p - \gamma_p\Big)^2 + \Big(\frac{1}{D_h}\sum_{d=1}^{D_h} M_d^h - \gamma_h\Big)^2 \\ &+ \Big(\frac{1}{D_b}\sum_{d=1}^{D_b} M_d^b - \gamma_b\Big)^2 \end{aligned} \tag{3}$$

where D_p, D_h, D_b represent the number of total patches, heads, and blocks of the entire transformer, respectively. γ_p, γ_h, γ_b denote the target computation budgets i.e. the percentage of patches/heads/blocks to keep. The total loss is a combination of task loss (cross-entropy) and computation loss: $L = L_{ce} + L_{usage}$.

Attack: To attack this model, we train the patch to maximize the computation loss L_{usage}. More specifically, we set the computation-target γ values to 0 and negate the L_{usage} term in Eq. 5.3. As a result, the

patch is optimized to maximize the probability of keeping the corresponding patch (p), attention head (h), and transformer block (b). We can also choose to attack the prediction performance by selectively including or excluding the L_{ce} term.

Attacking ATS:

Background: Given N tokens with the first one as the classification token, the transformer attention matrix A is calculated by the following dot product:
$A = \text{Softmax}\left(QK^T / \sqrt{d}\right)$ where $\sqrt{d}$ is a scaling coefficient, d is the dimension of tokens, Q, K and V are the query, key and value matrices, respectively. The value $A_{1,j}$ denotes the attention of the classification token to token j. ATS [127] assigns importance score S_j for each token j by measuring how much the classification token attends to it:

$$S_j = \frac{A_{1,j} \times ||V_j||}{\sum_{i=2} A_{1,i} \times ||V_i||} \tag{4}$$

The importance scores are converted to probabilities and are used to sample tokens, where tokens with a lower score have more of a chance of being dropped.

Attack: Since ATS uses inverse transform sampling, it results in fewer samples if the importance distribution is sharp. To maximize the computation in ATS, we aim to obtain a distribution of scores with high entropy to maximize the number of retained tokens. Therefore, we optimize the patch so that the attention of the classification token over other tokens is a uniform distribution using the following MSE loss:

$$L = \sum_{i=2}^{N} ||A_{1,i} - \frac{1}{N}||_2^2 \tag{5}$$

Note that one can optimize S to be uniform, but we found the above loss to be easier to optimize. For a multi-head attention layer, we calculate the loss for each head and then sum the loss over all heads. Moreover, ATS can be applied to any layer of a vision transformer. For a given model, we apply our loss at all ATS layers and use a weighted summation for optimization.

5.1.4 Defense

An obvious defense, although weak, will be to use non-dynamic efficient methods only, e.g., weight pruning, where the reduction in compute is deterministic and does not depend on the input. However, most

such methods do not achieve high levels of computation efficiency since they do not take advantage of the simplicity of images.

We adopt standard adversarial training as a better defense method for our attack. In the standard way, at each iteration of training the model, one would load an image, attack it, and then use it with correct labels in training the model. We cannot adopt this out-of-the-box since our attack generalizes across images and is not dependent on a single image only. To do this, we maintain a set of adversarial patches, and at each iteration sample one of them randomly (uniformly), and use it at the input while optimizing the original loss of the efficient model to train a robust model. To adapt the set of adversarial patches to the model being trained, we interrupt the training at every 20% mark of each epoch and optimize for a new patch to be added to the set of patches. To limit the computational cost of training, we use only 500 iterations to optimize for a new patch, which results in an attack with reasonable accuracy compared to our main results.

5.1.5 Experiments

5.1.5.1 Attack on Efficient Vision Transformers

Dataset: We evaluate the effectiveness of our attack on two datasets: ImageNet-1K [101] and CIFAR-10 [245]. ImageNet-1K contains 1.3M images in the train set and 50K images in the validation set with 1000 total categories. CIFAR-10 has 50K images for training and 10K images for validation with 10 total categories.

Metrics: We report Top-1 accuracy and average computation in terms of GFLOPs for both attacked and unattacked models. Similar to Attack Success Rate in a standard adversarial attack, we introduce a metric: Attack Success to quantify the efficacy of the attack. We define Attack Success as the number of FLOPs increased by the attack divided by the number of FLOPs decreased by the efficient method. Attack Success = $\frac{(\text{FLOPs}_{\text{attack}} - \text{FLOPs}_{\text{min}})}{(\text{FLOPs}_{\text{max}} - \text{FLOPs}_{\text{min}})}$ where $\text{FLOPs}_{\text{min}}$ is the compute of the efficient model and $\text{FLOPs}_{\text{max}}$ is that of the original inefficient model. Attack Success is thus capped at 100% while a negative value denotes a reduction in FLOPs. Note that our Attack Success metric illustrates the effectiveness of an attack in reversing the FLOPs reduction of a particular method.

Baselines: We propose three alternative approaches to SlowFormer (ours) to generate the patch.

Random Patch: A simple baseline is to generate a randomly initialized patch. We sample IID pixel values from a uniform distribution between 0 and 255 to create the patch.

NTAP: We consider a standard adversarial patch that is trained to attack the model task performance instead of compute. We use a non-targeted universal adversarial patch (NTAP) to attack the model. We train the patch to fool the model by misclassifying the image it is pasted on. We use the negative of the cross-entropy loss with the predicted and ground-truth labels as the loss to optimize the patch.

TAP: We train a universal targeted adversarial patch (TAP). The patch is optimized to classify all images in the train set to a single fixed category. Similar to NTAP, the adversarial attack here is on task performance and not computation. We experiment with ten randomly generated target category labels and report the averaged metrics.

Implementation Details: We use PyTorch [335] for all experiments. Unless specified, we use a patch of size 64×64, train and test on 224×224 images, and we paste the patch on the top-left corner. Note that our patch occupies just 8% of the total area of an input image. We use AdamW [292] optimizer to optimize the patches and use 4 NVIDIA RTX 3090 GPUs for each experiment. We use varying batch sizes and learning rates for each of the computation-efficient methods.

ATS Details: As in ATS [127], we replace layers 3 through 9 of ViT networks with the ATS block and set the maximum limit for the number of tokens sampled to 197 for each layer. We train the patch for 2 epochs with a learning rate of 0.4 for ViT-Tiny and $lr = 0.2$ for ViT-Base and ViT-Small. We use a batch size of 1024 and different loss coefficients for each layer of ATS. For DeiT-Tiny we use $[1.0, 0.2, 0.2, 0.2, 0.01, 0.01, 0.01]$, for DeiT-Small we use $[1.0, 0.2, 0.05, 0.01, 0.005, 0.005, 0.005]$, and for DeiT-Base we use $[2.0, 0.1, 0.02, 0.01, 0.005, 0.005, 0.005]$. The weights are vastly different at initial and final layers to account for the difference in loss magnitudes across layers.

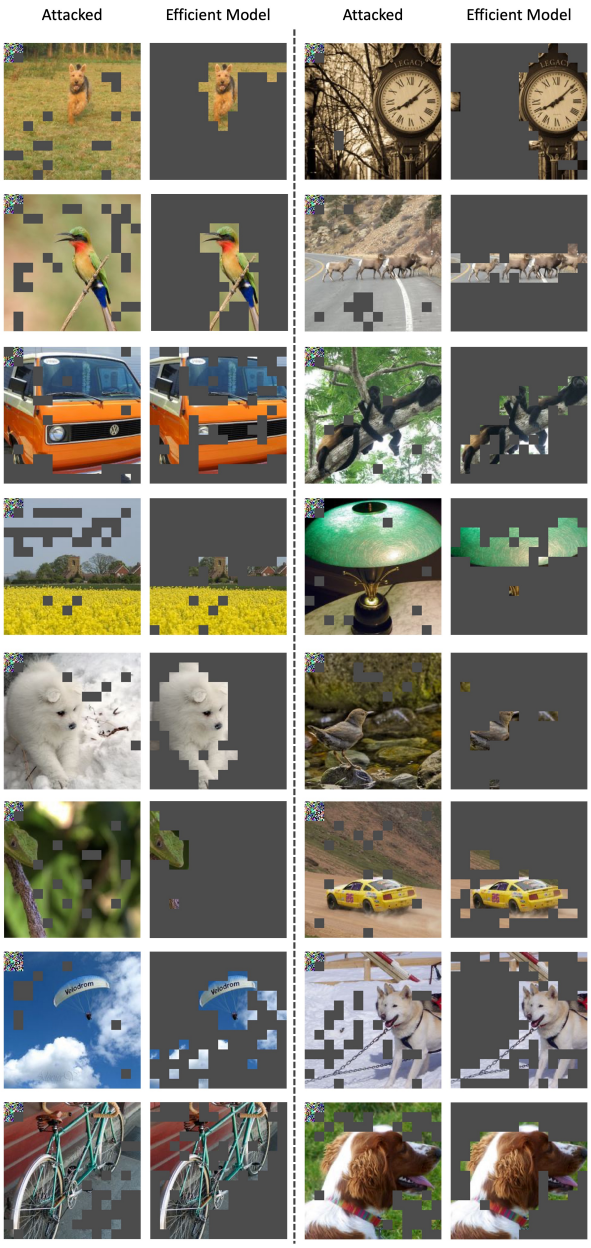
A-ViT Details: When attacking A-VIT [501], the patches are optimized for one epoch with a learning rate of 0.2 and a batch size of 512 (128×4GPUs) using AdamW [292] optimizer. We optimize the patches for 4 epochs for patch length 32 and below. For CIFAR-10 experiments, the images are resized from 32×32 to 256×256 and a 224×224 crop is used as the input. For the training of adversarial defense, we generate 5 patches per epoch of adversarial training and limit the number of iterations for patch generation to 500. The learning rate for patch optimization is increased to 0.8 for faster convergence.

AdaViT Details: For AdaViT [310], we first freeze the weights and use a learning rate of 0.2 and a batch size of 128 with 4 GPUs for patch optimization. We use AdamW [292] optimizer with no decay and train for 2 epochs with a patch size of 64 x 64. We train on the ImageNet-1k train dataset and evaluate it on the test set.

TABLE 5.1. **Computation and Energy Attack on Efficient Vision Transformers:** Comparison of the effect of our attack with baselines: No Attack, Random Patch, targeted (TAP), and non-targeted (NTAP) adversarial patches applied to three input-dynamic computation efficient pre-trained models of varying architectures. The maximum possible compute for a given architecture is provided in bold. On A-ViT , we completely undo the efficiency gains obtained by the efficient method through our attack, achieving Attack Success of 100%. We achieve high Attack Success on all approaches while the baselines expectedly do not contribute to increase in compute.

Method	Attack	Model GFLOPs	Top-1 Acc	Attack Success
	ViT-Tiny	**1.3**	-	-
	No attack	0.87	71.4%	-
	Random Patch	0.87	70.8%	-1%
A-ViT	TAP	0.85	0.1%	-5%
	NTAP	0.83	0.1%	-10%
	SlowFormer (ours)	1.3	4.7%	100%
	ViT-Small	**4.6**	-	-
	No attack	3.7	78.8%	-
	Random Patch	3.7	78.4%	-2%
A-ViT	TAP	3.6	0.1%	-12%
	NTAP	3.6	0.1%	-7%
	SlowFormer (ours)	4.6	2.3%	99%
	ViT-Tiny	**1.3**	-	-
	No attack	0.84	70.3%	-
	Random Patch	0.83	69.8%	-2%
ATS	TAP	0.76	0.1%	-17%
	NTAP	0.61	0.1%	-50%
	SlowFormer (ours)	1.0	1.2%	35%
	ViT-Small	**4.6**	-	-
	No attack	3.1	79.2%	-
	Random Patch	3.1	78.6%	-1%
ATS	TAP	3.0	0.1%	-7%
	NTAP	2.4	0.1%	-47%
	SlowFormer (ours)	4.0	1.0%	60%
	ViT-Base	**17.6**	-	-
	No attack	12.6	81.3%	-
	Random Patch	12.5	81.2%	-2%
ATS	TAP	12.0	0.1%	-12%
	NTAP	11.0	0.1%	-32%
	SlowFormer (ours)	15.4	0.2%	52%
	ViT-Small	**4.6**	-	-
	No attack	2.25	77.3%	-
	Random Patch	2.20	76.9%	-2%
AdaViT	TAP	2.28	0.1%	1%
	NTAP	2.15	0.1%	-4%
	SlowFormer (ours)	3.2	0.4%	40%

FIGURE 5.2. **Visualization of our Energy Attack on Vision Transformers:** We visualize the A-ViT-Small with and without our attack. We use patch size of 32 for the attack (on the top-left corner). We show pruned tokens at layer 8 of A-ViT-Small. Our attack can recover most of the pruned tokens, resulting in increased computation and power consumption. Note that although the patch is reasonably small and is in the corner of the view, it can affect the whole computational flow of the network. This is probably due to the global attention mechanism in transformers.



Results. The results of our attack, SlowFormer , on various methods on ImageNet dataset are shown in table 5.1. In A-ViT, we successfully recover 100% of the computation reduced by A-ViT . Our attack has an Attack Success of 60% on ATS and 40% on AdaViT with ViT-Small. A random patch attack has little effect on both the accuracy and computation of the method. Both standard adversarial attack baselines, TAP and NTAP, reduce the accuracy to nearly 0%. Interestingly, these patches further decrease the computation of the efficient model being attacked. This might be because of the increased importance of adversarial patch tokens to the task and thus reduced importance of other tokens. Targeted patch (TAP) has a significant reduction in FLOPs on the ATS method. Since the token dropping in ATS relies on the distribution of attention values of classification tokens, a sharper distribution due to the increased importance of a token can result in a reduction in computation. The computation increase with SlowFormer for AdaViT is comparatively low. To investigate, we ran a further experiment using a patch size of 224×224 (entire image size) to find the maximum possible computation for an image. This resulted in 4.18 GFLOPs on the ImageNet-1K validation set, which is markedly lower than the limit of 4.6. Using this as an upper-bound of GFLOPs increase, SlowFormer achieves a 49% Attack Success.

We report the results on CIFAR-10 dataset in Table 5.2. The efficient model (A-ViT) drastically reduces the computation from 1.26 GFLOPs to 0.11 GFLOPs. Most of the tokens are dropped as early as layer two in the efficient model. SlowFormer is able to effectively attack even in such extreme scenarios, achieving an Attack Success of 40% and increasing the mean depth of tokens from nearly one to five. SlowFormer is similarly effective on ATS with an Attack Success of 34%.

We additionally visualize the effectiveness of our attack in Figure 5.2. The un-attacked efficient method retains only highly relevant tokens at the latter layers of the network. However, our attack results in nearly the entire image being passed through all layers of the model for all inputs. In Fig. 5.3, we visualize the optimized patches for each of the three efficient methods.

5.1.6 Ablations:

We perform all ablations on the A-ViT approach using their pretrained ViT-Tiny architecture model.

Accuracy controlled compute adversarial attack: As seen in Table 5.1, our attack can not only increase the computation, but also reduce the model accuracy. This can be desirable or hurtful based on the attacker's goals. A low-accuracy model might be an added benefit, similar to regular adversaries, but might also lead to the victim detecting the attack. We show that it is possible to attack the computation of the model while

TABLE 5.2. **Results on CIFAR10 dataset.** We report results on CIFAR10 dataset to show that our attack is not specific to ImageNet alone. CIFAR-10 is a small dataset compared to ImageNet and thus results in an extremely efficient A-ViT model. Our attack increases the FLOPs from 0.11 to 0.58 which restores nearly 41% of the original reduction in the FLOPs.

Method	Model FLOPs	Top-1 Acc	Attack Success
ViT-Tiny	1.26	95.9%	-
A-ViT-Tiny	0.11	95.8%	-
SlowFormer (ours)	0.58	60.2%	41%
ATS-Tiny	0.85	94.7%	-
SlowFormer (ours)	0.99	24.7%	34.1%

TABLE 5.3. **Accuracy controlled compute adversarial attack:** We attack the the efficiency of A-ViT while either maintaining or destroying its classification performance. We observe that our attack can achieve a huge variation in task performance without affecting the Attack Success. The ability to attack the computation without affecting the task performance might be crucial in some applications.

Attack	Model GFLOPs	Attack Success	Top-1 Acc
ViT-Tiny	1.26	-	-
No attack	0.87	-	71.4%
Acc agnostic	1.26	100%	4.7%
Preserve acc	1.23	92%	68.5%
Destroy acc	1.26	100%	0.1%

either preserving or destroying the task performance by additionally employing a task loss in the patch optimization. Table 5.3 indicates that the accuracy can be significantly modified while maintaining a high Attack Success.

Effect of patch size: We vary the patch size from 64×64 to 16×16 (just a single token) and report the results in Table 5.4. Interestingly, our attack with ViT-Small has a 73% Attack Success with a 32×32 patch size, which occupies only 2% of the input image area.

Effect of patch location: We vary the location of the patch to study its effect on the Attack Success of the model. We randomly sample a location in the image for where we paste the patch on. We perform five such experiments and observe an Attack Success of 100% for all patch locations.

Perturbation attack: While we focus on patch based attacks in this paper, efficient transformers are also susceptible to perturbation based attacks (table 5.5). In perturbation attacks, all pixels in the image can be modified, but with an upper bound on the ℓ_∞ norm of the perturbation.

5.1.7 Adversarial training based defense

Our simple defense that is adopted from standard adversarial training is explained in Section 5.1.4. The results for defending against attacking A-ViT are shown in Table 5.6. The original A-ViT reduces the GFLOPs from 1.26 to 0.87, our attack increases it back to 1.26 with 100% attack success. The proposed

TABLE 5.4. **Effect of patch size:** Analysis of the effect of adversarial patch size on the attack success rate on A-ViT. Our attach is reasonably successful even using a small patch size (32×32), which is only 2% of the image area. Interestingly, a small patch on the corner of the view affects the computational flow of the entire transformer model. This might be due to the global attention mechanism in transformers.

Patch Size (Area)	Model GFLOPs	Top-1 Accuracy	Attack Success
ViT-Tiny	1.26	-	-
A-ViT-Tiny	0.87	71.4%	-
64 (8%)	1.26	4.7%	100%
48 (5%)	1.26	1.8%	99%
32 (2%)	1.22	17.4%	90%
16 (0.5%)	0.98	63.3%	27%
ViT-Small	4.6	-	-
A-ViT-Small	3.7	78.8%	-
64 (8%)	4.6	2.3%	99%
48 (5%)	4.6	5.1%	98%
32 (2%)	4.4	39.5%	78%
16 (0.5%)	3.8	78.2%	16%

TABLE 5.5. **Attack with adversarial perturbation on ImageNet**. The efficient methods are also susceptible universal perturbation based attacks. We use an ℓ_∞ bound on the perturbation.

Method	Epsilon (/255.)	Attack	Model GFLOPs	Top-1 Accuracy	Attack Success
		ViT-Tiny	**1.3**	-	-
	-	No attack	0.87	71.4%	-
A-ViT	16	SlowFormer	1.15	6.1%	73%
	32	SlowFormer	1.25	0.5%	98.4%
	-	No attack	0.84	70.3%	-
ATS	16	SlowFormer	0.98	15.6%	30.4%
	32	SlowFormer	1.04	0.8%	43.5%
		ViT-Small	**4.6**	-	-
	-	No attack	3.7	78.8%	-
A-ViT	16	SlowFormer	4.48	20%	86.4%
	32	SlowFormer	4.59	1%	98.1%
	-	No attack	3.1	79.2%	-
ATS	16	SlowFormer	3.6	31.0%	33.3%
	32	SlowFormer	3.8	3.6%	46.7%
	-	No attack	2.25	77.3%	-
AdaVit	16	SlowFormer	3.0	26.1%	31.9%
	32	SlowFormer	3.2	2.8%	40.4%

TABLE 5.6. **Defense using adversarial training:** We propose and show the impact of our defense for our adversarial attack on A-ViT. Our defense is simply maintaining a set of universal patches and training the model to be robust to a random sample of those at each iteration. The defense reduces the computation to some extent (1.26 to 1.01), but is still far from the unattacked model (0.87).

Method	GFLOPs	Top-1 Acc.	Attack Success
No attack	0.87	71.4	-
SlowFormer	1.26	4.7%	100%
Adv Defense + SlowFormer	1.0	65.8%	34%

FIGURE 5.3. **Visualization of optimized patch:** We show the learned universal patches for each of the three efficient methods.

defense reduces the GFLOPs to 1.01 which is still higher than the original 0.87. We hope our paper encourages the community to develop better defense methods to reduce the vulnerability of efficient vision transformers.

5.1.8 Conclusion

Recently, we have seen efficient vision transformer models in which the computation is adaptively modified based on the input. We argue that this is an important research direction and that there will be more progress in this direction in the future. However, we show that the current methods are vulnerable to a universal adversarial patch that increases the computation and thus power consumption at inference time. Our experiments show promising results for three SOTA efficient transformer models, where a small patch that is optimized on the training data can increase the computation to the maximum possible level in the testing data in some settings. We also propose a defense that reduces the effectiveness of our attack. We hope that our paper will encourage the community to study such attacks and develop better defense methods on various machine learning methods, including generative models, that reduce the computation adaptively with the input.

CHAPTER 6

Training Data Efficiency

6.1 GeNIe: Generative Hard Negative Images Through Diffusion

Data augmentation is crucial in training deep models, preventing them from overfitting to limited data. Recent advances in generative AI, e.g., diffusion models, have enabled more sophisticated augmentation techniques that produce data resembling natural images. We introduce `GeNIe` a novel augmentation method which leverages a latent diffusion model conditioned on a text prompt to combine two contrasting data points (an image from the source category and a text prompt from the target category) to generate challenging augmentations. To achieve this, we adjust the noise level (equivalently, number of diffusion iterations) to ensure the generated image retains low-level and background features from the source image while representing the target category, resulting in a *hard negative* sample for the source category. We further automate and enhance `GeNIe` by adaptively adjusting the noise level selection on a per image basis (coined as `GeNIe-Ada`), leading to further performance improvements. Our extensive experiments, in both few-shot and long-tail distribution settings, demonstrate the effectiveness of our novel augmentation method and its superior performance over the prior art. Our code is available here: https://github.com/UCDvision/GeNIe

6.1.1 Introduction

Augmentation has become an integral part of training deep learning models, particularly when faced with limited training data. For instance, when it comes to image classification with limited number of samples per class, model generalization ability can be significantly hindered. Simple transformations like rotation, cropping, and adjustments in brightness artificially diversify the training set, offering the model a more comprehensive grasp of potential data variations.

Hence, augmentation can serve as a practical strategy to boost the model's learning capacity, minimizing the risk of overfitting and facilitating effective knowledge transfer from limited labelled data to real-world scenarios. Various image augmentation methods, encompassing standard transformations, and learning-based approaches have been proposed [92, 96, 438, 513, 521]. Some augmentation strategies combine two

images possibly from two different categories to generate a new sample image. The simplest ones in this category are MixUp [521] and CutMix [513] where two images are combined in the pixel space. However, the resulting augmentations often do not lie within the manifold of natural images and act as out-of-distribution samples that will not be encountered during testing.

Recently, leveraging generative models for data augmentation has gained an upsurge of attention [180, 297,375,438]. These interesting studies, either based on fine-tuning or prompt engineering of diffusion models, are mostly focused on generating *generic augmentations* without considering the impact of other classes and incorporating that information into the generative process for a classification context. We take a different approach to generate challenging augmentations near the decision boundaries of a downstream classifier. Inspired by diffusion-based image editing methods [297, 309] some of which are previously used for data augmentation, we propose to use conditional latent diffusion models [372] for generating *hard negative* images. Our core idea (coined as `GeNIe`) is to sample source images from various categories and prompt the diffusion model with a contradictory text corresponding to a different target category. We demonstrate that the choice of noise level (or equivalently number of iterations) for the diffusion process plays a pivotal role in generating images that semantically belong to the target category while retaining low-level features from the source image. We argue that these generated samples serve as *hard negatives* [303, 494] for the source category (or from a dual perspective hard positives for the target category). To further enhance `GeNIe`, we propose an adaptive noise level selection strategy (dubbed as `GeNIe-Ada`) enabling it to adjust noise levels automatically per sample.

To establish the impact of `GeNIe`, we focus on two challenging scenarios: *long-tail* and *few-shot* settings. In real-world applications, data often follows a long-tail distribution, where common scenarios dominate and rare occurrences are underrepresented. For instance, a person jaywalking a highway causes models to struggle with such unusual scenarios. Combating such a bias or lack of sufficient data samples during model training is essential in building robust models for self-driving cars or surveillance systems, to name a few. Same challenge arises in few-shot learning settings where the model has to learn from only a handful of samples. Our extensive quantitative and qualitative experimentation, on a suite of few-shot and long-tail distribution settings, corroborate the effectiveness of the proposed novel augmentation method (`GeNIe`, `GeNIe-Ada`) in generating hard negatives, corroborating its significant impact on categories with a limited number of samples. A high-level sketch of `GeNIe` is illustrated in Fig. 6.1. Our main contributions are summarized below:

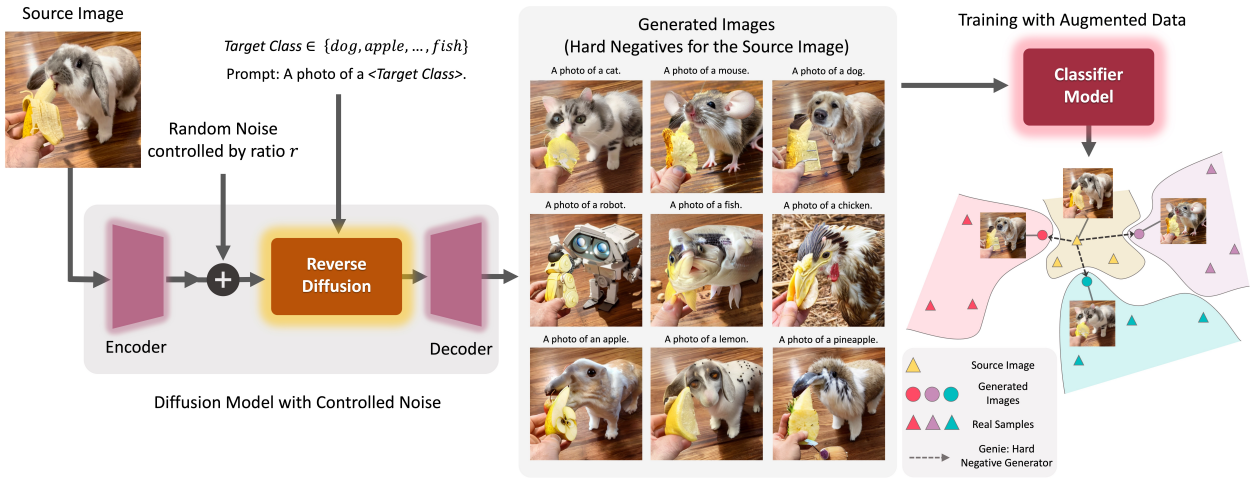


FIGURE 6.1. **Generative Hard Negative Images Through Diffusion (GeNIe):** generates hard negative images that belong to the target category but are similar to the source image from low-level feature and contextual perspectives. GeNIe starts from a source image passing it through a partial noise addition process, and conditioning it on a different target category. By controlling the amount of noise, the reverse latent diffusion process generates images that serve as *hard negatives* for the source category.

- We introduce GeNIe, a novel yet elegantly simple diffusion-based augmentation method to create challenging augmentations in the manifold of natural images. For the first time, to our best knowledge, GeNIe achieves this by combining two sources of information (a source image, and a contradictory target prompt) through a noise-level adjustment mechanism.
- We further extend GeNIe by automating the noise-level adjustment strategy on a per-sample basis (called GeNIe-Ada), to enable generating hard negative samples in the context of image classification, leading also to further performance enhancement.
- To substantiate the impact of GeNIe, we present a suit of quantitative and qualitative results including extensive experimentation on two challenging tasks: few-shot and long tail distribution settings corroborating that GeNIe (and its extension GeNIe-Ada) significantly improve the downstream classification performance.

6.1.2 Proposed Method: GeNIe

Given a source image X_S from category S = <source category>, we are interested in generating a target image X_r from category T = <target category>. In doing so, we intend to ensure the low-level visual features or background context of the source image are preserved, so that we generate samples that would serve as *hard negatives* for the *source* image. To this aim, we adopt a conditional latent diffusion

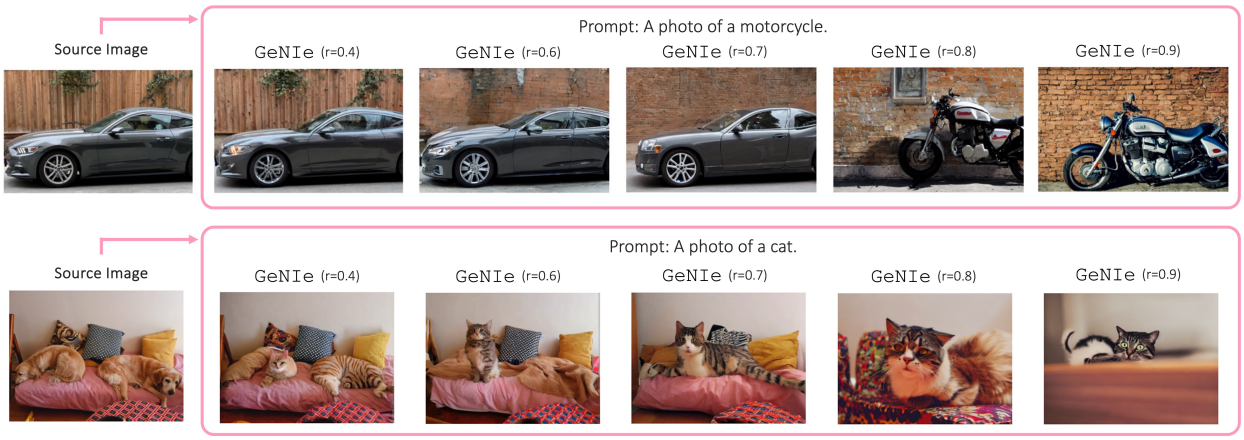


FIGURE 6.2. **Effect of noise ratio, r, in GeNIe:** we employ GeNIe to generate augmentations for the target classes (motorcycle and cat) with varying r. Smaller r yields images closely resembling the source semantics, creating an inconsistency with the intended target label. By tracing r from 0 to 1, augmentations gradually transition from source image characteristics to the target category. However, a distinct shift from the source to the target occurs at a specific r that may vary for different source images or target categories.

model (such as Stable Diffusion, [372]) conditioned on a text prompt of the following format "A photo of a T = <target category>".

Key Idea. GeNIe in its basic form is a simple yet effective augmentation sample generator for improving a classifier $f_\theta(.)$ with the following two key aspects: (i) inspired by [297, 309] instead of adding the full amount of noise σ_{max} and going through all N_{max} (being typically 50) steps of denoising, we use less amount of noise ($r\sigma_{max}$, with $r \in (0,1)$) and consequently fewer number of denoising iterations ($\lfloor rN_{max} \rfloor$); (ii) we prompt the diffusion model with a P mandating a target category T different than the source S. Hence, we denote the conditional diffusion process as $X_r = \texttt{STDiff}(X_S, P, r)$. In such a construct, the proximity of the final decoded image X_r to the source image X_S or the target category defined through the text prompt P depends on r. Hence, by controlling the amount of noise, we can generate images that blend characteristics of both the text prompt P and the source image X_S. If we do not provide much of visual details in the text prompt (e.g., desired background, etc.), we expect the decoded image X_r to follow the details of X_S while reflecting the semantics of the text prompt P. We argue, and demonstrate later, that the newly generated samples can serve as *hard negative* examples for the source category S since they share the low-level features of X_S while representing the semantics of the target category, T. Notably, the source category S can be randomly sampled or be carefully extracted from the confusion matrix of $f_\theta(.)$ based on real training data. The latter might result in even *harder negative* samples being now cognizant of model confusions. Finally, we will append our initial dataset with the newly generated hard negative samples through GeNIe and (re)train the classifier model.

Enhancing `GeNIe`: `GeNIe-Ada`. One of the remarkable aspects of `GeNIe` lies in its simple application, requiring only X_S, P, and r. However, selecting the appropriate value for r poses a challenge as it profoundly influences the outcome. When r is small, the resulting X_r tends to closely resemble X_S, and conversely, when r is large (closer to 1), it tends to resemble the semantics of the target category. This phenomenon arises because a smaller noise level restricts the capacity of the diffusion model to deviate from the semantics of the input X_S. Thus, a critical question emerges: how can we select r for a particular source image to generate samples that preserve the low-level semantics of the source category S in X_S while effectively representing the semantics of the target category T? We propose a method to determine an ideal value for r.

Our intuition suggests that by varying the noise ratio r from 0 to 1, X_r will progressively resemble category S in the beginning and category T towards the end. However, somewhere between 0 and 1, X_r will undergo a rapid transition from category S to T. This phenomenon is empirically observed in our experiments with varying r, as depicted in Fig. 6.2. Although the exact reason for this rapid change remains uncertain, one possible explanation is that the intermediate points between two categories reside far from the natural image manifold, thus, challenging the diffusion model's capability to generate them. Ideally, we should select r corresponding to just after this rapid semantic transition, as at this point, X_r exhibits the highest similarity to the source image while belonging to the target category.

We propose to trace the semantic trajectory between X_S and X_T through the lens of the classifier $f_\theta(.)$. As shown in Algorithm 2, assuming access to the classifier backbone $f_\theta(.)$ and at least one example X_T from the target category, we convert both X_S and X_T into their respective latent vectors Z_S and Z_T by passing them through $f_\theta(.)$. Then, we sample M values for r uniformly distributed $\in (0, 1)$, generating their corresponding X_r and their latent vectors Z_r for all those r. Subsequently, we calculate $d_r = \frac{(Z_r - Z_S)^T (Z_T - Z_S)}{\|Z_T - Z_S\|_2}$ as the distance between Z_r and Z_S projected onto the vector connecting Z_S and Z_T.

Our hypothesis posits that the rapid semantic transition corresponds to a sharp change in this projected distance. Therefore, we sample n values for r uniformly distributed between 0 and 1, and analyze the variations in d_r. We identify the largest gap in d_r and select the r value just after the gap when increasing r, as detailed in Algorithm 2 and illustrated in Fig. 6.3.

6.1.3 Experiments

Since the impact of augmentation is more pronounced when the training data is limited, we evaluate the impact of `GeNIe` on Few-Shot classification in Section 6.1.3.1, Long-Tailed classification in Section 6.1.3.2, and fine-grained classification in Section 6.1.3.3. For `GeNIe-Ada` in all scenarios, we utilize `GeNIe` to

Algorithm 2: `GeNIe-Ada`

Require: X_S, X_T, $f_\theta(.)$, `STDiff`$(.)$, M
Extract $Z_S \leftarrow f_\theta(X_s)$, $Z_T \leftarrow f_\theta(X_T)$
for $m \in [1, M]$ **do**
  $r \leftarrow \frac{m}{M}$, $Z_r \leftarrow f_\theta($ `STDiff`$(X, P, r))$
  $d_m \leftarrow \frac{(Z_r - Z_S)^T (Z_T - Z_S)}{\|Z_T - Z_S\|_2}$
$m^* \leftarrow \text{argmax}_m |d_m - d_{m-1}|, \forall m \in [2, M]$
$r^* \leftarrow \frac{m^*}{n}$
Return: X_{r^*} = `STDiff`(X_S, P, r^*)

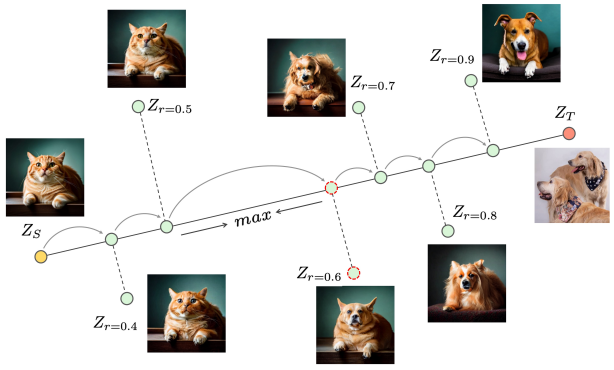


FIGURE 6.3. `GeNIe-Ada`: To choose r adaptively for each (source image, target category) pair, we propose tracing the semantic trajectory from Z_S (source image embeddings) to Z_T (target embeddings) through the lens of the classifier $f_\theta(\cdot)$ (Algorithm 2). We adaptively select the sample right after the largest semantic shift.

generate augmentations from the noise level set $\{0.5, 0.6, 0.7, 0.8, 0.9\}$. The selection of the appropriate noise level per source image and target is adaptive, achieved through Algorithm 2.

Baselines. We use Stable Diffusion 1.5 [372] as our base diffusion model. In all settings, we use the same prompt format to generate images for the target class: i.e., "A photo of a <`target category`>", where we replace the `target category` with the target category label. We generate 512 × 512 images for all methods. For fairness in comparison, we generate the same number of new images for each class. We use a single NVIDIA RTX 3090 for image generation. We consider 4 diffusion-based baselines and a suite of traditional data augmentation baselines:

`Img2Img` [297, 309]: We sample an image from a target class, add noise to its latent representation and then pass it along with a prompt for the target category through reverse diffusion. The focus here is on a target class for which we generate extra positive samples. Adding large amount of noise leads to generating an image less similar to the original image. We use two different noise magnitudes for this baseline: $r = 0.3$ and $r = 0.7$ and denote them by `Img2Img`^{L} and `Img2Img`^{H}, respectively.

`Txt2Img` [28, 180]: For this baseline, we omit the forward diffusion process and only use the reverse process starting from a text prompt for the target class of interest. This is similar to the base text-to-image generation strategy adopted in [28, 180, 296, 372, 393]. Fig. 6.4 illustrates a set of generated augmentation examples for `Txt2Img`, `Img2Img`, and `GeNIe`.

DAFusion [438]: In this method, an embedding is optimized with a set of images for each class to correspond to the classes in the dataset. This approach is introduced in Textual Inversion [143]. We optimize an embedding for 5000 iterations for each class in the dataset, followed by augmentation similar as the DAFusion method.

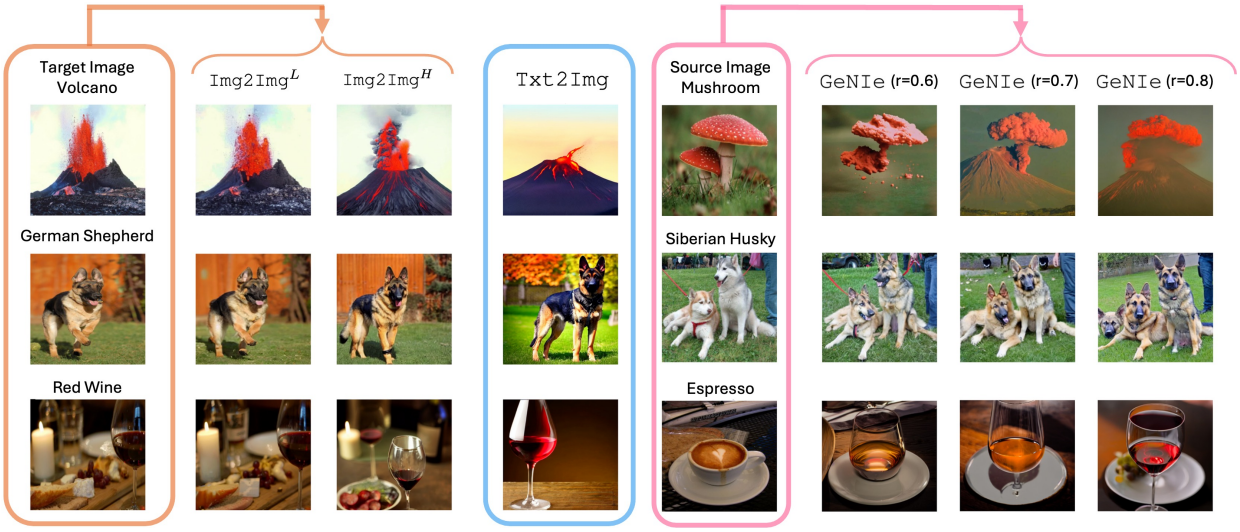


FIGURE 6.4. **Visualization of Generative Samples:** We compare GeNIe with two baselines: **Img2Img^L augmentation:** both image and text prompt are from the same category. Adding noise does not change the image much, so they are not hard examples. **Txt2Img augmentation:** We simply use the text prompt only to generate an image for the desired category (e.g., using a text2image method). Such images may be far from the domain of our task since the generation is not informed by any visual data from our task. **GeNIe augmentation:** We use the target class name in the text prompt only along with the source image.

Cap2Aug [375]: It is a recent diffusion-based data augmentation strategy that uses image captions as text prompts for an image-to-image diffusion model.

Traditional Data Augmentation: We consider both weak and strong traditional augmentations. More specifically, for weak augmentation we use random resize crop with scaling $\in [0.2, 1.0]$ and horizontal flipping. For strong augmentation, we consider random color jitter, random grayscale, and Gaussian blur. For the sake of completeness, we also compare against data augmentations such as CutMix [513] and MixUp [521] that combine two images together.

6.1.3.1 Few-shot Classification

We assess the impact of GeNIe compared to other augmentations in a number of few-shot classification (FSL) scenarios, where the model has to learn only from the samples contained in the (N-way, K-shot) support set and infer on the query set. Note that this corresponds to an inference-only FSL setting where a pretraining stage on an abundant dataset is discarded. The goal is to assess how well the model can benefit from the augmentations while keeping the original $N \times K$ samples intact.

Datasets. We conduct our few-shot experiments on two most commonly adopted few-shot classification datasets: *mini*-Imagenet [365] and *tiered*-Imagenet [369]. *mini*-Imagenet is a subset of ImageNet [101] for few-shot classification. It contains 100 classes with 600 samples each. We follow the predominantly adopted

TABLE 6.1. ***mini*-ImageNet:** We use our augmentations on (5-way, 1-shot) and (5-way, 5-shot) few-shot settings of mini-Imagenet dataset with 3 different backbones (ResNet-18, 34, and 50). We compare with various baselines and show that our augmentations with UniSiam outperform all the baselines including `Txt2Img` and DAFusion augmentation. The number of generated images per class is 4 for 1-shot and 20 for 5-shot settings.

ResNet-18				
Augmentation	**Method**	**Pre-training**	**1-shot**	**5-shot**
-	iDeMe-Net [78]	sup.	59.1±0.9	74.6±0.7
-	Robust + dist [118]	sup.	63.7±0.6	81.2±0.4
-	AFHN [266]	sup.	62.4±0.7	78.2±0.6
Weak	ProtoNet+SSL [406]	sup.+ssl	-	76.6
Weak	Neg-Cosine [282]	sup.	62.3±0.8	80.9±0.6
-	Centroid Align [18]	sup.	59.9±0.7	80.4±0.7
-	Baseline [75]	sup.	59.6±0.8	77.3±0.6
-	Baseline++ [75]	sup.	59.0±0.8	76.7±0.6
Weak	PSST [79]	sup.+ssl	59.5±0.5	77.4±0.5
Weak	UMTRA [229]	unsup.	43.1±0.4	53.4±0.3
Weak	ProtoCLR [307]	unsup.	50.9±0.4	71.6±0.3
Weak	SimCLR [71]	unsup.	62.6±0.4	79.7±0.3
Weak	SimSiam [77]	unsup.	62.8±0.4	79.9±0.3
Weak	UniSiam+dist [295]	unsup.	**64.1±0.4**	**82.3±0.3**
Weak	UniSiam [295]	unsup.	63.1±0.8	81.4±0.5
Strong	UniSiam [295]	unsup.	62.8±0.8	81.2±0.6
CutMix [513]	UniSiam [295]	unsup.	62.7±0.8	80.6±0.6
MixUp [521]	UniSiam [295]	unsup.	62.1±0.8	80.7±0.6
`Img2Img`^{L} [297]	UniSiam [295]	unsup.	63.9±0.8	82.1±0.5
`Img2Img`^{H} [297]	UniSiam [295]	unsup.	69.1±0.7	84.0±0.5
`Txt2Img` [28, 180]	UniSiam [295]	unsup.	74.1±0.6	84.6±0.5
DAFusion [438]	UniSiam [295]	unsup.	64.3±1.8	82.0±1.4
`GeNIe` (Ours)	UniSiam [295]	unsup.	**75.5±0.6**	**85.4±0.4**
`GeNIe-Ada` (Ours)	UniSiam [295]	unsup.	**76.8±0.6**	**85.9±0.4**

ResNet-34				
Augmentation	**Method**	**Pre-training**	**1-shot**	**5-shot**
Weak	Baseline [75]	sup.	49.8±0.7	73.5±0.7
Weak	Baseline++ [75]	sup.	52.7±0.8	76.2±0.6
Weak	SimCLR [71]	unsup.	64.0±0.4	79.8±0.3
Weak	SimSiam [77]	unsup.	63.8±0.4	80.4±0.3
Weak	UniSiam+dist [295]	unsup.	**65.6±0.4**	**83.4±0.2**
Weak	UniSiam [295]	unsup.	64.3±0.8	82.3±0.5
Strong	UniSiam [295]	unsup.	64.5±0.8	82.1±0.6
CutMix [513]	UniSiam [295]	unsup.	64.0±0.8	81.7±0.6
MixUp [521]	UniSiam [295]	unsup.	63.7±0.8	80.1±0.8
`Img2Img`^{L} [297]	UniSiam [295]	unsup.	65.5±0.8	82.9±0.5
`Img2Img`^{H} [297]	UniSiam [295]	unsup.	70.5±0.8	84.8±0.5
`Txt2Img` [28, 180]	UniSiam [295]	unsup.	75.4±0.6	85.5±0.5
DAFusion [438]	UniSiam [295]	unsup.	64.7±1.9	83.2±1.4
`GeNIe` (Ours)	UniSiam [295]	unsup.	**77.1±0.6**	**86.3±0.4**
`GeNIe-Ada` (Ours)	UniSiam [295]	unsup.	**78.5±0.6**	**86.6±0.4**
ResNet-50				
Weak	PDA+Net [73]	unsup.	63.8±0.9	83.1±0.6
Weak	Meta-DM [199]	unsup.	66.7±0.4	85.3±0.2
Weak	UniSiam [295]	unsup.	64.6±0.8	83.4±0.5
Strong	UniSiam [295]	unsup.	64.8±0.8	83.2±0.5
CutMix [513]	UniSiam [295]	unsup.	64.3±0.8	83.2±0.5
MixUp [521]	UniSiam [295]	unsup.	63.8±0.8	84.6±0.5
`Img2Img`^{L} [297]	UniSiam [295]	unsup.	66.0±0.8	84.0±0.5
`Img2Img`^{H} [297]	UniSiam [295]	unsup.	71.1±0.7	85.7±0.5
`Txt2Img` [28, 180]	UniSiam [295]	unsup.	76.4±0.6	86.5±0.4
DAFusion [438]	UniSiam [295]	unsup.	65.7±1.8	83.9±1.2
`GeNIe` (Ours)	UniSiam [295]	unsup.	**77.3±0.6**	**87.2±0.4**
`GeNIe-Ada` (Ours)	UniSiam [295]	unsup.	**78.6±0.6**	**87.9±0.4**

settings of [75, 365] where we split the entire dataset into 64 classes for training, 16 for validation and 20 for testing. *tiered*-Imagenet is a larger subset of ImageNet with 608 classes and a total of 779, 165 images, which are grouped into 34 higher-level nodes in the *ImageNet* human-curated hierarchy. This set of nodes is partitioned into 20, 6, and 8 disjoint sets of training, validation, and testing nodes, and the corresponding classes form the respective meta-sets.

Evaluation. To quantify the impact of different augmentation methods, we evaluate the test-set accuracies of a state-of-the-art unsupervised few-shot learning method with `GeNIe` and compare them against the accuracies obtained using other augmentation methods. Specifically, we use UniSiam [295] pre-trained with ResNet-18, ResNet-34 and ResNet-50 backbones and follow its evaluation strategy of fine-tuning a logistic regressor to perform (N-way, K-shot) classification on the test sets of *mini*- and *tiered*-Imagenet. Following [365], an episode consists of a labeled support-set and an unlabelled query-set. The support-set contains N randomly sampled classes where each class contains K samples, whereas the query-set contains Q randomly sampled unlabeled images per class. We conduct our experiments on the two most commonly adopted settings: (5-way, 1-shot) and (5-way, 5-shot) classification settings. Following the literature, we sample 16-shots per class for the query set in both settings. We report the test accuracies along with the 95% confidence interval over 600 and 1000 episodes for *mini*-ImageNet and *tiered*-ImageNet, respectively.

TABLE 6.2. ***tiered*-ImageNet:** Accuracies (% ± std) for 5-way, 1-shot and 5-way, 5-shot classification settings on the test-set. We compare against various SOTA supervised and unsupervised few-shot classification baselines as well as other augmentation methods, with UniSiam [295] pre-trained ResNet-18,50 backbones.

Augmentation	Method	Pre-training	1-shot	5-shot
ResNet-18				
Weak	SimCLR [71]	unsup.	63.4±0.4	79.2±0.3
Weak	SimSiam [77]	unsup.	64.1±0.4	81.4±0.3
Weak	UniSiam [295]	unsup.	63.1±0.7	81.0±0.5
Strong	UniSiam [295]	unsup.	62.8±0.7	80.9±0.5
CutMix [513]	UniSiam [295]	unsup.	62.1±0.7	78.9±0.6
MixUp [521]	UniSiam [295]	unsup.	62.1±0.7	78.4±0.6
Img2Img^L [297]	UniSiam [295]	unsup.	63.9±0.7	81.8±0.5
Img2Img^H [297]	UniSiam [295]	unsup.	68.7±0.7	83.5±0.5
Txt2Img [180]	UniSiam [295]	unsup.	72.9±0.6	84.2±0.5
DAFusion [438]	UniSiam [295]	unsup.	62.6±2.1	81.0±1.5
GeNIe(Ours)	UniSiam [295]	unsup.	**73.6±0.6**	**85.0±0.4**
GeNIe-Ada(Ours)	UniSiam [295]	unsup.	**75.1±0.6**	**85.5±0.5**
ResNet-50				
Weak	PDA+Net [73]	unsup.	69.0±0.9	84.2±0.7
Weak	Meta-DM [199]	unsup.	69.6±0.4	86.5±0.3
Weak	UniSiam + dist [295]	unsup.	69.6±0.4	86.5±0.3
Weak	UniSiam [295]	unsup.	66.8±0.7	84.7±0.5
Strong	UniSiam [295]	unsup.	66.5±0.7	84.5±0.5
CutMix [513]	UniSiam [295]	unsup.	66.0±0.7	83.3±0.5
MixUp [521]	UniSiam [295]	unsup.	66.1±0.5	84.1±0.8
Img2Img^L [297]	UniSiam [295]	unsup.	67.8±0.7	85.3±0.5
Img2Img^H [297]	UniSiam [295]	unsup.	72.4±0.7	86.7±0.4
Txt2Img [180]	UniSiam [295]	unsup.	77.1±0.6	87.3±0.4
DAFusion [438]	UniSiam [295]	unsup.	66.5±2.2	84.8±1.4
GeNIe (Ours)	UniSiam [295]	unsup.	**78.0±0.6**	**88.0±0.4**
GeNIe-Ada (Ours)	UniSiam [295]	unsup.	**78.8±0.6**	**88.6±0.6**

TABLE 6.3. **Long-Tailed ImageNet-LT:** We compare different augmentation methods on ImageNet-LT and report Top-1 accuracy for "Few", "Medium", and "Many" sets. On the "Few" set and LiVT method, our augmentations improve the accuracy by 11.7 points compared to LiVT original augmentation and 4.4 points compared to Txt2Img. GeNIe-Ada outperforms Cap2Aug baseline in "Few" categories by 7.6%.

Method	Many	Med.	Few	Overall Acc
ResNet-50				
ResLT [97]	63.3	53.3	40.3	55.1
PaCo [98]	68.2	58.7	41.0	60.0
LWS [222]	62.2	48.6	31.8	51.5
Zero-shot CLIP [352]	60.8	59.3	58.6	59.8
DRO-LT [386]	64.0	49.8	33.1	53.5
VL-LTR [423]	77.8	67.0	50.8	70.1
Cap2Aug [375]	78.5	**67.7**	51.9	70.9
GeNIe-Ada	**79.2**	64.6	**59.5**	**71.5**
ViT-B				
ViT [111]	50.5	23.5	6.9	31.6
MAE [173]	74.7	48.2	19.4	54.5
DeiT [434]	70.4	40.9	12.8	48.4
LiVT [493]	73.6	56.4	41.0	60.9
LiVT + Img2Img^L	74.3	56.4	34.3	60.5
LiVT + Img2Img^H	73.8	56.4	45.3	61.6
LiVT + Txt2Img	**74.9**	55.6	48.3	62.2
LiVT + GeNIe-Ada	74.0	**56.9**	**52.7**	**63.1**

Implementation Details: GeNIe generates augmented images for each class using images from all other classes as the source image. We use $r = 0.8$ in our experiments. We generate 4 samples per class as augmentations in the 5-way, 1-shot setting and 20 samples per class as augmentations in the 5-way, 5-shot setting. For the sake of a fair comparison, we ensure that the total number of labelled samples in the support set after augmentation remains the same across all different traditional and generative augmentation methodologies. Due to the expensive training of embeddings for each class in each episode, we only evaluated the DA-Fusion baseline on the first 100 episodes.

Results: The results on *mini*-Imagenet and *tiered*-Imagenet for both (5-way, 1 and 5-shot) settings are summarized in Table 6.1 and Table 6.2, respectively. Regardless of the choice of backbone, we observe that GeNIe helps consistently improve UniSiam's performance and outperform other supervised and unsupervised few-shot classification methods as well as other diffusion-based [180, 297, 371, 438] and classical [513, 521] data augmentation techniques on both datasets, across both (5-way, 1 and 5-shot) settings. Our noise adaptive method of selecting optimal augmentations per source image (GeNIe-Ada) further improves GeNIe's performance across all three backbones, both few-shot settings, and both datasets (*mini* and

tiered-Imagenet). Note that employing CutMix and MixUp seems to lead to performance degradation compared to weak augmentations, probably due to overfitting since these methods can only choose from 4 other classes to mix.

6.1.3.2 Long-Tailed Classification

We evaluate our method on long-tailed data, where the number of instances per class is unbalanced, with most categories having limited samples (tail). Our goal is to mitigate this bias by augmenting the tail of the distribution with generated samples. We evaluate `GeNIe` using two different backbones and methods: the ViT architecture with LViT [493], and ResNet50 with VL-LTR [423].

Following LViT [493], we first train an MAE [174] and ViT on the unbalanced dataset without any augmentation. Next, we train the Balanced Fine-Tuning stage of LViT by incorporating the augmentation data generated using `GeNIe` or other baselines. For ResNet50, we use VL-LTR code to fine-tune the CLIP [352] ResNet50 pretrained backbone with generated augmentations by `GeNIe`.

Dataset: We perform experiments on ImageNet-LT [291]. It contains 115.8K images from 1,000 categories. The number of images per class varies from 1280 to 5. Imagenet-LT classes can be divided into 3 groups: "Few" with less than 20 images, "Med" with 20 – 100 images, and "Many" with more than 100 images. Imagenet-LT uses the same validation set as ImageNet. We augment "Few" categories only and limit the number of generated images to 50 samples per class. For `GeNIe`, instead of randomly sampling the source images from other classes, we use a confusion matrix on the training data to find the top-4 most confused classes and only consider those classes for random sampling of the source image. The source category may be from "Many", "Med", or "Few sets".

Results: Augmenting training data with `GeNIe-Ada` improves accuracy on the "Few" set by 11.7% and 4.4% compared with LViT only and LViT with `Txt2Img` augmentation baselines respectively. In ResNet50, `GeNIe-Ada` outperforms Cap2Aug baseline in "Few" categories by 7.6%. The results are in Table 6.3.

Implementation Details of LViT: We download the pre-trained ViT-B of LViT [493] and finetune it with Bal-BCE loss proposed therein on the augmented dataset. Training takes 2 hours on four NVIDIA RTX 3090 GPUs. We use the same hyperparameters as in [493] for finetuning: 100 epochs, lr = 0.008, batch size of 1024, CutMix and MixUp for the data augmentation.

Implementation Details of VL-LTR: We use the official code of VL-LTR [423] for our experiments. We use a pre-trained CLIP ResNet-50 backbone. We followed the hyperparameters reported in VL-LTR [423]. We augment only "Few" category and train the backbone with the VL-LTR [423] method. Training takes 4 hours on 8 NVIDIA RTX 3090 GPUs.

TABLE 6.4. **Few-shot Learning on Fine-grained dataset:** We utilize an SVM classifier trained atop the DINOV2 ViT-G pretrained backbone, reporting Top-1 accuracy for the test set of each dataset. The baseline is an SVM trained on the same backbone using weak augmentation. Across all datasets, `GeNIe` surpasses this baseline.

Method	Birds CUB200 [452]	Cars Cars196 [244]	Foods Food101 [41]	Aircraft Aircraft [301]
Baseline	90.3	49.8	82.9	29.2
`Img2Img`^{L} [297]	90.7	50.4	87.4	31.0
`Img2Img`^{H} [297]	91.3	56.4	91.7	34.7
`Txt2Img` [180]	92.0	81.3	93.0	41.7
`GeNIe` (r=0.5)	92.0	84.6	91.5	39.8
`GeNIe` (r=0.6)	92.2	87.1	92.5	45.0
`GeNIe` (r=0.7)	92.5	**87.9**	92.9	**47.0**
`GeNIe` (r=0.8)	92.5	87.7	**93.1**	46.5
`GeNIe` (r=0.9)	92.4	87.1	**93.1**	45.7
`GeNIe-Ada`	**92.6**	**87.9**	**93.1**	46.9

6.1.3.3 Fine-grained Few-shot Classification

To further investigate the impact of the proposed method, we compare `GeNIe` with other text-based data augmentation techniques across four distinct fine-grained datasets in a 20-way, 1-shot classification setting. We employ the pre-trained DINOV2 ViT-G [327] backbone as a feature extractor to derive features from training images. Subsequently, an SVM classifier is trained on these features, and we report the Top-1 accuracy of the model on the test set.

Datasets: We assess our method on several datasets: Food101 [41] with 101 classes of various foods, CUB200 [452] with 200 bird species classes, Cars196 [244] with 196 car model classes, and FGVC-Aircraft [301] with 41 aircraft manufacturer classes. The reported metric is the average Top-1 accuracy over 100 episodes. Each episode involves sampling 20 classes and 1-shot from the training set, with the final model evaluated on the respective test set.

Implementation Details: We enhance the basic prompt by incorporating the superclass name for the fine-grained dataset: "A photo of a `<target class>`, a type of `<superclass>`". For instance, in the *food* dataset and the *burger* class, our prompt reads: "A photo of a *burger*, a type of *food*." No additional augmentation is used for generative methods in this context. We generate 19 samples for both cases of our method and also the baseline with weak augmentation.

Results: Table 6.4 summarizes the results. `GeNIe` helps outperform all other baselines and augmentations, including `Txt2Img`, by margins upto 0.5% on CUB200 [452], 6.6% on Cars196 [244], 0.1% on Food101 [41] and 5.3% on FGVC-Aircraft [301]. Notably, `GeNIe` exhibits great effectiveness in more challenging datasets, outperforming the baseline with traditional augmentation by about 38% for the Cars dataset

and by roughly 17% for the Aircraft dataset. It can be observed here that `GeNIe-Ada` performs on-par with `GeNIe` with a fixed noise level, eliminating the necessity for noise level search in `GeNIe`.

6.1.3.4 Ablation and Analysis

Semantic Shift from Source to Target Class. The core motivation behind `GeNIe-Ada` is that by varying the noise ratio r from 0 to 1, augmented sample X_r will progressively shift its semantic category from source (S) in the beginning to target category (T) towards the end. However, somewhere between 0 and 1, X_r will undergo a rapid transition from S to T. To demonstrate this hypothesis empirically, in Figs. 6.5, we visualize pairs of source images and target categories with their respective `GeNIe` generated augmentations for different noise ratios r, along with their corresponding PCA-projected embedding scatter plots (on the far left). We extract embeddings for all the images using a DINOv2 ViT-G pretrained backbone, which we assume as an oracle model in identifying the right category. We observe that as r increases from 0.3 to 0.8, the images transition to embody more of the target category's semantics while preserving the contextual features of the source image. This transition of semantics can also be observed in the embedding plots (on the left) where they consistently shift from the proximity of the source image (blue star) to the target class's centroid (red cross) as the noise ratio r increases. The sparse distribution of points within $r = [0.4, 0.6]$ for the first image and $r = [0.2, 0.4]$ for the second image aligns with our intuition of a rapid transition from category S to T, thus empirically affirming our motivation behind `GeNIe-Ada`.

To further establish this, in Fig. 6.6, we demonstrate the efficacy of `GeNIe` in generating hard negatives at the decision boundaries of an SVM classifier, which is trained on the labelled support set of the few-shot tasks of *mini*-Imagenet, without any augmentations. We then plot source and target class probabilities ($P(Y_S|X_r)$ and $P(Y_T|X_r)$, respectively) of the generated augmentation samples X_r. For both $r = 0.6$ and 0.7, there is significant overlap between $P(Y_S|X_r)$ and $P(Y_T|X_r)$, making it difficult for the classifier to decide the correct class. On the right-hand-side, `GeNIe-Ada` automatically selects the best r resulting in the most overlap between the two distributions, thus offering the hardest negative sample among the considered r values. Note that a large overlap between distributions is not sufficient to call the generated samples hard negatives because they should also belong to the target category. This is, however, confirmed by the high Oracle accuracy in Table 6.5 (elaborated in detail in the following paragraph) which verifies that majority of the generated augmentation samples do belong to the target category.

Label consistency of the generated samples. The choice of noise ratio r is important in producing hard negative examples. In Table 6.5, we present the accuracy of the `GeNIe` model across various noise ratios,

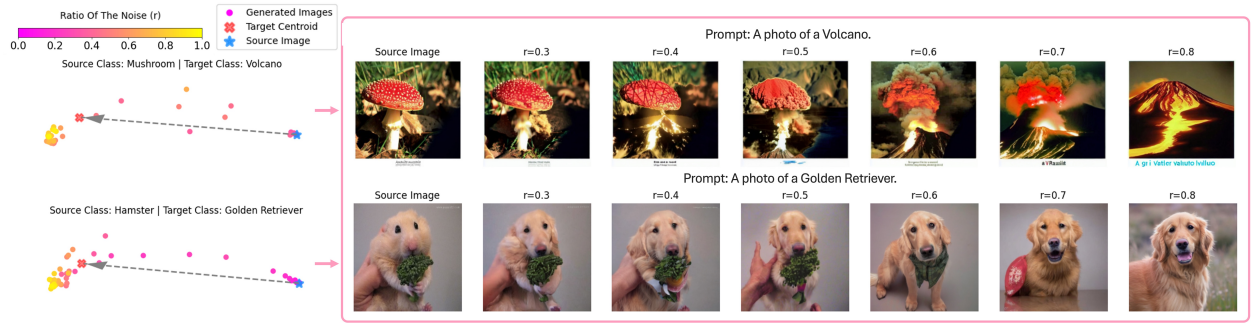


FIGURE 6.5. **Embedding visualizations of generative augmentations:** We pass all generative augmentations through DINOv2 ViT-G (serving as an oracle) to extract their corresponding embeddings and visualize them with PCA. As shown, the extent of semantic shifts varies based on both the source image and the target class.

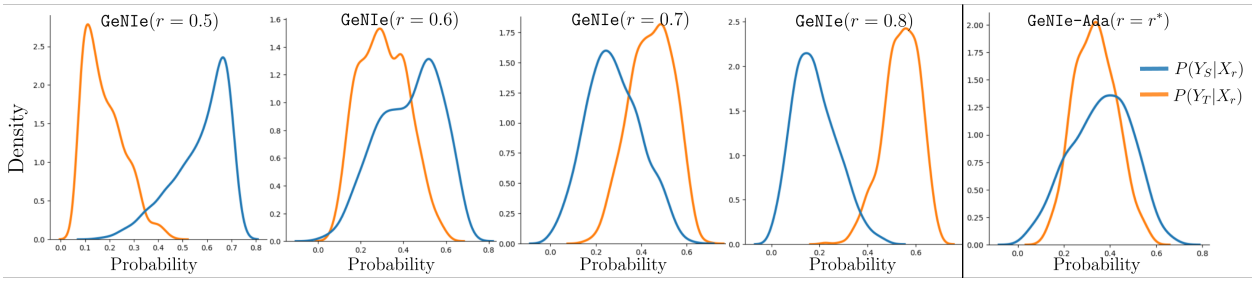


FIGURE 6.6. **Why GeNIe augmentations are challenging?** While deciding which class the generated augmentations (X_r) belong to is already difficult within $r = [0.6, 0.7]$ (due to high overlap between $P(Y_S|X_r)$ and $P(Y_T|X_r)$), GeNIe-Ada selects the best noise threshold (r^*) offering the hardest negative sample.

alongside the oracle accuracy, which is an ImageNet pre-trained DeiT-Base [433] classifier. We observe a decline in the label consistency of generated data (quantified by the performance of the oracle model) when decreasing the noise level. Reducing r also results in a degradation in the performance of the final few-shot model (87.2% → 77.6%) corroborating that an appropriate choice of r plays a crucial role in our design strategy. We investigate this further in the following paragraph.

Effect of Noise in GeNIe. We examine the impact of noise on the performance of the few-shot model in Table 6.5. Noise levels $r \in [0.7, 0.8]$ yield the best performance. Conversely, utilizing noise levels below 0.7 diminishes performance due to label inconsistency, as is demonstrated in Table 6.5 and Fig 6.5. As such, determining the appropriate noise level is pivotal for the performance of GeNIe to be able to generate challenging hard negatives while maintaining label consistency. An alternative approach to finding the optimal noise level involves using GeNIe-Ada to adaptively select the noise level for each source image and target class. As demonstrated in Tables 6.5 and 6.4, GeNIe-Ada achieves performance that is comparable to or surpasses that of GeNIe with fixed noise levels.

TABLE 6.5. **Effect of Noise and Diffusion Models in GeNIe:** We use the same setting as in Table 6.1 to study the effect of the amount of noise. As expected (also shown in Fig 6.5), small noise results in worse accuracy since some generated images may be from the source category rather than the target one. For $r = 0.5$ only 73% of the generated data is from the target category. This behavior is also shown in Fig. 6.2. Notably, reducing the noise level below 0.7 is associated with a decline in oracle accuracy and subsequent degradation in the performance of the final few-shot model. Note that the high oracle accuracy of GeNIe-Ada demonstrates its capability to adaptively select the noise level per source and target, ensuring semantic consistency with the intended target. To further demonstrate GeNIe's ability to generalize across different diffusion models, we replace the diffusion model with SD3 and SDXL-Turbo. The resulting accuracies follow a similar trend to those in Table 6.1, confirming GeNIe's advantage over Txt2Img across various diffusion models.

Method	Generative Model	Noise r=	ResNet-18		ResNet-34		ResNet-50		Oracle Acc
			1-shot	5-shot	1-shot	5-shot	1-shot	5-shot	
Txt2Img	SD 1.5	-	74.1±0.6	84.6±0.5	75.4±0.6	85.5±0.5	76.4±0.6	86.5±0.4	-
GeNIe	SD 1.5	0.5	60.4±0.8	74.1±0.6	62.0±0.8	75.8±0.6	63.7±0.9	77.6±0.6	73.4±0.5
GeNIe	SD 1.5	0.6	69.7±0.7	80.7±0.5	71.1±0.7	82.2±0.5	72.1±0.7	82.8±0.5	85.8±0.4
GeNIe	SD 1.5	0.7	74.5±0.6	83.3±0.5	76.4±0.6	84.4±0.5	77.1±0.6	85.0±0.4	94.5±0.2
GeNIe	SD 1.5	0.8	75.5±0.6	85.4±0.4	77.1±0.6	86.3±0.4	77.3±0.6	87.2±0.4	98.2±0.1
GeNIe	SD 1.5	0.9	75.0±0.6	85.3±0.4	77.6±0.6	86.2±0.4	77.7±0.6	87.0±0.4	99.3±0.1
GeNIe-Ada	SD 1.5	Adaptive	76.8±0.6	85.9±0.4	78.5±0.6	86.6±0.4	78.6±0.6	87.9±0.4	98.9±0.2
Txt2Img	SDXL-Turbo	-	72.5±0.3	82.1±0.6	76.2±0.2	84.4±0.3	76.7±0.6	85.9±0.5	-
GeNIe	SDXL-Turbo	0.5	61.2±0.5	73.5±0.2	61.5±0.2	74.9±0.3	63.1±0.2	76.5±0.6	-
GeNIe	SDXL-Turbo	0.6	70.2±0.2	79.3±0.4	71.2±0.7	81.4±0.6	73.2±0.2	82.4±0.5	-
GeNIe	SDXL-Turbo	0.7	73.1±0.3	83.5±0.5	76.1±0.6	85.3±0.4	77.2±0.6	84.2±0.4	-
GeNIe	SDXL-Turbo	0.8	74.2±0.3	85.1±0.3	76.9±0.4	85.5±0.5	78.7±0.6	87.7±0.4	-
GeNIe	SDXL-Turbo	0.9	73.9±0.4	84.9±0.7	76.6±0.7	84.2±0.6	78.1±0.5	87.0±0.4	-
GeNIe-Ada	SDXL-Turbo	Adaptive	75.1±0.3	87.1±0.8	78.9±0.5	85.2±0.5	79.0±0.6	88.6±0.2	-
Txt2Img	SD 3	-	73.6±1.7	82.9±1.2	76.7±1.5	85.5±1.3	77.2±1.9	85.0±1.2	-
GeNIe	SD 3	0.5	62.0±1.2	72.9±1.1	62.5±0.9	73.9±1.0	64.1±0.5	76.1±1.9	-
GeNIe	SD 3	0.6	70.8±1.5	79.1±1.9	71.8±1.2	82.1±1.3	74.1±1.5	83.4±1.8	-
GeNIe	SD 3	0.7	74.6±0.8	84.5±1.2	76.5±1.9	86.2±1.6	78.5±1.9	84.0±1.1	-
GeNIe	SD 3	0.8	75.9±1.2	86.3±1.7	77.8±1.9	85.5±1.9	79.2±1.7	88.3±1.9	-
GeNIe	SD 3	0.9	75.1±0.5	85.2±1.2	78.1±1.3	86.2±1.2	77.1±1.9	88.9±0.8	-
GeNIe-Ada	SD 3	Adaptive	76.8±1.3	87.5±1.5	78.9±1.3	87.7±1.5	79.1±1.4	89.5±1.0	-

Effect of Diffusion Models in GeNIe. We have tried experimenting with both smaller as well as more recent diffusion models. More specifically, we have used Stable Diffusion XL-Turbo to generate hard-negatives through GeNIe and GeNIe-Ada. Few-shot classification results on miniImagenet with these augmentations are shown in Table 6.5. The accuracies follow a similar trend to that of Table 6.1, where Stable Diffusion 1.5 was used to generate augmentations. GeNIe-Ada improves UniSiam's few-shot performance the most as compared to GeNIe with different noise ratios r, and even when compared to Txt2Img. This empirically indicates the robustness of GeNIe and GeNIe-Ada to different diffusion engines. Note that, Stable Diffusion XL-Turbo by default uses 4 steps for the sake of optimization, and to ensure we can have the right granularity for the choice of r we have set the number of steps to 10. That is already 5 times faster than the standard Stable Diffusion v1.5 with 50 steps. Our experiments with Stable Diffusion v3 (which is a totally different model with a Transformers backbone) also in Table 6.5 also convey the same message. As such, we believe our approach is generalizable across different diffusion models.

6.1.4 Related Work

Data Augmentations. Simple flipping, cropping, colour jittering, and blurring are some forms of image augmentations [395]. These augmentations are commonly adopted in training deep learning models. However, using these data augmentations is not trivial in some domains. For example, using blurring might remove important low-level information from medical images. More advanced approaches, such as MixUp [521] and CutMix [513], mix images and their labels accordingly [95, 186, 232, 287]. However, the resulting augmentations are not natural images anymore, and thus, act as out-of-distribution samples that will not be seen at test time. Another strand of research tailors the augmentation strategy through a learning process to fit the training data [92, 96, 107]. Unlike the above methods, we propose to utilize pre-trained latent diffusion models to generate hard negatives (in contrast to generic augmentations) through a noise adaptation strategy discussed in Section 6.1.2.

Data Augmentation with Generative Models. Using synthesized images from generative models to augment training data has been studied before in many domains [138, 387], including domain adaptation [206], visual alignment [339], and mitigation of dataset bias [182, 346, 391]. For example, [346] introduces a methodology aimed at enhancing test set evaluation through augmentation. While previous methods predominantly relied on GANs [262, 442, 530] as the generative model, more recent studies promote using diffusion models to augment the data [28, 61, 117, 134, 180, 212, 296, 372, 375, 393, 438]. More specifically, [28, 180, 375, 438] study the effectiveness of text-to-image diffusion models in data augmentation by diversification of each class with synthetic images. [438] leverages a text-to-image diffusion model and fine-tunes it on the downstream dataset using textual-inversion [144] to increase the diversity of existing samples. [375] also utilizes a text-to-image diffusion model, but with a BLIP [264] model to generate meaningful captions from the existing images. [212] utilizes diffusion models for augmentation to correct model mistakes. [134] uses CLIP [352] to filter generated images. [117] utilizes text-based diffusion and a large language model (LLM) to diversify the training data. [61] uses an LLM to generate text descriptions of failure modes associated with spurious correlations, which are then used to generate synthetic data through generative models. The challenge is the LLM's lack of understanding of failure scenarios and their contexts.

We take a completely different approach here, without replying on any extra source of information (e.g., through an LLM). Inspired by image editing approaches such as Boomerang [297] and SDEdit [309], we propose to adaptively guide a latent diffusion model to generate *hard negatives* images [303, 494] on a per-sample basis per category. In a nutshell, the aforementioned studies focus on improving the diversity of each class with effective prompts and diffusion models, however, we focus on generating effective *hard*

negative samples for each class by combining two sources of contradicting information (images from the source category and text prompt from the target category).

Language Guided Recognition Models. Vision-Language foundation models (VLMs) [22, 352, 356, 357, 372, 383] utilize human language to guide the generation of images or to extract features from images that are aligned with human language. For example, CLIP [352] shows decent zero-shot performance on many downstream tasks by matching images to their text descriptions. Some recent works improve the utilization of human language in the prompt [116, 342], and others use a diffusion model directly as a classifier [259]. Similar to the above, we use a foundation model (Stable Diffusion 1.5 [372]) to improve the downstream task. Concretely, we utilize category names of the downstream tasks to augment their associate training data with hard negative samples.

Few-Shot Learning. In Few-shot Learning (FSL), we pre-train a model with abundant data to learn a rich representation, then fine-tune it on new tasks with only a few available samples. In supervised FSL [18, 75, 118, 266, 348, 399, 410, 499, 542], pretraining is done on a labeled dataset, whereas in unsupervised FSL [24, 196, 213, 229, 295, 307, 349, 394, 455] the pre-training has to be conducted on an unlabeled dataset. We assess the impact of `GeNIe` on a number of few-shot scenarios and state-of-the-art baselines by accentuating on its impact on the few-shot inference stage.

6.1.5 Conclusion

`GeNIe`, for the first time to our knowledge, combines contradictory sources of information (a source image, and a different target category prompt) through a noise adjustment strategy into a conditional latent diffusion model to generate challenging augmentations, which can serve as hard negatives.

Limitation. The required time to create augmentations through `GeNIe` is on par with any typical diffusion-based competitors azizi2023synthetic,he2022synthetic; however, this is naturally slower than traditional augmentation techniques Cutmix,mixup. This is not a bottleneck in offline augmentation strategies, but can be considered a limiting factor in real-time scenarios. Recent studies are already mitigating this through advancements in diffusion model efficiency sauer2023adversarial,meng2023distillation,liu2023instaflow. Another challenge present in any generative AI-based augmentation technique is the domain shift between the distribution of training data and the downstream context they might be used for augmentation. A possible remedy is to fine-tune the diffusion backbone on a rather small dataset from the downstream task.

CHAPTER 7

Conclusion

As discussed earlier, scaling data and compute has been a key strategy for improving the representations learned by deep learning models. Self-supervised learning techniques, such as auto-regressive approaches in language models, have significantly facilitated this scaling by reducing the need for annotated labels. This progress has led to the development of the powerful large language models we use today. However, these models require substantial energy and computational resources, raising concerns about their accessibility.

In this dissertation, we explore the efficiency of deep learning models from various perspectives. Below, we briefly discuss several promising directions to further improve the efficiency of deep learning models in the era of generative AI.

7.1 Energy-Efficiency and Robustness to Energy Adversarial Attacks:

The rapid scaling of large language models (LLMs) has highlighted energy consumption as a critical bottleneck in their development and deployment. Studies such as [337] emphasize the significant costs associated with the energy demands of these models during training and inference. Note that energy usage does not always directly correlate with FLOPs, as memory-intensive operations, such as frequent memory access in models with large embeddings or sparse computation patterns, can increase energy consumption. For instance, a model with lower FLOPs may still be energy-inefficient if its architecture relies heavily on memory bandwidth, resulting in higher energy costs per operation.

To address these challenges, future research must focus on developing energy-efficient architectures. Techniques such as sparse activation [312] and memory-efficient model designs [422] offer promising avenues for reducing energy consumption while maintaining model performance. The growing importance of energy consumption in these models also introduces new risks, such as energy adversarial attacks. As highlighted in Chapter 5.1, these attacks manipulate input patterns to deliberately induce excessive energy usage, posing a particular threat to efficient transformer architectures.

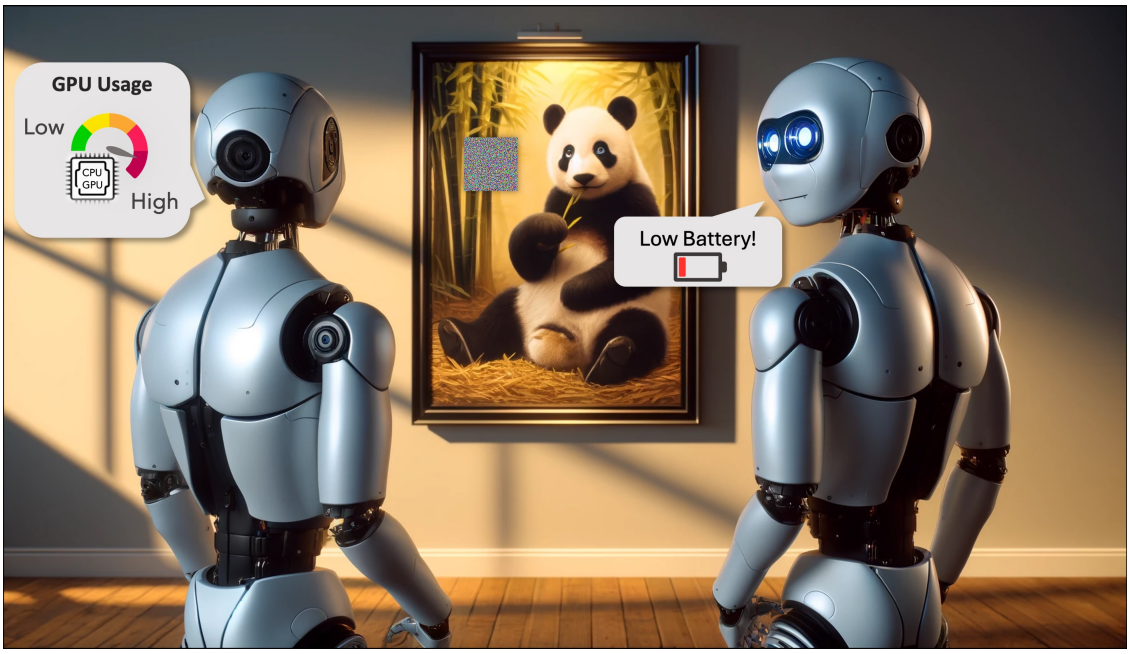


FIGURE 7.1. **Energy Attack:** As discussed in Chapter 5.1, these attacks exploit specific input patterns to intentionally trigger excessive energy and compute consumption, posing a significant threat to the efficiency and reliability of transformer architectures.

To ensure the sustainability and scalability of generative AI, it is crucial to design models that are not only computationally efficient but also resilient to energy-related vulnerabilities, safeguarding their reliability in real-world applications.

7.2 Synthetic Data Generation to Address Model Failures

Deep learning models often struggle with rare scenarios (the long tail of the data distribution) due to insufficient data representing these cases. Collecting data for the long tail is often infeasible or prohibitively expensive. Synthetic data generation provides a scalable solution to address these specific failure modes, such as handling edge cases or underrepresented data. By leveraging generative models to create diverse and controlled datasets, researchers can systematically address model weaknesses and enhance robustness.

In Chapter 6.1, we explored one approach using text-to-image diffusion models and category labels in classification tasks to generate hard-negative challenging images for classifiers. While the scope of our work is limited to image classification, future research could expand this direction by developing automated pipelines for identifying model failure points and generating high-quality synthetic data tailored to mitigate these issues across diverse domains.

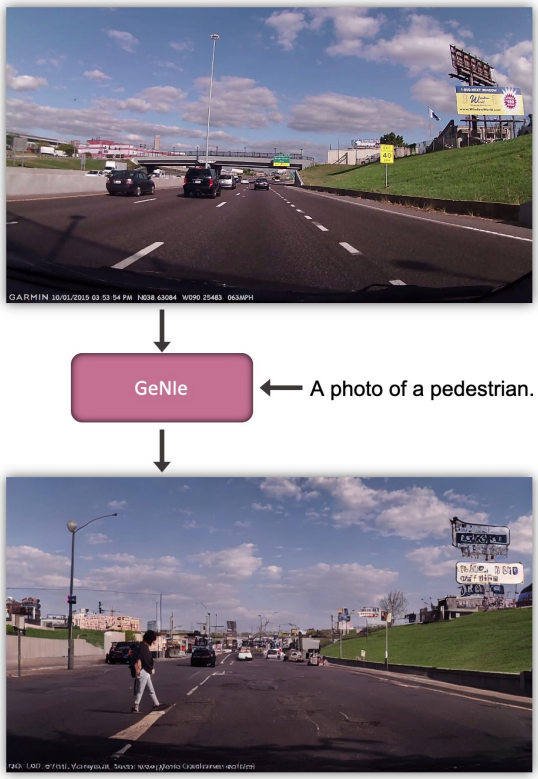


FIGURE 7.2. **Synthetic Data Generation to Address Model Failures:** Future research could extend the ideas presented in Chapter 6.1 by developing automated pipelines to systematically identify model failure points and generate high-quality synthetic data specifically tailored to address these deficiencies. For instance, a rare scenario like a pedestrian crossing a highway poses significant challenges for deep learning models due to its underrepresentation in training datasets. However, as demonstrated in Chapter 6.1, it is possible to effectively generate synthetic data using GeNIe by editing a source image and producing a new image aligned with the target prompt, thus addressing such rare and critical scenarios.

7.3 Dynamic Resource Allocation for Individual Inputs

Deep learning models often allocate static computational resources to all inputs, regardless of their complexity or difficulty. For instance, vision transformers typically use a fixed amount of computation for every input with specific resolution, which can lead to inefficiencies when processing data with varying levels of complexity. Dynamic computation techniques offer a solution by enabling models to allocate computational resources selectively: simpler tokens or regions are processed with reduced effort, while more resources are dedicated to complex areas of the input. For example, in image classification, a clear image of a single object may require fewer computational resources, while a cluttered image with multiple overlapping objects demands greater processing effort.

In Adaptive Token Sampling (ATS), discussed in Chapter 2.1, we explore how to dynamically select important tokens for further processing in vision transformers. This approach allocates more tokens to challenging images and fewer to simpler ones, optimizing computation. Such strategies become even more critical in large language models, where the variance in input complexity is significantly higher than in image classification tasks.

However, as highlighted in Chapter 5.1, dynamic resource allocation introduces the risk of adversarial energy attacks. Attackers can manipulate inputs to artificially increase resource usage, creating potential vulnerabilities. Addressing these risks while leveraging dynamic computation remains a crucial area for future research.

7.4 Parameter-Efficient Fine-Tuning

As generative AI becomes more accessible and democratized, the number of fine-tuned models is expected to explode in the near future. Organizations will increasingly customize foundation models to meet unique requirements, driving exponential growth in the variety and number of fine-tuned variants. However, this rapid expansion presents significant challenges, particularly regarding memory usage and storage. Each fine-tuned model often requires maintaining a separate instance as large as the original model for individual use cases, further straining computational and storage resources.

This increases the memory footprint, particularly in scenarios where devices or servers need to host multiple models concurrently. For edge devices, such as smartphones and IoT devices, the limited memory capacity becomes a critical bottleneck. Storing and running multiple fine-tuned models locally is infeasible, especially when these models demand significant computational and memory resources. Even with advances in model compression techniques, such as quantization or pruning, the challenge of managing a growing library of fine-tuned models remains substantial.

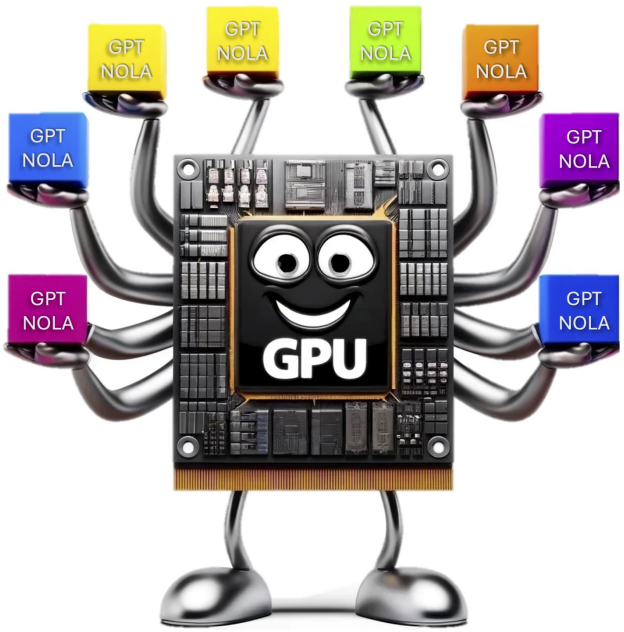


FIGURE 7.3. **The Growing Memory and Storage Challenges of Fine-Tuned Models:** The rapid growth of fine-tuned models poses significant challenges in memory usage and storage. Fine-tuned models are tailored to specific tasks, often requiring multiple models to be maintained for various use cases. This creates substantial strain on hardware resources, particularly when deploying these models on a single GPU, where memory limitations can hinder scalability and performance. Addressing these challenges is crucial as the demand for task-specific fine-tuning continues to expand.

Techniques like parameter-efficient fine-tuning, such as LoRA or adapters, reduce the size of fine-tuned models by updating only a subset of parameters, thereby minimizing memory and storage overhead. In Chapter 3.1, we introduce NOLA, a parameter-efficient method for fine-tuning that allows flexibility in the number of allocated parameters for each task, providing adaptability across various applications. Future research could explore identifying the optimal parameter allocation for different tasks, exploring whether specific task characteristics—such as complexity, data availability, or semantic similarity to the pre-trained model—affect the required parameter count. Understanding these relationships would not only improve model efficiency but also guide the design of more adaptive fine-tuning techniques.

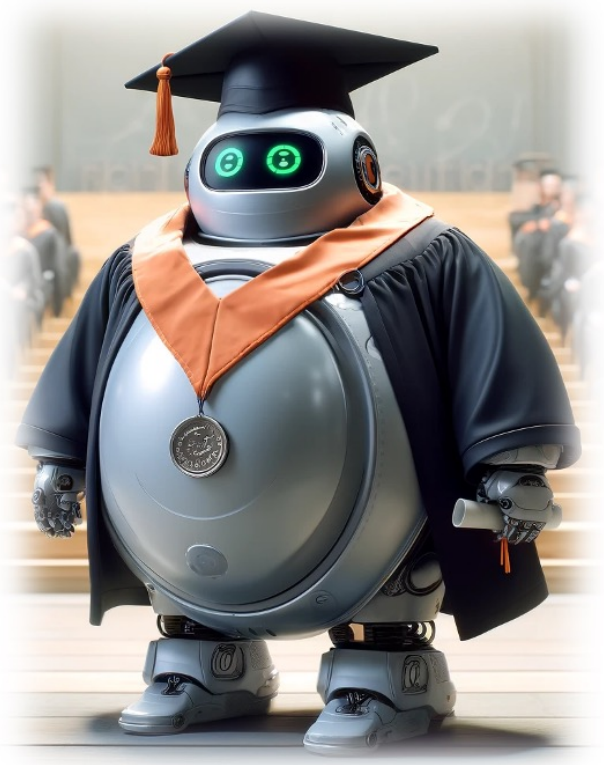


FIGURE 7.4. **Balancing Scale and Expertise:** Training large language models with extensive knowledge across all domains is inefficient. A more effective alternative is to train lightweight models specialized for specific domains. This approach strikes a balance between expertise and model size, allowing for flexibility in tailoring the model's capabilities to meet user requirements while optimizing computational resources.

7.5 The Shift Towards Lightweight Domain-Specialized Models

Recent advancements in models like GPT-4o showcase exceptional zero-shot performance across a wide range of tasks. However, in practical applications, the goal is not to develop AI agents proficient in every conceivable task but to focus on compact, lightweight models specialized in specific domains. This specialization allows for efficient and targeted solutions that align with real-world needs.

Balancing expertise with model size offers flexibility, tailoring the model's capabilities to user requirements while optimizing computational resources. Most user requests necessitate domain-specific expertise, making it more efficient to deploy lightweight, specialized models rather than a single massive model capable of general-purpose performance. This approach not only reduces resource consumption but also enhances the speed and efficiency of task execution.

As discussed in CompRess [240] in Chapter 2.3, one viable strategy involves leveraging large models to acquire a comprehensive range of knowledge and then transferring this expertise to smaller, task-specific models. This is achieved through knowledge distillation, where the larger model serves as a teacher, guiding the training of a smaller student model focused solely on the relevant domain. Unlike humans, AI systems can transfer knowledge far more efficiently. For example, in image classification, a teacher model can provide rich probability distributions over thousands of classes, offering significantly more information than human-provided one-hot annotations for each image. Future work should focus on designing tasks that effectively equip the student model with the essential skills and knowledge needed for the specified domain expertise.

Bibliography

[1] *Code and weights for byol*. https://github.com/deepmind/deepmind-research/tree/master/byol.

[2] *Contrastive multiview coding*. https://github.com/HobbitLong/CMC.

[3] *Contrastive representation distillation (crd), and benchmark of recent knowledge distillation methods*. https://github.com/HobbitLong/RepDistiller.

[4] *Deploying transformers on the apple neural engine: https://machinelearning.apple.com/research/neural-engine-transformers*. https://machinelearning.apple.com/research/neural-engine-transformers.

[5] *Gpt store: https://openai.com/blog/introducing-the-gpt-store*. https://openai.com/blog/introducing-the-gpt-store.

[6] *Gtc march 2024 keynote with nvidia ceo jensen huang : https://www.youtube.com/watch?v=y2f8yisis6e*. https://www.youtube.com/watch?v=Y2F8yisiS6E.

[7] *iphone battery capacity*. https://bigthink.com/the-future/battery-technology-lags/.

[8] *Jetson nano: https://developer.nvidia.com/embedded/jetson-nano*. https://developer.nvidia.com/embedded/jetson-nano.

[9] *A library for efficient similarity search and clustering of dense vectors*. https://github.com/facebookresearch/faiss.

[10] *Nvidia mlperf llm inference records https://developer.nvidia.com/blog/nvidia-h200-tensor-core-gpus-and-nvidia-tensorrt-llm-set-mlperf-llm-inference-records/*. https://developer.nvidia.com/blog/nvidia-h200-tensor-core-gpus-and-nvidia-tensorrt-llm-set-mlperf-llm-inference-records/.

[11] *Official pytorch supervised imagenet training code*. https://github.com/pytorch/examples/blob/master/imagenet/main.py.

[12] *Pytorch implementation of moco: https://arxiv.org/abs/1911.05722*. https://github.com/facebookresearch/moco.

[13] *Silicon valley is pricing academics out of ai research: https://www.washingtonpost.com/technology/2024/03/10/big-tech-companies-ai-research/*. https://www.washingtonpost.com/technology/2024/03/10/big-tech-companies-ai-research/.

[14] *Simclr - a simple framework for contrastive learning of visual representations https://arxiv.org/abs/2002.05709*. https://github.com/google-research/simclr.

[15] *Starship robot*. https://www.wevolver.com/specs/starship-technologies-starship-robot.

[16] *Torchvision models*. https://pytorch.org/docs/stable/torchvision/models.html.

[17] *Unsupervised learning of visual features by contrasting cluster assignments*. https://github.com/facebookresearch/swav.

[18] A. Afrasiyabi, J.-F. Lalonde, and C. Gagné, *Associative alignment for few-shot image classification*, in ECCV, 2019.

[19] A. Aghajanyan, S. Gupta, and L. Zettlemoyer, *Intrinsic dimensionality explains the effectiveness of language model fine-tuning*, in Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), 2021, pp. 7319–7328.

[20] S. Ahn, S. X. Hu, A. Damianou, N. D. Lawrence, and Z. Dai, *Variational information distillation for knowledge transfer*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 9163–9171.

[21] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, *Optuna: A next-generation hyperparameter optimization framework*, in Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining, 2019, pp. 2623–2631.

[22] J.-B. Alayrac, J. Donahue, P. Luc, A. Miech, I. Barr, Y. Hasson, K. Lenc, A. Mensch, K. Millican, M. Reynolds, R. Ring, E. Rutherford, S. Cabi, T. Han, Z. Gong, S. Samangooei, M. Monteiro, J. Menick, S. Borgeaud, A. Brock, A. Nematzadeh, S. Sharifzadeh, M. Binkowski, R. Barreira, O. Vinyals, A. Zisserman, and K. Simonyan, *Flamingo: a visual language model for few-shot learning*, 2022.

[23] A. Ali, H. Touvron, M. Caron, P. Bojanowski, M. Douze, A. Joulin, I. Laptev, N. Neverova, G. Synnaeve, J. Verbeek, et al., *Xcit: Cross-covariance image transformers*, Advances in neural information processing systems, 34 (2021).

[24] A. Antoniou and A. Storkey, *Assume, augment and learn: Unsupervised few-shot meta-learning via random labels and data augmentation*, arxiv:1902.09884, (2019).

[25] M. Assran, N. Ballas, L. Castrejon, and M. Rabbat, *Supervision accelerates pre-training in contrastive semi-supervised learning of visual representations*, arXiv preprint arXiv:2006.10803, (2020).

[26] M. Assran, M. Caron, I. Misra, P. Bojanowski, A. Joulin, N. Ballas, and M. Rabbat, *Semi-supervised learning of visual features by non-parametrically predicting view assignments with support samples*, ICCV, (2021).

[27] M. Azabou, M. G. Azar, R. Liu, C.-H. Lin, E. C. Johnson, K. Bhaskaran-Nair, M. Dabagia, B. Avila-Pires, L. Kitchell, K. B. Hengen, et al., *Mine your own view: Self-supervised learning through across-sample prediction*, arXiv preprint arXiv:2102.10106, (2021).

[28] S. Azizi, S. Kornblith, C. Saharia, M. Norouzi, and D. J. Fleet, *Synthetic data from diffusion models improves imagenet classification*, 2023.

[29] J. Ba and R. Caruana, *Do deep nets really need to be deep?*, in Advances in neural information processing systems, 2014, pp. 2654–2662.

[30] J. L. Ba, J. R. Kiros, and G. E. Hinton, *Layer normalization*, arXiv preprint arXiv:1607.06450, (2016).

[31] P. Bachman, R. D. Hjelm, and W. Buchwalter, *Learning representations by maximizing mutual information across views*, in Advances in Neural Information Processing Systems, 2019, pp. 15509–15519.

[32] H. Bagherinezhad, M. Horton, M. Rastegari, and A. Farhadi, *Label refinery: Improving imagenet classification through label progression*, arXiv preprint arXiv:1805.02641, (2018).

[33] K. Banerjee, R. R. Gupta, K. Vyas, B. Mishra, et al., *Exploring alternatives to softmax function*, arXiv preprint arXiv:2011.11538, (2020).

[34] E. Baum and F. Wilczek, *Supervised learning of probability distributions by neural networks*, in Neural Information Processing Systems, D. Anderson, ed., American Institute of Physics, 1988.

[35] M. A. Bautista, A. Sanakoyeu, E. Tikhoncheva, and B. Ommer, *Cliquecnn: Deep unsupervised exemplar learning*, in Advances in Neural Information Processing Systems, vol. 29, Curran Associates, Inc., 2016.

[36] B. E. Bejnordi, T. Blankevoort, and M. Welling, *Batch-shaping for learning conditional channel gated networks*, arXiv preprint arXiv:1907.06627, (2019).

[37] Y. Bengio, N. Léonard, and A. Courville, *Estimating or propagating gradients through stochastic neurons for conditional computation*, 2013.

[38] G. Bertasius, H. Wang, and L. Torresani, *Is space-time attention all you need for video understanding*, in International Conference on Machine Learning (ICML), 2021.

[39] T. Bolukbasi, J. Wang, O. Dekel, and V. Saligrama, *Adaptive neural networks for efficient inference*, in International Conference on Machine Learning, PMLR, 2017, pp. 527–536.

[40] D. Bolya, C.-Y. Fu, X. Dai, P. Zhang, C. Feichtenhofer, and J. Hoffman, *Token merging: Your vit but faster*, arXiv preprint arXiv:2210.09461, (2022).

[41] L. Bossard, M. Guillaumin, and L. Van Gool, *Food-101 – mining discriminative components with random forests*, in European Conference on Computer Vision, 2014.

[42] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, *Signature verification using a" siamese" time delay neural network*, Advances in neural information processing systems, 6 (1993), pp. 737–744.

[43] T. Brooks, B. Peebles, C. Holmes, W. DePue, Y. Guo, L. Jing, D. Schnurr, J. Taylor, T. Luhman, E. Luhman, C. Ng, R. Wang, and A. Ramesh, *Video generation models as world simulators*, (2024).

[44] J. R. Brown, Y. Zhao, I. Shumailov, and R. D. Mullins, *Dartformer: Finding the best type of attention*, arXiv preprint arXiv:2210.00641, (2022).

[45] T. B. Brown, D. Mané, A. Roy, M. Abadi, and J. Gilmer, *Adversarial patch*, arXiv preprint arXiv:1712.09665, (2017).

[46] T. B. Brown, B. Mann, et al., *Language models are few-shot learners*, in NeurIPS, 2020.

[47] C. Buciluǎ, R. Caruana, and A. Niculescu-Mizil, *Model compression*, in Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, 2006, pp. 535–541.

[48] A. Bulat, J. M. Perez Rua, S. Sudhakaran, B. Martinez, and G. Tzimiropoulos, *Space-time mixing attention for video transformer*, in Advances in Neural Information Processing Systems (NeurIPS), 2021.

[49] H. Cai, C. Gan, and S. Han, *Efficientvit: Enhanced linear attention for high-resolution low-computation visual recognition*, arXiv preprint arXiv:2205.14756, (2022).

[50] J. Cai, Y. Wang, J.-N. Hwang, et al., *Ace: Ally complementary experts for solving long-tailed recognition in one-shot*, in ICCV, 2021, pp. 112–121.

[51] S. Cai, Y. Shu, and W. Wang, *Dynamic routing networks*, in Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, 2021, pp. 3588–3597.

[52] K. Cao, C. Wei, A. Gaidon, N. Arechiga, and T. Ma, *Learning imbalanced datasets with label-distribution-aware margin loss*, NeurIPS, 32 (2019).

[53] S. Cao, L. Ma, W. Xiao, C. Zhang, Y. Liu, L. Zhang, L. Nie, and Z. Yang, *Seernet: Predicting convolutional neural network feature-map sparsity through low-bit quantization*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 11216–11225.

[54] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, *End-to-end object detection with transformers*, in ECCV, Springer, 2020, pp. 213–229.

[55] ———, *End-to-end object detection with transformers*, in ECCV, 2020, pp. 213–229.

[56] M. Caron, P. Bojanowski, A. Joulin, and M. Douze, *Deep clustering for unsupervised learning of visual features*, in Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 132–149.

[57] M. Caron, I. Misra, J. Mairal, P. Goyal, P. Bojanowski, and A. Joulin, *Unsupervised learning of visual features by contrasting cluster assignments*, in Advances in Neural Information Processing Systems, vol. 33, 2020.

[58] M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, and A. Joulin, *Pytorch implementation of dino*. https://github.com/facebookresearch/dino.

[59] M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, and A. Joulin, *Emerging properties in self-supervised vision transformers*, (2021).

[60] J. Carreira, E. Noland, A. Banki-Horvath, C. Hillier, and A. Zisserman, *A short note about kinetics-600*, in arXiv preprint arXiv:1808.01340v1, 2018.

[61] A. Chegini and S. Feizi, *Identifying and mitigating model failures through few-shot clip-aided diffusion generation*, arXiv preprint arXiv:2312.05464, (2023).

[62] C.-F. Chen, Q. Fan, and R. Panda, *Crossvit: Cross-attention multi-scale vision transformer for image classification*, in IEEE/CVF International Conference on Computer Vision (ICCV), 2021.

[63] G. Chen, C. Lin, L. Ren, J. Lu, and J. Zhou, *Self-critical attention learning for person re-identification*, in ICCV, 2019, pp. 9637–9646.

[64] J. Chen, X. Wang, Z. Guo, X. Zhang, and J. Sun, *Dynamic region-aware convolution*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 8064–8073.

[65] K. Chen, J. Wang, J. Pang, Y. Cao, Y. Xiong, X. Li, S. Sun, W. Feng, Z. Liu, J. Xu, et al., *Mmdetection: Open mmlab detection toolbox and benchmark*, arXiv preprint arXiv:1906.07155, (2019).

[66] S. Chen, H. Chen, M. Haque, C. Liu, and W. Yang, *Slothbomb: Efficiency poisoning attack against dynamic neural networks*.

[67] S. Chen, C. Ge, Z. Tong, J. Wang, Y. Song, J. Wang, and P. Luo, *Adaptformer: Adapting vision transformers for scalable visual recognition*, Advances in Neural Information Processing Systems, 35 (2022), pp. 16664–16678.

[68] S. Chen, M. Haque, Z. Song, C. Liu, and W. Yang, *Transslowdown: Efficiency attacks on neural machine translation systems*, (2021).

[69] S. Chen, Z. Song, M. Haque, C. Liu, and W. Yang, *Nicgslowdown: Evaluating the efficiency robustness of neural image caption generation models*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 15365–15374.

[70] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, *A simple framework for contrastive learning of visual representations*, arXiv preprint arXiv:2002.05709, (2020).

[71] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, *A simple framework for contrastive learning of visual representations*, in ICML, 2020.

[72] T. Chen, S. Kornblith, K. Swersky, M. Norouzi, and G. E. Hinton, *Big self-supervised models are strong semi-supervised learners*, Advances in Neural Information Processing Systems, 33 (2020), pp. 22243–22255.

[73] W. Chen, C. Si, W. Wang, L. Wang, Z. Wang, and T. Tan, *Few-shot learning with part discovery and augmentation from unlabeled images*, arXiv preprint arXiv:2105.11874, (2021).

[74] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen, *Compressing neural networks with the hashing trick*, in International conference on machine learning, PMLR, 2015, pp. 2285–2294.

[75] W.-Y. Chen, Y.-C. Liu, Z. Kira, Y.-C. F. Wang, and J.-B. Huang, *A closer look at few-shot classification.*, in ICLR, 2019.

[76] X. Chen, H. Fan, R. Girshick, and K. He, *Improved baselines with momentum contrastive learning*, arXiv preprint arXiv:2003.04297, (2020).

[77] X. Chen and K. He, *Exploring simple siamese representation learning*, 2020.

[78] Z. Chen, Y. Fu, Y.-X. Wang, L. Ma, W. Liu, and M. Hebert, *Image deformation meta-networks for one-shot learning*, in CVPR, 2019.

[79] Z. Chen, J. Ge, H. Zhan, S. Huang, and D. Wang, *Pareto self-supervised training for few-shot learning*, in CVPR, 2021.

[80] B. Cheng, A. G. Schwing, and A. Kirillov, *Per-pixel classification is not all you need for semantic segmentation*, in NeurIPS, 2021.

[81] ———, *Per-pixel classification is not all you need for semantic segmentation*, in Advances in Neural Information Processing Systems (NeurIPS), 2021.

[82] Y. Cheng, *Mean shift, mode seeking, and clustering*, IEEE transactions on pattern analysis and machine intelligence, 17 (1995), pp. 790–799.

[83] R. Child, S. Gray, A. Radford, and I. Sutskever, *Generating long sequences with sparse transformers*, in arXiv preprint arXiv:1904.10509, 2019.

[84] F. Chollet, *Xception: Deep learning with depthwise separable convolutions*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 1251–1258.

[85] S. Chopra, R. Hadsell, and Y. LeCun, *Learning a similarity metric discriminatively, with application to face verification*, in 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), vol. 1, IEEE, 2005, pp. 539–546.

[86] K. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlos, P. Hawkins, J. Davis, A. Mohiuddin, L. Kaiser, et al., *Rethinking attention with performers*, arXiv preprint arXiv:2009.14794, (2020).

[87] X. Chu, Z. Tian, Y. Wang, B. Zhang, H. Ren, X. Wei, H. Xia, and C. Shen, *Twins: Revisiting the design of spatial attention in vision transformers*, in Advances in Neural Information Processing Systems, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, eds., vol. 34, Curran Associates, Inc., 2021, pp. 9355–9366.

[88] X. Chu, Z. Tian, B. Zhang, X. Wang, X. Wei, H. Xia, and C. Shen, *Conditional positional encodings for vision transformers*, in arXiv preprint arXiv:2102.10882, 2021.

[89] C.-Y. Chuang, J. Robinson, Y.-C. Lin, A. Torralba, and S. Jegelka, *Debiased contrastive learning*, in Advances in Neural Information Processing Systems, 2020.

[90] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi, *Describing textures in the wild*, in Computer Vision and Pattern Recognition, 2014.

[91] D. Comaniciu and P. Meer, *Mean shift: A robust approach toward feature space analysis*, IEEE Transactions on pattern analysis and machine intelligence, 24 (2002), pp. 603–619.

[92] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le, *Autoaugment: Learning augmentation policies from data*, 2019.

[93] E. D. Cubuk, B. Zoph, J. Shlens, and Q. Le, *Randaugment: Practical automated data augmentation with a reduced search space*, in NeurIPS, 2020.

[94] E. D. Cubuk, B. Zoph, J. Shlens, and Q. Le, *Randaugment: Practical automated data augmentation with a reduced search space*, in Advances in Neural Information Processing Systems, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, eds., vol. 33, Curran Associates, Inc., 2020, pp. 18613–18624.

[95] ———, *Randaugment: Practical automated data augmentation with a reduced search space*, in Advances in Neural Information Processing Systems, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, eds., vol. 33, Curran Associates, Inc., 2020, pp. 18613–18624.

[96] E. D. Cubuk, B. Zoph, J. Shlens, and Q. V. Le, *Randaugment: Practical automated data augmentation with a reduced search space*, 2019.

[97] J. Cui, S. Liu, Z. Tian, Z. Zhong, and J. Jia, *Reslt: Residual learning for long-tailed recognition*, IEEE transactions on pattern analysis and machine intelligence, 45 (2022), pp. 3695–3706.

[98] J. Cui, Z. Zhong, S. Liu, B. Yu, and J. Jia, *Parametric contrastive learning*, in Proceedings of the IEEE/CVF international conference on computer vision, 2021, pp. 715–724.

[99] ———, *Parametric contrastive learning*, in ICCV, 2021, pp. 715–724.

[100] Y. Cui, M. Jia, T.-Y. Lin, Y. Song, and S. Belongie, *Class-balanced loss based on effective number of samples*, in CVPR, 2019, pp. 9268–9277.

[101] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, *Imagenet: A large-scale hierarchical image database*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2009.

[102] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, *Qlora: Efficient finetuning of quantized llms*, 2023.

[103] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, arXiv:1810.04805 [cs], (2019). arXiv: 1810.04805.

[104] A. Diba, M. Fayyaz, V. Sharma, M. M. Arzani, R. Yousefzadeh, J. Gall, and L. Van Gool, *Spatio-temporal channel correlation networks for action classification*, in European Conference on Computer Vision (ECCV), 2018.

[105] A. Diba, M. Fayyaz, V. Sharma, M. Paluri, J. Gall, R. Stiefelhagen, and L. V. Gool, *Large scale holistic video understanding*, in European Conference on Computer Vision (ECCV), 2020.

[106] A. Diba, V. Sharma, L. V. Gool, and R. Stiefelhagen, *Dynamonet: Dynamic action and motion network*, in IEEE/CVF International Conference on Computer Vision (ICCV), 2019.

[107] M. Ding, B. An, Y. Xu, A. Satheesh, and F. Huang, *SAFLEX: Self-adaptive augmentation via feature label extrapolation*, in The Twelfth International Conference on Learning Representations, 2024.

[108] C. Doersch, A. Gupta, and A. A. Efros, *Unsupervised visual representation learning by context prediction*, in Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 1422–1430.

[109] J. Donahue, P. Krähenbühl, and T. Darrell, *Adversarial feature learning*, in International Conference on Learning Representations, ICLR, 2016.

[110] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al., *An image is worth 16x16 words: Transformers for image recognition at scale*, arXiv preprint arXiv:2010.11929, (2020).

[111] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, *An image is worth 16x16 words: Transformers for image recognition at scale*, in International Conference on Learning Representations (ICLR), 2021.

[112] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, *An image is worth 16x16 words: Transformers for image recognition at scale*, in International Conference on Learning Representations (ICLR), 2021.

[113] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, *An image is worth 16x16 words: Transformers for image recognition at scale*, 2021.

[114] A. Dosovitskiy, J. T. Springenberg, M. Riedmiller, and T. Brox, *Discriminative unsupervised feature learning with convolutional neural networks*, in Advances in neural information processing systems, 2014, pp. 766–774.

[115] G. Du, C. Tian, Z. Li, D. Zhang, Y. Yin, and Y. Ouyang, *Efficient softmax hardware architecture for deep neural networks*, in Proceedings of the 2019 on Great Lakes Symposium on VLSI, 2019, pp. 75–80.

[116] L. Dunlap, C. Mohri, H. Zhang, D. Guillory, T. Darrell, J. E. Gonzalez, A. Rohrbach, and A. Raghunathan, *Using language to extend to unseen domains*, International Conference on Learning Representations (ICLR), (2023).

[117] L. Dunlap, A. Umino, H. Zhang, J. Yang, J. E. Gonzalez, and T. Darrell, *Diversify your vision datasets with automatic diffusion-based augmentation*, 2023.

[118] N. Dvornik, J. Mairal, and C. Schmid, *Diversity with cooperation: Ensemble methods for few-shot classification*, in ICCV, 2019.

[119] D. Dwibedi, Y. Aytar, J. Tompson, P. Sermanet, and A. Zisserman, *With a little help from my friends: Nearest-neighbor contrastive learning of visual representations*, 2021.

[120] M. Elbayad, J. Gu, E. Grave, and M. Auli, *Depth-adaptive transformer*, arXiv preprint arXiv:1910.10073, (2019).

[121] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, *The pascal visual object classes (voc) challenge*, International journal of computer vision, 88 (2010), pp. 303–338.

[122] H. Fan, B. Xiong, K. Mangalam, Y. Li, Z. Yan, J. Malik, and C. Feichtenhofer, *Multiscale vision transformers*, in IEEE/CVF International Conference on Computer Vision (ICCV), 2021.

[123] Q. Fan, C.-F. R. Chen, H. Kuehne, M. Pistoia, and D. Cox, *More Is Less: Learning Efficient Video Representations by Temporal Aggregation Modules*, in Advances in Neural Information Processing Systems (NeurIPS), 2019.

[124] Z. Fang, J. Wang, L. Wang, L. Zhang, Y. Yang, and Z. Liu, *Seed: Self-supervised distillation for visual representation*, in International Conference on Learning Representations, 2021.

[125] M. Fayyaz, E. Bahrami, A. Diba, M. Noroozi, E. Adeli, L. Van Gool, and J. Gall, *3d cnns with adaptive temporal feature resolutions*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2021.

[126] M. Fayyaz, S. A. Koohpayegani, F. R. Jafari, S. Sengupta, H. R. V. Joze, E. Sommerlade, H. Pirsiavash, and J. Gall, *Adaptive token sampling for efficient vision transformers*, 2021.

[127] M. Fayyaz, S. A. Koohpayegani, F. R. Jafari, S. Sengupta, H. R. V. Joze, E. Sommerlade, H. Pirsiavash, and J. Gall, *Adaptive token sampling for efficient vision transformers*, in Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XI, Springer, 2022, pp. 396–414.

[128] M. Fayyaz, S. A. Kouhpayegani, F. R. Jafari, E. Sommerlade, H. R. V. Joze, H. Pirsiavash, and J. Gall, *Ats: Adaptive token sampling for efficient vision transformers*, arXiv preprint arXiv:2111.15667, (2021).

[129] W. Fedus, B. Zoph, and N. Shazeer, *Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity*, J. Mach. Learn. Res, 23 (2021), pp. 1–40.

[130] L. Fei-Fei, R. Fergus, and P. Perona, *Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories*, Computer Vision and Pattern Recognition Workshop, (2004).

[131] C. Feichtenhofer, *X3d: Expanding architectures for efficient video recognition*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020.

[132] C. Feichtenhofer, H. Fan, J. Malik, and K. He, *Slowfast networks for video recognition*, in IEEE/CVF international conference on computer vision (ICCV), 2019.

[133] R. Feinman, R. R. Curtin, S. Shintre, and A. B. Gardner, *Detecting adversarial samples from artifacts*, arXiv preprint arXiv:1703.00410, (2017).

[134] C.-M. Feng, K. Yu, Y. Liu, S. Khan, and W. Zuo, *Diverse data augmentation with diffusions for effective test-time prompt tuning*, 2023.

[135] Z. Feng, C. Xu, and D. Tao, *Self-supervised representation learning by rotation feature decoupling*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 10364–10374.

[136] M. Figurnov, M. D. Collins, Y. Zhu, L. Zhang, J. Huang, D. Vetrov, and R. Salakhutdinov, *Spatially adaptive computation time for residual networks*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 1039–1048.

[137] C. Finn, P. Abbeel, and S. Levine, *Model-agnostic meta-learning for fast adaptation of deep networks*, in Proceedings of the 34th International Conference on Machine Learning, 2017, pp. 1126–1135.

[138] M. Frid-Adar, I. Diamant, E. Klang, M. Amitai, J. Goldberger, and H. Greenspan, *Gan-based synthetic medical image augmentation for increased cnn performance in liver lesion classification*, Neurocomputing, (2018).

[139] J. Fu, H. Zheng, and T. Mei, *Look closer to see better: Recurrent attention convolutional neural network for fine-grained image recognition*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.

[140] Y. Fu, S. Zhang, S. Wu, C. Wan, and Y. Lin, *Patch-fool: Are vision transformers always robust against adversarial perturbations?*, arXiv preprint arXiv:2203.08392, (2022).

[141] T. Furlanello, Z. C. Lipton, M. Tschannen, L. Itti, and A. Anandkumar, *Born-again neural networks*, in Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018, J. G. Dy and A. Krause, eds., vol. 80 of Proceedings of Machine Learning Research, PMLR, 2018, pp. 1602–1611.

[142] T. Furlanello, Z. C. Lipton, M. Tschannen, L. Itti, and A. Anandkumar, *Born again neural networks*, 2018.

[143] R. Gal, Y. Alaluf, Y. Atzmon, O. Patashnik, A. H. Bermano, G. Chechik, and D. Cohen-Or, *An image is worth one word: Personalizing text-to-image generation using textual inversion*, 2022.

[144] R. Gal, Y. Alaluf, Y. Atzmon, O. Patashnik, A. H. Bermano, G. Chechik, and D. Cohen-Or, *An image is worth one word: Personalizing text-to-image generation using textual inversion*, arXiv preprint arXiv:2208.01618, (2022).

[145] X. Gao, Y. Zhao, Ł. Dudziak, R. Mullins, and C.-z. Xu, *Dynamic channel pruning: Feature boosting and suppression*, arXiv preprint arXiv:1810.05331, (2018).

[146] Y. Gao, W. Liu, and F. Lombardi, *Design and implementation of an approximate softmax layer for deep neural networks*, in 2020 IEEE International Symposium on Circuits and Systems (ISCAS), IEEE, 2020, pp. 1–5.

[147] C. Gardent, A. Shimorina, S. Narayan, and L. Perez-Beltrachini, *The webnlg challenge: Generating text from rdf data*, in Proceedings of the 10th International Conference on Natural Language Generation, 2017, pp. 124–133.

[148] S. Gidaris, A. Bursuc, G. Puy, N. Komodakis, M. Cord, and P. Perez, *Obow: Online bag-of-visual-words generation for self-supervised learning*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), June 2021, pp. 6830–6840.

[149] S. Gidaris, P. Singh, and N. Komodakis, *Unsupervised representation learning by predicting image rotations*, in International Conference on Learning Representations, 2018.

[150] R. Girshick, *Fast r-cnn*, in Proceedings of the IEEE international conference on computer vision, 2015, pp. 1440–1448.

[151] J. Goldberger, G. E. Hinton, S. Roweis, and R. R. Salakhutdinov, *Neighbourhood components analysis*, Advances in neural information processing systems, 17 (2004), pp. 513–520.

[152] Y. Gong, L. Liu, M. Yang, and L. Bourdev, *Compressing deep convolutional networks using vector quantization*, in arXiv preprint arXiv:1412.6115, 2014.

[153] Z. Gong, J. Liu, Q. Wang, Y. Yang, J. Wang, W. Wu, Y. Xian, D. Zhao, and R. Yan, *Prequant: A task-agnostic quantization approach for pre-trained language models*, 2023.

[154] I. J. Goodfellow, J. Shlens, and C. Szegedy, *Explaining and harnessing adversarial examples*, in arXiv preprint arXiv:1412.6572, 2014.

[155] P. Goyal, D. Mahajan, A. Gupta, and I. Misra, *Scaling and benchmarking self-supervised visual representation learning*, in Proceedings of the IEEE International Conference on Computer Vision, 2019, pp. 6391–6400.

[156] S. Goyal, A. R. Choudhury, S. M. Raje, V. T. Chakaravarthy, Y. Sabharwal, and A. Verma, *Power-bert: Accelerating bert inference via progressive word-vector elimination*, in International Conference on Machine Learning (ICML), 2020.

[157] B. Graham, A. El-Nouby, H. Touvron, P. Stock, A. Joulin, H. Jégou, and M. Douze, *Levit: a vision transformer in convnet's clothing for faster inference*, in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021, pp. 12259–12269.

[158] A. Graves, *Adaptive computation time for recurrent neural networks*, arXiv preprint arXiv:1603.08983, (2016).

[159] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. Richemond, E. Buchatskaya, C. Doersch, B. Avila Pires, Z. Guo, M. Gheshlaghi Azar, B. Piot, k. kavukcuoglu, R. Munos, and M. Valko, *Bootstrap your own latent - a new approach to self-supervised learning*, in Advances in Neural Information Processing Systems, 2020.

[160] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. H. Richemond, E. Buchatskaya, C. Doersch, B. A. Pires, Z. D. Guo, M. G. Azar, et al., *Bootstrap your own latent: A new approach to self-supervised learning*, arXiv preprint arXiv:2006.07733, (2020).

[161] J. Guan, Y. Liu, Q. Liu, and J. Peng, *Energy-efficient amortized inference with cascaded deep classifiers*, arXiv preprint arXiv:1710.03368, (2017).

[162] J. Guo, Y. Tang, K. Han, X. Chen, H. Wu, C. Xu, C. Xu, and Y. Wang, *Hire-mlp: Vision mlp via hierarchical rearrangement*, arXiv preprint arXiv:2108.13341, (2021).

[163] M.-H. Guo, Z.-N. Liu, T.-J. Mu, and S.-M. Hu, *Beyond self-attention: External attention using two linear layers for visual tasks*, IEEE Transactions on Pattern Analysis and Machine Intelligence, (2022).

[164] Q. Guo, X. Qiu, P. Liu, Y. Shao, X. Xue, and Z. Zhang, *Star-transformer*, in arXiv preprint arXiv:1902.09113, 2019.

[165] R. Hadsell, S. Chopra, and Y. LeCun, *Dimensionality reduction by learning an invariant mapping*, in 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06), vol. 2, IEEE, 2006, pp. 1735–1742.

[166] T. J. Ham, S. J. Jung, S. Kim, Y. H. Oh, Y. Park, Y. Song, J.-H. Park, S. Lee, K. Park, J. W. Lee, et al., *Aˆ 3: Accelerating attention mechanisms in neural networks with approximation*, in 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA), IEEE, 2020, pp. 328–341.

[167] K. Hambardzumyan, H. Khachatrian, and J. May, *WARP: Word-level Adversarial ReProgramming*, arXiv:2101.00121 [cs], (2020). arXiv: 2101.00121.

[168] K. Han, A. Xiao, E. Wu, J. Guo, C. Xu, and Y. Wang, *Transformer in transformer*, in Advances in Neural Information Processing Systems (NeurIPS), 2021.

[169] T. Han, W. Xie, and A. Zisserman, *Self-supervised co-training for video representation learning*, 2021.

[170] M. Haque, A. Chauhan, C. Liu, and W. Yang, *Ilfo: Adversarial attack on adaptive neural networks*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 14264–14273.

[171] M. Haque, S. Chen, W. A. Haque, C. Liu, and W. Yang, *Nodeattack: Adversarial attack on the energy consumption of neural odes*, (2021).

[172] S. Hayou, J.-F. Ton, A. Doucet, and Y. W. Teh, *Robust pruning at initialization*, arXiv preprint arXiv:2002.08797, (2020).

[173] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. B. Girshick, *Masked autoencoders are scalable vision learners*, in CVPR, IEEE, 2022, pp. 15979–15988.

[174] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick, *Masked autoencoders are scalable vision learners*, 2021.

[175] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, *Momentum contrast for unsupervised visual representation learning*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 9729–9738.

[176] K. He, G. Gkioxari, P. Dollar, and R. Girshick, *Mask r-cnn*, in ICCV, 2017.

[177] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[178] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[179] ———, *Deep residual learning for image recognition*, in CVPR, 2016, pp. 770–778.

[180] R. He, S. Sun, X. Yu, C. Xue, W. Zhang, P. Torr, S. Bai, and X. Qi, *Is synthetic data from generative models ready for image recognition?*, arXiv preprint arXiv:2210.07574, (2022).

[181] Y. He, X. Zhang, and J. Sun, *Channel pruning for accelerating very deep neural networks*, in IEEE/CVF International Conference on Computer Vision (ICCV), 2017.

[182] R. A. Hemmat, M. Pezeshki, F. Bordes, M. Drozdzal, and A. Romero-Soriano, *Feedback-guided data synthesis for imbalanced classification*, 2023.

[183] O. J. Hénaff, A. Srinivas, J. De Fauw, A. Razavi, C. Doersch, S. Eslami, and A. v. d. Oord, *Data-efficient image recognition with contrastive predictive coding*, arXiv preprint arXiv:1905.09272, (2019).

[184] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt, *Measuring massive multitask language understanding*, Proceedings of the International Conference on Learning Representations (ICLR), (2021).

[185] D. Hendrycks and K. Gimpel, *Gaussian error linear units (gelus)*, arXiv preprint arXiv:1606.08415, (2016).

[186] D. Hendrycks, N. Mu, E. D. Cubuk, B. Zoph, J. Gilmer, and B. Lakshminarayanan, *AugMix: A simple data processing method to improve robustness and uncertainty*, Proceedings of the International Conference on Learning Representations (ICLR), (2020).

[187] C. Herrmann, R. S. Bowen, and R. Zabih, *Channel selection using gumbel softmax*, in Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXVII, Springer, 2020, pp. 241–257.

[188] G. Hinton, O. Vinyals, and J. Dean, *Distilling the knowledge in a neural network*, in NIPS Deep Learning and Representation Learning Workshop, 2015.

[189] R. D. Hjelm, A. Fedorov, S. Lavoie-Marchildon, K. Grewal, P. Bachman, A. Trischler, and Y. Bengio, *Learning deep representations by mutual information estimation and maximization*, in International Conference on Learning Representations, 2019.

[190] J. Ho, N. Kalchbrenner, D. Weissenborn, and T. Salimans, *Axial attention in multidimensional transformers*, arXiv preprint arXiv:1912.12180, (2019).

[191] S. Hong, Y. Kaya, I.-V. Modoranu, and T. Dumitraş, *A panda? no, it's a sloth: Slowdown attacks on adaptive multi-exit neural network inference*, arXiv preprint arXiv:2010.02432, (2020).

[192] Y. Hong, J. Zhang, Z. Sun, and K. Yan, *Safa: Sample-adaptive feature augmentation for long-tailed image classification*, in ECCV, 2022.

[193] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. de Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly, *Parameter-Efficient Transfer Learning for NLP*, arXiv:1902.00751 [cs, stat], (2019).

[194] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, *Mobilenets: Efficient convolutional neural networks for mobile vision applications*, in arXiv preprint arXiv:1704.04861, 2017.

[195] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, *Mobilenets: Efficient convolutional neural networks for mobile vision applications*, arXiv preprint arXiv:1704.04861, (2017).

[196] K. Hsu, S. Levine, and C. Finn, *Unsupervised learning via meta-learning*, in ICLR, 2018.

[197] E. J. Hu, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen, et al., *Lora: Low-rank adaptation of large language models*, in International Conference on Learning Representations, 2021.

[198] J. Hu, L. Shen, and G. Sun, *Squeeze-and-excitation networks*, in CVPR, 2018, pp. 7132–7141.

[199] W. Hu, X. Jiang, J. Liu, Y. Yang, and H. Tian, *Meta-dm: Applications of diffusion models on few-shot learning*, 2023.

[200] W. Hua, Y. Zhou, C. M. De Sa, Z. Zhang, and G. E. Suh, *Channel gating neural networks*, Advances in Neural Information Processing Systems, 32 (2019).

[201] G. Huang, D. Chen, T. Li, F. Wu, L. Van Der Maaten, and K. Q. Weinberger, *Multi-scale dense networks for resource efficient image classification*, arXiv preprint arXiv:1703.09844, (2017).

[202] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, *Deep networks with stochastic depth*, in European conference on computer vision, Springer, 2016, pp. 646–661.

[203] J. Huang, Q. Dong, S. Gong, and X. Zhu, *Unsupervised deep learning by neighbourhood discovery*, vol. 97 of Proceedings of Machine Learning Research, Long Beach, California, USA, 09–15 Jun 2019, PMLR, pp. 2849–2858.

[204] ———, *Unsupervised deep learning by neighbourhood discovery*, in International Conference on Machine Learning, PMLR, 2019, pp. 2849–2858.

[205] ———, *Unsupervised deep learning via affinity diffusion*, in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, 2020, pp. 11029–11036.

[206] S.-W. Huang, C.-T. Lin, S.-P. Chen, Y.-Y. W. an Po-Hao Hsu, and S.-H. Lai, *Auggan: Cross domain adaptation with gan-based data augmentation*, European Conference on Computer Vision, (2018).

[207] T. Huynh, S. Kornblith, M. R. Walter, M. Maire, and M. Khademi, *Boosting contrastive self-supervised learning with false negative cancellation*, 2020.

[208] B. Isik, T. Weissman, and A. No, *An information-theoretic justification for model pruning*, in International Conference on Artificial Intelligence and Statistics, PMLR, 2022, pp. 3821–3846.

[209] A. Ivanov, N. Dryden, T. Ben-Nun, S. Li, and T. Hoefler, *Data movement is all you need: A case study on optimizing transformers*, Proceedings of Machine Learning and Systems, 3 (2021), pp. 711–732.

[210] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, *Quantization and training of neural networks for efficient integer-arithmetic-only inference*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 2704–2713.

[211] M. Jaderberg, A. Vedaldi, and A. Zisserman, *Speeding up convolutional neural networks with low rank expansions*, in arXiv preprint arXiv:1405.3866, 2014.

[212] S. Jain, H. Lawrence, A. Moitra, and A. Madry, *Distilling model failures as directions in latent space*, in ArXiv preprint arXiv:2206.14754, 2022.

[213] H. Jang, H. Lee, and J. Shin, *Unsupervised meta-learning via few-shot pseudo-supervised contrastive learning*, in The Eleventh International Conference on Learning Representations, 2022.

[214] S. Jaszczur, A. Chowdhery, A. Mohiuddin, L. Kaiser, W. Gajewski, H. Michalewski, and J. Kanerva, *Sparse is enough in scaling transformers*, in Advances in Neural Information Processing Systems (NeurIPS), 2021.

[215] S. Jenni and P. Favaro, *Self-supervised feature learning by learning to spot artifacts*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 2733–2742.

[216] X. Ji, J. F. Henriques, and A. Vedaldi, *Invariant information clustering for unsupervised image classification and segmentation*, in Proceedings of the IEEE International Conference on Computer Vision, 2019, pp. 9865–9874.

[217] B. Jiang, M. Wang, W. Gan, W. Wu, and J. Yan, *Stm: Spatiotemporal and motion encoding for action recognition*, in IEEE/CVF International Conference on Computer Vision (ICCV), 2019.

[218] Z. Jiang, Q. Hou, L. Yuan, D. Zhou, X. Jin, A. Wang, and J. Feng, *Token labeling: Training a 85.5% top-1 accuracy vision transformer with 56m parameters on imagenet*, in arXiv preprint arXiv:2104.10858v2, 2021.

[219] Z. Jiang, Q. Hou, L. Yuan, D. Zhou, Y. Shi, X. Jin, A. Wang, and J. Feng, *All tokens matter: Token labeling for training better vision transformers*, in Advances in Neural Information Processing Systems (NeurIPS), 2021.

[220] X. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, and Q. Liu, *Tinybert: Distilling bert for natural language understanding*, in arXiv preprint arXiv:1909.10351, 2020.

[221] Y. Kalantidis, M. B. Sariyildiz, N. Pion, P. Weinzaepfel, and D. Larlus, *Hard negative mixing for contrastive learning*, Advances in Neural Information Processing Systems, (2020).

[222] B. Kang, S. Xie, M. Rohrbach, Z. Yan, A. Gordo, J. Feng, and Y. Kalantidis, *Decoupling representation and classifier for long-tailed recognition*, arXiv preprint arXiv:1910.09217, (2019).

[223] ———, *Decoupling representation and classifier for long-tailed recognition*, in ICLR, 2020.

[224] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, *Scaling laws for neural language models*, 2020.

[225] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret, *Transformers are rnns: Fast autoregressive transformers with linear attention*, in International Conference on Machine Learning, PMLR, 2020, pp. 5156–5165.

[226] W. Kay, J. Carreira, K. Simonyan, B. Zhang, C. Hillier, S. Vijayanarasimhan, F. Viola, T. Green, T. Back, A. Natsev, M. Suleyman, and A. Zisserman, *The kinetics human action video dataset*, in arXiv preprint arXiv:1705.06950, 2017.

[227] F. D. Keles, P. M. Wijewardena, and C. Hegde, *On the computational complexity of self-attention*, arXiv preprint arXiv:2209.04881, (2022).

[228] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, *3d gaussian splatting for real-time radiance field rendering*, ACM Transactions on Graphics, 42 (2023).

[229] S. Khodadadeh, L. Boloni, and M. Shah, *Unsupervised meta-learning for few-shot image classification*, in NeurIPS, 2019.

[230] P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, and D. Krishnan, *Supervised contrastive learning*, Advances in Neural Information Processing Systems, 33 (2020).

[231] J. Kim, S. Park, and N. Kwak, *Paraphrasing complex network: Network compression via factor transfer*, in Advances in Neural Information Processing Systems, 2018, pp. 2760–2769.

[232] J.-H. Kim, W. Choo, and H. O. Song, *Puzzle mix: Exploiting saliency and local statistics for optimal mixup*, in International Conference on Machine Learning, PMLR, 2020, pp. 5275–5285.

[233] W. Kim, S. Kim, M. Park, and G. Jeon, *Neuron merging: Compensating for pruned neurons*, Advances in Neural Information Processing Systems, 33 (2020), pp. 585–595.

[234] S. Kong and C. Fowlkes, *Pixel-wise attentional gating for scene parsing*, in 2019 IEEE Winter Conference on Applications of Computer Vision (WACV), IEEE, 2019, pp. 1024–1033.

[235] S. A. Koohpayegani, N. K. L, P. Nooralinejad, S. Kolouri, and H. Pirsiavash, *NOLA: Compressing loRA using linear combination of random basis*, in The Twelfth International Conference on Learning Representations, 2024.

[236] S. A. Koohpayegani and H. Pirsiavash, *Sima: Simple softmax-free attention for vision transformers*, 2022.

[237] ———, *Sima: Simple softmax-free attention for vision transformers*, arXiv preprint arXiv:2206.08898, (2022).

[238] S. A. Koohpayegani, A. Singh, K. L. Navaneet, H. Jamali-Rad, and H. Pirsiavash, *Genie: Generative hard negative images through diffusion*, 2024.

[239] S. A. Koohpayegani, A. Tejankar, and H. Pirsiavash, *Compress: Self-supervised learning by compressing representations*, in NeurIPS, 2020.

[240] ———, *Compress: Self-supervised learning by compressing representations*, Advances in neural information processing systems, (2020).

[241] ———, *Mean shift for self-supervised learning*, in ICCV, 2021.

[242] S. A. Koohpayegani, A. Tejankar, and H. Pirsiavash, *Mean shift for self-supervised learning*, 2021.

[243] J. Krause, M. Stark, J. Deng, and L. Fei-Fei, *3d object representations for fine-grained categorization*, in Proceedings of the IEEE international conference on computer vision workshops, 2013, pp. 554–561.

[244] J. Krause, M. Stark, J. Deng, and L. Fei-Fei, *3D object representations for fine-grained categorization*, in Workshop on 3D Representation and Recognition, Sydney, Australia, 2013.

[245] A. Krizhevsky, *Learning multiple layers of features from tiny images*, tech. rep., University of Toronto, 2009.

[246] ———, *One weird trick for parallelizing convolutional neural networks*, in ArXiv preprint arXiv:1404.5997, 2014.

[247] A. Krizhevsky, G. Hinton, et al., *Learning multiple layers of features from tiny images*, (2009).

[248] A. Krizhevsky, V. Nair, and G. Hinton, *The cifar-10 dataset*, online: http://www. cs. toronto. edu/kriz/cifar. html, 55 (2014).

[249] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *Imagenet classification with deep convolutional neural networks*, in Advances in neural information processing systems, 2012, pp. 1097–1105.

[250] H. W. Kuhn, *The hungarian method for the assignment problem*, Naval research logistics quarterly, 2 (1955), pp. 83–97.

[251] A. Kurakin, I. J. Goodfellow, and S. Bengio, *Adversarial examples in the physical world*, in Artificial intelligence safety and security, Chapman and Hall/CRC, 2018, pp. 99–112.

[252] A. Kusupati, V. Ramanujan, R. Somani, M. Wortsman, P. Jain, S. Kakade, and A. Farhadi, *Soft threshold weight reparameterization for learnable sparsity*, in Proceedings of the International Conference on Machine Learning, July 2020.

[253] S. J. Kwon, J. Kim, J. Bae, K. M. Yoo, J.-H. Kim, B. Park, B. Kim, J.-W. Ha, N. Sung, and D. Lee, *Alphatuning: Quantization-aware parameter-efficient adaptation of large-scale pre-trained language models*, arXiv preprint arXiv:2210.03858, (2022).

[254] D.-H. Lee et al., *Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks*, in Workshop on challenges in representation learning, ICML, vol. 3, 2013, p. 896.

[255] J. Lee, D. Kim, and B. Ham, *Network quantization with element-wise gradient scaling*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 6448–6457.

[256] K. Lee, Y. Zhu, K. Sohn, C.-L. Li, J. Shin, and H. Lee, *i-mix: A domain-agnostic strategy for contrastive representation learning*, in International Conference on Learning Representations, 2020.

[257] B. Lester, R. Al-Rfou, and N. Constant, *The Power of Scale for Parameter-Efficient Prompt Tuning*, arXiv:2104.08691 [cs], (2021). arXiv: 2104.08691.

[258] E. Levin and M. Fleisher, *Accelerated learning in layered neural networks*, Complex systems, 2 (1988), p. 3.

[259] A. C. Li, M. Prabhudesai, S. Duggal, E. Brown, and D. Pathak, *Your diffusion model is secretly a zero-shot classifier*, 2023.

[260] B. Li, Z. Han, H. Li, H. Fu, and C. Zhang, *Trustworthy long-tailed classification*, in CVPR, 2022, pp. 6970–6979.

[261] C. Li, H. Farkhoor, R. Liu, and J. Yosinski, *Measuring the intrinsic dimension of objective landscapes*, in International Conference on Learning Representations, 2018.

[262] D. Li, H. Ling, S. W. Kim, K. Kreis, A. Barriuso, S. Fidler, and A. Torralba, *Bigdatasetgan: Synthesizing imagenet with pixel-wise annotations*, 2022.

[263] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, *Pruning filters for efficient convnets*, arXiv preprint arXiv:1608.08710, (2016).

[264] J. Li, D. Li, C. Xiong, and S. Hoi, *Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation*, 2022.

[265] J. Li, Z. Tan, J. Wan, Z. Lei, and G. Guo, *Nested collaborative learning for long-tailed visual recognition*, in CVPR, 2022, pp. 6949–6958.

[266] K. Li, Y. Zhang, K. Li, and Y. Fu, *Adversarial feature hallucination networks for few-shot learning*, in CVPR, 2020.

[267] M. Li, Y.-m. Cheung, Y. Lu, et al., *Long-tailed visual recognition via gaussian clouded logit adjustment*, in CVPR, 2022, pp. 6929–6938.

[268] T. Li, P. Cao, Y. Yuan, L. Fan, Y. Yang, R. S. Feris, P. Indyk, and D. Katabi, *Targeted supervised contrastive learning for long-tailed recognition*, in CVPR, 2022, pp. 6918–6928.

[269] X. Li and F. Li, *Adversarial examples detection in deep networks with convolutional filter statistics*, in Proceedings of the IEEE international conference on computer vision, 2017, pp. 5764–5772.

[270] X. L. Li and P. Liang, *Prefix-Tuning: Optimizing Continuous Prompts for Generation*, arXiv:2101.00190 [cs], (2021).

[271] Y. Li, B. Ji, X. Shi, J. Zhang, B. Kang, and L. Wang, *Tea: Temporal excitation and aggregation for action recognition*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020.

[272] Y. Li, S. Lin, J. Liu, Q. Ye, M. Wang, F. Chao, F. Yang, J. Ma, Q. Tian, and R. Ji, *Towards compact cnns via collaborative compression*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 6438–6447.

[273] Z. Li, Y. Yang, X. Liu, F. Zhou, S. Wen, and W. Xu, *Dynamic computational time for visual attention*, in Proceedings of the IEEE International Conference on Computer Vision Workshops, 2017, pp. 1199–1209.

[274] Y. Liang, C. Ge, Z. Tong, Y. Song, J. Wang, and P. Xie, *Not all patches are what you need: Expediting vision transformers via token reorganizations*, in International Conference on Learning Representations (ICLR), 2022.

[275] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica, *Tune: A research platform for distributed model selection and training*, arXiv preprint arXiv:1807.05118, (2018).

[276] J. Lin, C. Gan, and S. Han, *Tsm: Temporal shift module for efficient video understanding*, in IEEE/CVF International Conference on Computer Vision (ICCV), 2019.

[277] T. Lin, S. U. Stich, L. Barba, D. Dmitriev, and M. Jaggi, *Dynamic model pruning with feedback*, arXiv preprint arXiv:2006.07253, (2020).

[278] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, *Feature pyramid networks for object detection*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 2117–2125.

[279] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, *Microsoft coco: Common objects in context*, in European conference on computer vision, Springer, 2014, pp. 740–755.

[280] Y. Lin, Y. Li, T. Liu, T. Xiao, T. Liu, and J. Zhu, *Towards fully 8-bit integer inference for the transformer model*, arXiv preprint arXiv:2009.08034, (2020).

[281] Z. Lin, A. Madotto, and P. Fung, *Exploring versatile generative language model via parameter-efficient transfer learning*, in Findings of the Association for Computational Linguistics: EMNLP 2020, Online, Nov. 2020, Association for Computational Linguistics, pp. 441–459.

[282] B. Liu, Y. Cao, Y. Lin, Q. Li, Z. Zhang, M. Long, and H. Hu, *Negative margin matters: Understanding margin in few-shot classification*, in ECCV, 2020.

[283] B. Liu, Y. Rao, J. Lu, J. Zhou, and C. jui Hsieh, *Metadistiller: Network self-boosting via meta-learned top-down distillation*, in European Conference on Computer Vision (ECCV), 2020.

[284] H. Liu, D. Tam, M. Muqeeth, J. Mohta, T. Huang, M. Bansal, and C. A. Raffel, *Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning*, Advances in Neural Information Processing Systems, 35 (2022), pp. 1950–1965.

[285] J. Liu, Z. Pan, H. He, J. Cai, and B. Zhuang, *Ecoformer: Energy-saving attention with linear complexity*, arXiv preprint arXiv:2209.09004, (2022).

[286] X. Liu, Y. Zheng, Z. Du, M. Ding, Y. Qian, Z. Yang, and J. Tang, *GPT Understands, Too*, arXiv:2103.10385 [cs], (2021). arXiv: 2103.10385.

[287] Z. Liu, S. Li, D. Wu, Z. Liu, Z. Chen, L. Wu, and S. Z. Li, *Automix: Unveiling the power of mixup for stronger classifiers*, in Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXIV, Springer, 2022, pp. 441–458.

[288] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, *Swin transformer: Hierarchical vision transformer using shifted windows*, in Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), 2021.

[289] ———, *Swin transformer: Hierarchical vision transformer using shifted windows*, in Proceedings of the IEEE/CVF international conference on computer vision, 2021, pp. 10012–10022.

[290] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, *A convnet for the 2020s*, Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), (2022).

[291] Z. Liu, Z. Miao, X. Zhan, J. Wang, B. Gong, and S. X. Yu, *Large-scale long-tailed recognition in an open world*, in CVPR, 2019.

[292] I. Loshchilov and F. Hutter, *Decoupled weight decay regularization*, arXiv preprint arXiv:1711.05101, (2017).

[293] J. Lu, J. Yao, J. Zhang, X. Zhu, H. Xu, W. Gao, C. Xu, T. Xiang, and L. Zhang, *Soft: Softmax-free transformer with linear complexity*, Advances in Neural Information Processing Systems, 34 (2021).

[294] W. Lu, J. Jiao, and R. Zhang, *Twinbert: Distilling knowledge to twin-structured bert models for efficient retrieval*, arXiv preprint arXiv:2002.06275, (2020).

[295] Y. Lu, L. Wen, J. Liu, Y. Liu, and X. Tian, *Self-supervision can be a good few-shot learner*, in European Conference on Computer Vision, Springer, 2022, pp. 740–758.

[296] X.-J. Luo, S. Wang, Z. Wu, C. Sakaridis, Y. Cheng, D.-P. Fan, and L. V. Gool, *Camdiff: Camouflage image augmentation via diffusion model*, 2023.

[297] L. Luzi, A. Siahkoohi, P. M. Mayer, J. Casco-Rodriguez, and R. Baraniuk, *Boomerang: Local sampling on image manifolds using diffusion models*, 2022.

[298] C. J. Maddison, A. Mnih, and Y. W. Teh, *The concrete distribution: A continuous relaxation of discrete random variables*, arXiv preprint arXiv:1611.00712, (2016).

[299] R. K. Mahabadi, J. Henderson, and S. Ruder, *Compacter: Efficient low-rank hypercomplex adapter layers*, 2021.

[300] K. Mahmood, R. Mahmood, and M. Van Dijk, *On the robustness of vision transformers to adversarial examples*, in Proceedings of the IEEE/CVF international conference on computer vision, 2021, pp. 7838–7847.

[301] S. Maji, E. Rahtu, J. Kannala, M. Blaschko, and A. Vedaldi, *Fine-grained visual classification of aircraft*, in arXiv preprint arXiv:1306.5151, 2013.

[302] T. Malisiewicz, A. Gupta, and A. A. Efros, *Ensemble of exemplar-svms for object detection and beyond*, in 2011 International conference on computer vision, IEEE, 2011, pp. 89–96.

[303] J. Mao, T. Xiao, Y. Jiang, and Z. Cao, *What can help pedestrian detection?*, 2017.

[304] D. Marin, J.-H. R. Chang, A. Ranjan, A. Prabhu, M. Rastegari, and O. Tuzel, *Token pooling in vision transformers*, arXiv preprint arXiv:2110.03860, (2021).

[305] D. Marin, J.-H. R. Chang, A. Ranjan, A. K. Prabhu, M. Rastegari, and O. Tuzel, *Token pooling in vision transformers*, arXiv preprint arXiv:2110.03860, (2021).

[306] N. Maslej, L. Fattorini, E. Brynjolfsson, J. Etchemendy, K. Ligett, T. Lyons, J. Manyika, H. Ngo, J. C. Niebles, V. Parli, et al., *Artificial intelligence index report 2023*, arXiv preprint arXiv:2310.03715, (2023).

[307] C. Medina, A. Devos, and M. Grossglauser, *Self-supervised prototypical transfer learning for few-shot classification*, in ICMLW, 2020.

[308] S. Mehta and M. Rastegari, *Mobilevit: Light-weight, general-purpose, and mobile-friendly vision transformer*, 2021.

[309] C. Meng, Y. He, Y. Song, J. Song, J. Wu, J.-Y. Zhu, and S. Ermon, *Sdedit: Guided image synthesis and editing with stochastic differential equations*, arXiv preprint arXiv:2108.01073, (2021).

[310] L. Meng, H. Li, B.-C. Chen, S. Lan, Z. Wu, Y.-G. Jiang, and S.-N. Lim, *Adavit: Adaptive vision transformers for efficient image recognition*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 12309–12318.

[311] A. K. Menon, S. Jayasumana, A. S. Rawat, H. Jain, A. Veit, and S. Kumar, *Long-tail learning via logit adjustment*, in ICLR, 2021.

[312] I. Mirzadeh, K. Alizadeh, S. Mehta, C. C. D. Mundo, O. Tuzel, G. Samei, M. Rastegari, and M. Farajtabar, *Relu strikes back: Exploiting activation sparsity in large language models*, 2023.

[313] I. Misra and L. van der Maaten, *Self-supervised learning of pretext-invariant representations*, arXiv preprint arXiv:1912.01991, (2019).

[314] T. Miyato, S.-i. Maeda, M. Koyama, and S. Ishii, *Virtual adversarial training: a regularization method for supervised and semi-supervised learning*, IEEE transactions on pattern analysis and machine intelligence, 41 (2018), pp. 1979–1993.

[315] R. Müller, S. Kornblith, and G. Hinton, *When does label smoothing help?*, 2020.

[316] L. Nan, D. Radev, R. Zhang, A. Rau, A. Sivaprasad, C. Hsieh, X. Tang, A. Vyas, N. Verma, P. Krishna, et al., *Dart: Open-domain structured data record to text generation*, arXiv preprint arXiv:2007.02871, (2020).

[317] K. Navaneet, S. A. Koohpayegani, E. Sleiman, and H. Pirsiavash, *Slowformer: Universal adversarial patch for attack on compute and energy efficiency of inference efficient vision transformers*, arXiv preprint arXiv:2310.02544, (2023).

[318] K. Navaneet, K. P. Meibodi, S. A. Koohpayegani, and H. Pirsiavash, *Compact3d: Compressing gaussian splat radiance field models with vector quantization*, arXiv preprint arXiv:2311.18159, (2023).

[319] K. L. Navaneet, S. A. Koohpayegani, A. Tejankar, and H. Pirsiavash, *Simreg: Regression as a simple yet effective tool for self-supervised knowledge distillation*, in BMVC, 2021.

[320] K. L. Navaneet, A. Tejankar, S. A. Koohpayegani, K. Pourahmadi, A. Subramanya, and H. Pirsiavash, *Constrained mean shift using distant yet related neighbors for representation learning*, in ECCV, 2022.

[321] M.-E. Nilsback and A. Zisserman, *Automated flower classification over a large number of classes*, in Indian Conference on Computer Vision, Graphics and Image Processing, 2008.

[322] P. Nooralinejad, A. Abbasi, S. A. Koohpayegani, K. P. Meibodi, R. M. S. Khan, S. Kolouri, and H. Pirsiavash, *Pranc: Pseudo random networks for compacting deep models*, in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2023, pp. 17021–17031.

[323] M. Noroozi and P. Favaro, *Unsupervised learning of visual representations by solving jigsaw puzzles*, in European Conference on Computer Vision, Springer, 2016, pp. 69–84.

[324] M. Noroozi, H. Pirsiavash, and P. Favaro, *Representation learning by learning to count*, in Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 5898–5906.

[325] M. Noroozi, A. Vinjimoor, P. Favaro, and H. Pirsiavash, *Boosting self-supervised learning via knowledge transfer*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 9359–9367.

[326] J. Novikova, O. Dušek, and V. Rieser, *The e2e dataset: New challenges for end-to-end generation*, arXiv preprint arXiv:1706.09254, (2017).

[327] M. Oquab, T. Darcet, T. Moutakanni, H. Vo, M. Szafraniec, V. Khalidov, P. Fernandez, D. Haziza, F. Massa, A. El-Nouby, M. Assran, N. Ballas, W. Galuba, R. Howes, P.-Y. Huang, S.-W. Li, I. Misra, M. Rabbat, V. Sharma, G. Synnaeve, H. Xu, H. Jegou, J. Mairal, P. Labatut, A. Joulin, and P. Bojanowski, *Dinov2: Learning robust visual features without supervision*, 2023.

[328] B. Pan, R. Panda, Y. Jiang, Z. Wang, R. Feris, and A. Oliva, *IA-RED^2: Interpretability-aware redundancy reduction for vision transformers*, in Advances in Neural Information Processing Systems (NeurIPS), 2021.

[329] J. Pan, Q. Zheng, Z. Fan, H. Rahmani, Q. Ke, and J. Liu, *Gradauto: Energy-oriented attack on dynamic neural networks*, in Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part IV, Springer, 2022, pp. 637–653.

[330] Z. Pan, B. Zhuang, J. Liu, H. He, and J. Cai, *Scalable vision transformers with hierarchical pooling*, in IEEE/CVF International Conference on Computer Vision (ICCV), 2021.

[331] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, *Distillation as a defense to adversarial perturbations against deep neural networks*, in 2016 IEEE symposium on security and privacy (SP), IEEE, 2016, pp. 582–597.

[332] W. Park, D. Kim, Y. Lu, and M. Cho, *Relational knowledge distillation*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 3967–3976.

[333] O. M. Parkhi, A. Vedaldi, A. Zisserman, and C. V. Jawahar, *Cats and dogs*, in Computer Vision and Pattern Recognition, 2012.

[334] N. Passalis and A. Tefas, *Learning deep representations with probabilistic knowledge transfer*, in Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 268–284.

[335] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al., *Pytorch: An imperative style, high-performance deep learning library*, in NeurIPS, 2019.

[336] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros, *Context encoders: Feature learning by inpainting*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 2536–2544.

[337] D. A. Patterson, J. Gonzalez, Q. V. Le, C. Liang, L.-M. Munguía, D. Rothchild, D. R. So, M. Texier, and J. Dean, *Carbon emissions and large neural network training*, ArXiv, abs/2104.10350 (2021).

[338] W. Peebles and S. Xie, *Scalable diffusion models with transformers*, 2023.

[339] W. Peebles, J.-Y. Zhu, R. Zhang, A. Torralba, A. Efros, and E. Shechtman, *Gan-supervised dense visual alignment*, in CVPR, 2022.

[340] B. Peng, X. Jin, J. Liu, D. Li, Y. Wu, Y. Liu, S. Zhou, and Z. Zhang, *Correlation congruence for knowledge distillation*, in Proceedings of the IEEE International Conference on Computer Vision, 2019, pp. 5007–5016.

[341] H. Peng, N. Pappas, D. Yogatama, R. Schwartz, N. A. Smith, and L. Kong, *Random feature attention*, arXiv preprint arXiv:2103.02143, (2021).

[342] S. Petryk, L. Dunlap, K. Nasseri, J. Gonzalez, T. Darrell, and A. Rohrbach, *On guiding visual attention with language specification*, in Conference on Computer Vision and Pattern Recognition (CVPR), 2022.

[343] J. Pfeiffer, A. Kamath, A. Rücklé, K. Cho, and I. Gurevych, *Adapterfusion: Non-destructive task composition for transfer learning*, 2021.

[344] H. Pham, Z. Dai, Q. Xie, and Q. V. Le, *Meta pseudo labels*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 11557–11568.

[345] V. Pillai, S. A. Koohpayegani, A. Ouligian, D. Fong, and H. Pirsiavash, *Consistent explanations by contrastive learning*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 10213–10222.

[346] V. Prabhu, S. Yenamandra, P. Chattopadhyay, and J. Hoffman, *Lance: Stress-testing visual models by generating language-guided counterfactual images*, Advances in Neural Information Processing Systems, 36 (2024).

[347] G. Prato, E. Charlaix, and M. Rezagholizadeh, *Fully quantized transformer for machine translation*, arXiv preprint arXiv:1910.10485, (2019).

[348] S. Qiao, C. Liu, W. Shen, and A. Yuille, *Few-shot image recognition by predicting parameters from activations*, in CVPR, 2018.

[349] T. Qin, W. Li, Y. Shi, and G. Yang, *Unsupervised few-shot learning via distribution shift-based augmentation*, arxiv:2004.05805, (2020).

[350] Z. Qin, W. Sun, H. Deng, D. Li, Y. Wei, B. Lv, J. Yan, L. Kong, and Y. Zhong, *cosformer: Rethinking softmax in attention*, arXiv preprint arXiv:2202.08791, (2022).

[351] Z. Qiu, T. Yao, C.-W. Ngo, X. Tian, and T. Mei, *Learning spatio-temporal representation with local and global diffusion*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019.

[352] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al., *Learning transferable visual models from natural language supervision*, in International Conference on Machine Learning, PMLR, 2021, pp. 8748–8763.

[353] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, *Learning transferable visual models from natural language supervision*, 2021.

[354] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, et al., *Improving language understanding by generative pre-training*, (2018).

[355] I. Radosavovic, R. P. Kosaraju, R. Girshick, K. He, and P. Dollár, *Designing network design spaces*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020.

[356] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen, *Hierarchical text-conditional image generation with clip latents*, arXiv preprint arXiv:2204.06125, 1 (2022), p. 3.

[357] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever, *Zero-shot text-to-image generation*, in ICML, 2021.

[358] Y. Rao, J. Lu, J. Lin, and J. Zhou, *Runtime network routing for efficient image classification*, in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 41, 2019, pp. 2291–2304.

[359] Y. Rao, W. Zhao, B. Liu, J. Lu, J. Zhou, and C.-J. Hsieh, *Dynamicvit: Efficient vision transformers with dynamic token sparsification*, in Advances in Neural Information Processing Systems (NeurIPS), 2021.

[360] ———, *Dynamicvit: Efficient vision transformers with dynamic token sparsification*, Advances in neural information processing systems, 34 (2021).

[361] Y. Rao, W. Zhao, Z. Zhu, J. Lu, and J. Zhou, *Global filter networks for image classification*, Advances in neural information processing systems, 34 (2021), pp. 980–993.

[362] ———, *Global filter networks for image classification*, in Advances in Neural Information Processing Systems (NeurIPS), 2021.

[363] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, *Xnor-net: Imagenet classification using binary convolutional neural networks*, 2016.

[364] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, *Xnor-net: Imagenet classification using binary convolutional neural networks*, in European conference on computer vision, Springer, 2016, pp. 525–542.

[365] S. Ravi and H. Larochelle, *Optimization as a model for few-shot learning*, in ICLR, 2017.

[366] S.-A. Rebuffi, H. Bilen, and A. Vedaldi, *Learning multiple visual domains with residual adapters*, arXiv:1705.08045 [cs, stat], (2017). arXiv: 1705.08045.

[367] C. J. Reed, S. Metzger, A. Srinivas, T. Darrell, and K. Keutzer, *Selfaugment: Automatic augmentation policies for self-supervised learning*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 2674–2683.

[368] M. Ren, A. Pokrovsky, B. Yang, and R. Urtasun, *Sbnet: Sparse blocks network for fast inference*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 8711–8720.

[369] M. Ren, S. Ravi, E. Triantafillou, J. Snell, K. Swersky, J. B. Tenenbaum, H. Larochelle, and R. S. Zemel, *Meta-learning for semi-supervised few-shot classification*, in International Conference on Learning Representations, 2018.

[370] S. Ren, K. He, R. Girshick, and J. Sun, *Faster r-cnn: Towards real-time object detection with region proposal networks*, in NIPS, 2015, pp. 91–99.

[371] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, *High-resolution image synthesis with latent diffusion models*, 2021.

[372] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, *High-resolution image synthesis with latent diffusion models*, in CVPR, 2022.

[373] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, *Fitnets: Hints for thin deep nets*, in 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, Y. Bengio and Y. LeCun, eds., 2015.

[374] A. Roy, M. Saffar, A. Vaswani, and D. Grangier, *Efficient content-based sparse attention with routing transformers*, in Transactions of the Association for Computational Linguistics, vol. 9, 2021, pp. 53–68.

[375] A. Roy, A. Shah, K. Shah, A. Roy, and R. Chellappa, *Cap2aug: Caption guided image to image data augmentation*, 2023.

[376] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning representations by back-propagating errors*, nature, 323 (1986), pp. 533–536.

[377] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, *ImageNet Large Scale Visual Recognition Challenge*, International Journal of Computer Vision (IJCV), 115 (2015), pp. 211–252.

[378] ———, *ImageNet Large Scale Visual Recognition Challenge*, International Journal of Computer Vision (IJCV), 115 (2015), pp. 211–252.

[379] M. S. Ryoo, A. Piergiovanni, A. Arnab, M. Dehghani, and A. Angelova, *Tokenlearner: What can 8 learned tokens do for images and videos?*, arXiv preprint arXiv:2106.11297, (2021).

[380] A. Rücklé, G. Geigle, M. Glockner, T. Beck, J. Pfeiffer, N. Reimers, and I. Gurevych, *Adapterdrop: On the efficiency of adapters in transformers*, 2020.

[381] A. Saha, A. Subramanya, K. B. Patil, and H. Pirsiavash, *Adversarial patches exploiting contextual reasoning in object detection*, in ArXiv, vol. abs/1910.00068, 2019.

[382] A. Saha, A. Tejankar, S. A. Koohpayegani, and H. Pirsiavash, *Backdoor attacks on self-supervised learning*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 13337–13346.

[383] C. Saharia, W. Chan, S. Saxena, L. Li, J. Whang, E. L. Denton, K. Ghasemipour, R. Gontijo Lopes, B. Karagol Ayan, T. Salimans, et al., *Photorealistic text-to-image diffusion models with deep language understanding*, Advances in Neural Information Processing Systems, 35 (2022), pp. 36479–36494.

[384] R. Salakhutdinov and G. Hinton, *Learning a nonlinear embedding by preserving class neighbourhood structure*, in Artificial Intelligence and Statistics, PMLR, 2007, pp. 412–419.

[385] S. Samsi, D. Zhao, J. McDonald, B. Li, A. Michaleas, M. Jones, W. Bergeron, J. Kepner, D. Tiwari, and V. Gadepally, *From words to watts: Benchmarking the energy costs of large language model inference*, 2023.

[386] D. Samuel and G. Chechik, *Distributional robustness loss for long-tail learning*, in ICCV, 2021.

[387] S. Sankaranarayanan, Y. Balaji, C. D. Castillo, and R. Chellappa, *Generate to adapt: Aligning domains using generative adversarial networks*, Conference on Computer Vision and Pattern Recognition (CVPR), (2018).

[388] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom, *Toolformer: Language models can teach themselves to use tools*, arXiv preprint arXiv:2302.04761, (2023).

[389] F. Schroff, D. Kalenichenko, and J. Philbin, *Facenet: A unified embedding for face recognition and clustering*, in CVPR, 2015, pp. 815–823.

[390] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, *Grad-cam: Visual explanations from deep networks via gradient-based localization*, in Proceedings of the IEEE international conference on computer vision, 2017, pp. 618–626.

[391] V. Sharmanska, L. A. Hendricks, T. Darrell, and N. Quadrianto, *Contrastive examples for addressing the tyranny of the majority*, CoRR, abs/2004.06524 (2020).

[392] Z. Shen, M. Zhang, H. Zhao, S. Yi, and H. Li, *Efficient attention: Attention with linear complexities*, in Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, 2021, pp. 3531–3539.

[393] J. Shipard, A. Wiliem, K. N. Thanh, W. Xiang, and C. Fookes, *Boosting zero-shot classification with synthetic data diversity via stable diffusion*, arXiv preprint arXiv:2302.03298, (2023).

[394] O. K. Shirekar, A. Singh, and H. Jamali-Rad, *Self-attention message passing for contrastive few-shot learning*, in Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV), January 2023, pp. 5426–5436.

[395] C. Shorten and T. M. Khoshgoftaar, *A survey on image data augmentation for deep learning*, Journal of big data, 6 (2019), pp. 1–48.

[396] J. N. Siems, A. Klein, C. Archambeau, and M. Mahsereci, *Dynamic pruning of a neural network via gradient signal-to-noise ratio*, in 8th ICML Workshop on Automated Machine Learning (AutoML), 2021.

[397] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, arXiv preprint arXiv:1409.1556, (2014).

[398] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, arXiv preprint arXiv:1409.1556, (2015).

[399] A. R. Singh and H. Jamali-Rad, *Transductive decoupled variational inference for few-shot classification*, Transactions on Machine Learning Research, (2023).

[400] J. Snell, K. Swersky, and R. Zemel, *Prototypical networks for few-shot learning*, in Advances in Neural Information Processing Systems, 2017.

[401] J. Snell, K. Swersky, and R. S. Zemel, *Prototypical networks for few-shot learning*, arXiv preprint arXiv:1703.05175, (2017).

[402] K. Sohn, *Improved deep metric learning with multi-class n-pair loss objective*, in Proceedings of the 30th International Conference on Neural Information Processing Systems, 2016, pp. 1857–1865.

[403] K. Sohn, D. Berthelot, N. Carlini, Z. Zhang, H. Zhang, C. A. Raffel, E. D. Cubuk, A. Kurakin, and C.-L. Li, *Fixmatch: Simplifying semi-supervised learning with consistency and confidence*, Advances in Neural Information Processing Systems, 33 (2020).

[404] K. Soomro, A. R. Zamir, and M. Shah, *Ucf101: A dataset of 101 human actions classes from videos in the wild*, 2012.

[405] J. R. Stevens, R. Venkatesan, S. Dai, B. Khailany, and A. Raghunathan, *Softermax: Hardware/software co-design of an efficient softmax for transformers*, in 2021 58th ACM/IEEE Design Automation Conference (DAC), IEEE, 2021, pp. 469–474.

[406] J.-C. Su, S. Maji, and B. Hariharan, *When does self-supervision improve few-shot learning?*, in ECCV, 2020.

[407] A. Subramanya, S. A. Koohpayegani, A. Saha, A. Tejankar, and H. Pirsiavash, *A closer look at robustness of vision transformers to backdoor attacks*, in Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, 2024, pp. 3874–3883.

[408] S. Sukhbaatar, J. Bruna, M. Paluri, L. Bourdev, and R. Fergus, *Training convolutional networks with noisy labels*, 2015.

[409] S. Sukhbaatar, E. Grave, P. Bojanowski, and A. Joulin, *Adaptive attention span in transformers*, in ACL, 2019.

[410] F. Sung, Y. Yang, L. Zhang, T. Xiang, P. H. Torr, and T. M. Hospedales, *Learning to compare: Relation network for few-shot learning*, in CVPR, 2018.

[411] Y.-L. Sung, J. Cho, and M. Bansal, *Lst: Ladder side-tuning for parameter and memory efficient transfer learning*, Advances in Neural Information Processing Systems, 35 (2022), pp. 12991–13005.

[412] ———, *Vl-adapter: Parameter-efficient transfer learning for vision-and-language tasks*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 5227–5237.

[413] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, *Rethinking the inception architecture for computer vision*, 2015.

[414] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, *Intriguing properties of neural networks*, arXiv preprint arXiv:1312.6199, (2013).

[415] M. Tan and Q. Le, *EfficientNet: Rethinking model scaling for convolutional neural networks*, in International Conference on Machine Learning (ICML), 2019.

[416] K. Tang, J. Huang, and H. Zhang, *Long-tailed classification by keeping the good and removing the bad momentum causal effect*, NeurIPS, 33 (2020), pp. 1513–1524.

[417] R. Taori, I. Gulrajani, T. Zhang, Y. Dubois, X. Li, C. Guestrin, P. Liang, and T. B. Hashimoto, *Stanford alpaca: An instruction-following llama model*. https://github.com/tatsu-lab/stanford_alpaca, 2023.

[418] A. Tarvainen and H. Valpola, *Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results*, in Advances in neural information processing systems, 2017, pp. 1195–1204.

[419] S. Teerapittayanon, B. McDanel, and H.-T. Kung, *Branchynet: Fast inference via early exiting from deep neural networks*, in 2016 23rd International Conference on Pattern Recognition (ICPR), IEEE, 2016, pp. 2464–2469.

[420] A. Tejankar, S. A. Koohpayegani, V. Pillai, P. Favaro, and H. Pirsiavash, *Isd: Self-supervised learning by iterative similarity distillation*, 2020.

[421] A. Tejankar, S. A. Koohpayegani, V. Pillai, P. Favaro, and H. Pirsiavash, *Isd: Self-supervised learning by iterative similarity distillation*, in ICCV, 2021.

[422] C. Thrash, A. Abbasi, P. Nooralinejad, S. A. Koohpayegani, R. Andreas, H. Pirsiavash, and S. Kolouri, *Mcnc: Manifold constrained network compression*, 2024.

[423] C. Tian, W. Wang, X. Zhu, J. Dai, and Y. Qiao, *Vl-ltr: Learning class-wise visual-linguistic representation for long-tailed visual recognition*, in ECCV 2022, 2022.

[424] Y. Tian, D. Krishnan, and P. Isola, *Contrastive multiview coding*, arXiv preprint arXiv:1906.05849, (2019).

[425] Y. Tian, D. Krishnan, and P. Isola, *Contrastive representation distillation*, in International Conference on Learning Representations, 2020.

[426] Y. Tian, C. Sun, B. Poole, D. Krishnan, C. Schmid, and P. Isola, *What makes for good views for contrastive learning*, arXiv preprint arXiv:2005.10243, (2020).

[427] R. Tiwari, U. Bamba, A. Chavan, and D. K. Gupta, *Chipnet: Budget-aware pruning with heaviside continuous approximations*, arXiv preprint arXiv:2102.07156, (2021).

[428] I. Tolstikhin, N. Houlsby, A. Kolesnikov, L. Beyer, X. Zhai, T. Unterthiner, J. Yung, D. Keysers, J. Uszkoreit, M. Lucic, et al., *Mlp-mixer: An all-mlp architecture for vision*, in arXiv preprint arXiv:2105.01601, 2021.

[429] H. Touvron, P. Bojanowski, M. Caron, M. Cord, A. El-Nouby, E. Grave, A. Joulin, G. Synnaeve, J. Verbeek, and H. Jégou, *Resmlp: Feedforward networks for image classification with data-efficient training*, in arXiv preprint arXiv:2105.03404, 2021.

[430] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, *Training data-efficient image transformers & distillation through attention*, in International Conference on Machine Learning (ICML), 2021.

[431] ———, *Training data-efficient image transformers & distillation through attention*, in International Conference on Machine Learning, PMLR, 2021, pp. 10347–10357.

[432] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jegou, *Training data-efficient image transformers and distillation through attention*, in International Conference on Machine Learning (ICML), 2021.

[433] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, *Training data-efficient image transformers and distillation through attention*, 2021.

[434] H. Touvron, M. Cord, and H. Jégou, *Deit iii: Revenge of the vit*, in ECCV, 2022.

[435] H. Touvron, M. Cord, A. Sablayrolles, G. Synnaeve, and H. Jégou, *Going deeper with image transformers*, in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021, pp. 32–42.

[436] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom, *Llama 2: Open foundation and fine-tuned chat models*, 2023.

[437] H. Touvron, A. Sablayrolles, M. Douze, M. Cord, and H. Jégou, *Grafit: Learning fine-grained image representations with coarse labels*, 2020.

[438] B. Trabucco, K. Doherty, M. A. Gurinas, and R. Salakhutdinov, *Effective data augmentation with diffusion models*, in The Twelfth International Conference on Learning Representations, 2024.

[439] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, *Learning spatiotemporal features with 3d convolutional networks*, in IEEE International Conference on Computer Vision (ICCV), 2015.

[440] D. Tran, H. Wang, L. Torresani, and M. Feiszli, *Video classification with channel-separated convolutional networks*, in IEEE/CVF International Conference on Computer Vision (ICCV), 2019.

[441] D. Tran, H. Wang, L. Torresani, J. Ray, Y. LeCun, and M. Paluri, *A closer look at spatiotemporal convolutions for action recognition*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018.

[442] N. Tritrong, P. Rewatbowornwong, and S. Suwajanakorn, *Repurposing gans for one-shot semantic part segmentation*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2021.

[443] Y.-H. H. Tsai, T. Li, W. Liu, P. Liao, R. Salakhutdinov, and L.-P. Morency, *Integrating auxiliary information in self-supervised learning*, 2021.

[444] F. Tung and G. Mori, *Similarity-preserving knowledge distillation*, in Proceedings of the IEEE International Conference on Computer Vision, 2019, pp. 1365–1374.

[445] A. van den Oord, Y. Li, and O. Vinyals, *Representation learning with contrastive predictive coding*, 2018.

[446] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, *Attention is all you need*, in arXiv preprint arXiv:1706.03762, 2017.

[447] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, *Attention is all you need*, in Advances in Neural Information Processing Systems (NeuRIPS), 2017.

[448] I. Vasyltsov and W. Chang, *Efficient softmax approximation for deep neural networks with attention mechanism*, arXiv preprint arXiv:2111.10770, (2021).

[449] A. Veit and S. Belongie, *Convolutional networks with adaptive inference graphs*, in Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 3–18.

[450] T. Verelst and T. Tuytelaars, *Dynamic convolutions: Exploiting spatial sparsity for faster inference*, in Proceedings of the ieee/cvf conference on computer vision and pattern recognition, 2020, pp. 2320–2329.

[451] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra, *Matching networks for one shot learning*, 2017.

[452] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie, *The caltech-ucsd birds-200-2011 dataset*, 2011.

[453] F. Wang, H. Liu, D. Guo, and S. Fuchun, *Unsupervised representation learning by invariance propagation*, in Advances in Neural Information Processing Systems, 2020.

[454] G. Wang, K. Wang, G. Wang, P. H. S. Torr, and L. Lin, *Solving inefficiency of self-supervised representation learning*, 2021.

[455] H. Wang and Z.-H. Deng, *Contrastive prototypical network with wasserstein confidence penalty*, in European Conference on Computer Vision, Springer, 2022, pp. 665–682.

[456] H. Wang, S. Fu, X. He, H. Fang, Z. Liu, and H. Hu, *Towards calibrated hyper-sphere representation via distribution overlap coefficient for long-tailed learning*, in ECCV, 2022.

[457] H. Wang, C. Qin, Y. Zhang, and Y. Fu, *Neural pruning via growing regularization*, arXiv preprint arXiv:2012.09243, (2020).

[458] H. Wang, D. Tran, L. Torresani, and M. Feiszli, *Video modeling with correlation networks*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020.

[459] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, *Haq: Hardware-aware automated quantization with mixed precision*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2019.

[460] S. Wang, B. Z. Li, M. Khabsa, H. Fang, and H. Ma, *Linformer: Self-attention with linear complexity*, arXiv preprint arXiv:2006.04768, (2020).

[461] W. Wang, E. Xie, X. Li, D.-P. Fan, K. Song, D. Liang, T. Lu, P. Luo, and L. Shao, *Pyramid vision transformer: A versatile backbone for dense prediction without convolutions*, in IEEE/CVF International Conference on Computer Vision (ICCV), 2021.

[462] W. Wang, E. Xie, X. Li, D.-P. Fan, K. Song, D. Liang, T. Lu, P. Luo, and L. Shao, *Pyramid vision transformer: A versatile backbone for dense prediction without convolutions*, in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021, pp. 568–578.

[463] X. Wang, R. Girshick, A. Gupta, and K. He, *Non-local neural networks*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2018.

[464] X. Wang, L. Lian, Z. Miao, Z. Liu, and S. X. Yu, *Long-tailed recognition by routing diverse distribution-aware experts*, in ICLR, OpenReview.net, 2021.

[465] X. Wang, Z. Liu, and S. X. Yu, *Unsupervised feature learning by cross-level instance-group discrimination*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), June 2021, pp. 12586–12595.

[466] X. Wang, X. Xiong, M. Neumann, A. J. Piergiovanni, M. S. Ryoo, A. Angelova, K. M. Kitani, and W. Hua, *Attentionnas: Spatiotemporal attention cell search for video classification*, in European Conference on Computer Vision (ECCV), 2020.

[467] X. Wang, F. Yu, Z.-Y. Dou, T. Darrell, and J. E. Gonzalez, *Skipnet: Learning dynamic routing in convolutional networks*, in Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 409–424.

[468] C. Wei, H. Wang, W. Shen, and A. Yuille, *Co2: Consistent contrast for unsupervised visual representation learning*, arXiv preprint arXiv:2010.02217, (2020).

[469] Z. Wei, J. Chen, M. Goldblum, Z. Wu, T. Goldstein, and Y.-G. Jiang, *Towards transferable adversarial attacks on vision transformers*, in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 36, 2022, pp. 2668–2676.

[470] K. Q. Weinberger, J. Blitzer, and L. K. Saul, *Distance metric learning for large margin nearest neighbor classification*, in Advances in neural information processing systems, 2006, pp. 1473–1480.

[471] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona, *Caltech-ucsd birds 200*, (2010).

[472] R. Whitman, *Timm*. https://github.com/huggingface/pytorch-image-models/tree/main/timm. [Online; accessed 28-Sep-2023].

[473] R. Wightman, *Pytorch image models*, 2019.

[474] A. Wu, W.-S. Zheng, X. Guo, and J.-H. Lai, *Distilled person re-identification: Towards a more scalable system*, in The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2019.

[475] C.-Y. Wu, R. Manmatha, A. J. Smola, and P. Krahenbuhl, *Sampling matters in deep embedding learning*, in Proceedings of the IEEE International Conference on Computer Vision (ICCV), Oct 2017.

[476] H. Wu, B. Xiao, N. Codella, M. Liu, X. Dai, L. Yuan, and L. Zhang, *Cvt: Introducing convolutions to vision transformers*, in IEEE/CVF International Conference on Computer Vision (ICCV), 2021.

[477] ———, *Cvt: Introducing convolutions to vision transformers*, in Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), October 2021, pp. 22–31.

[478] ———, *Cvt: Introducing convolutions to vision transformers*, in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021, pp. 22–31.

[479] Z. Wu, A. A. Efros, and S. X. Yu, *Improving generalization via scalable neighborhood component analysis*, 2018.

[480] Z. Wu, Y. Xiong, S. X. Yu, and D. Lin, *Unsupervised feature learning via non-parametric instance discrimination*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 3733–3742.

[481] J. Xiao, J. Hays, K. A. Ehinger, A. Oliva, and A. Torralba, *Sun database: Large-scale scene recognition from abbey to zoo*, in Computer Vision and Pattern Recognition, 2010.

[482] C. Xie, J. Wang, Z. Zhang, Y. Zhou, L. Xie, and A. Yuille, *Adversarial examples for semantic segmentation and object detection*, in Proceedings of the IEEE international conference on computer vision, 2017, pp. 1369–1378.

[483] J. Xie, R. Girshick, and A. Farhadi, *Unsupervised deep embedding for clustering analysis*, in Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48, ICML'16, JMLR.org, 2016, p. 478–487.

[484] Q. Xie, Z. Dai, E. Hovy, M.-T. Luong, and Q. V. Le, *Unsupervised data augmentation for consistency training*, NeurIPS, (2020).

[485] S. Xie, C. Sun, J. Huang, Z. Tu, and K. Murphy, *Rethinking spatiotemporal feature learning: Speed-accuracy trade-offs in video classification*, 2018.

[486] Z. Xie, Y. Lin, Z. Yao, Z. Zhang, Q. Dai, Y. Cao, and H. Hu, *Self-supervised learning with swin transformers*, arXiv preprint arXiv:2105.04553, (2021).

[487] Z. Xie, Z. Zhang, X. Zhu, G. Huang, and S. Lin, *Spatially adaptive inference with stochastic feature sampling and interpolation*, in Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I 16, Springer, 2020, pp. 531–548.

[488] Y. Xiong, Z. Zeng, R. Chakraborty, M. Tan, G. Fung, Y. Li, and V. Singh, *Nyströmformer: A nyström-based algorithm for approximating self-attention*, (2021).

[489] W. Xu, Y. Xu, T. Chang, and Z. Tu, *Co-scale conv-attentional image transformers*, in arXiv preprint arXiv:2104.06399, 2021.

[490] Y. Xu, Y.-L. Li, J. Li, and C. Lu, *Constructing balance from imbalance for long-tailed image recognition*, in ECCV, Springer, 2022, pp. 38–56.

[491] Y. Xu, Q. Qian, H. Li, R. Jin, and J. Hu, *Weakly supervised representation learning with coarse labels*, 2021.

[492] Y. Xu, L. Xie, X. Gu, X. Chen, H. Chang, H. Zhang, Z. Chen, X. Zhang, and Q. Tian, *Qa-lora: Quantization-aware low-rank adaptation of large language models*, 2023.

[493] Z. Xu, R. Liu, S. Yang, Z. Chai, and C. Yuan, *Learning imbalanced data with vision transformers*, 2023.

[494] H. Xuan, A. Stylianou, X. Liu, and R. Pless, *Hard negative examples are hard, but useful*, 2021.

[495] X. Yan, I. Misra, A. Gupta, D. Ghadiyaram, and D. Mahajan, *Clusterfit: Improving generalization of visual representations*, in CVPR, 2020.

[496] J. Yang, D. Parikh, and D. Batra, *Joint unsupervised learning of deep representations and image clusters*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[497] L. Yang, Y. Han, X. Chen, S. Song, J. Dai, and G. Huang, *Resolution adaptive networks for efficient inference*, in Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2020, pp. 2369–2378.

[498] Z. Yang, Y. Xu, W. Dai, and H. Xiong, *Dynamic-stride-net: Deep convolutional neural network with dynamic stride*, in Optoelectronic Imaging and Multimedia Technology VI, vol. 11187, SPIE, 2019, pp. 42–53.

[499] H.-J. Ye, H. Hu, D.-C. Zhan, and F. Sha, *Few-shot learning via embedding adaptation with set-to-set functions*, in CVPR, 2020.

[500] J. Yim, D. Joo, J. Bae, and J. Kim, *A gift from knowledge distillation: Fast optimization, network minimization and transfer learning*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 4133–4141.

[501] H. Yin, A. Vahdat, J. M. Alvarez, A. Mallya, J. Kautz, and P. Molchanov, *A-vit: Adaptive tokens for efficient vision transformer*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 10809–10818.

[502] A. YM., R. C., and V. A., *Self-labelling via simultaneous clustering and representation learning*, in International Conference on Learning Representations, 2020.

[503] L. Yu, V. O. Yazici, X. Liu, J. v. d. Weijer, Y. Cheng, and A. Ramisa, *Learning metrics from teachers: Compact networks for image embedding*, in The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2019.

[504] W. Yu, M. Luo, P. Zhou, C. Si, Y. Zhou, X. Wang, J. Feng, and S. Yan, *Metaformer is actually what you need for vision*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 10819–10829.

[505] X. Yu, T. Liu, X. Wang, and D. Tao, *On compressing deep models by low rank and sparse decomposition*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2017.

[506] X. Yu, Y. Rao, Z. Wang, Z. Liu, J. Lu, and J. Zhou, *Pointr: Diverse point cloud completion with geometry-aware transformers*, in ICCV, 2021.

[507] ———, *Pointr: Diverse point cloud completion with geometry-aware transformers*, in IEEE/CVF International Conference on Computer Vision (ICCV), 2021.

[508] L. Yuan, Y. Chen, T. Wang, W. Yu, Y. Shi, Z. Jiang, F. E. Tay, J. Feng, and S. Yan, *Tokens-to-token vit: Training vision transformers from scratch on imagenet*, in arXiv preprint arXiv:2101.11986, 2021.

[509] L. Yuan, Y. Chen, T. Wang, W. Yu, Y. Shi, Z.-H. Jiang, F. E. Tay, J. Feng, and S. Yan, *Tokens-to-token vit: Training vision transformers from scratch on imagenet*, in Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), October 2021, pp. 558–567.

[510] Z. Yuan, B. Wu, G. Sun, Z. Liang, S. Zhao, and W. Bi, *S2dnas: Transforming static cnn model for dynamic inference via neural architecture search*, in Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16, Springer, 2020, pp. 175–192.

[511] X. Yue, S. Sun, Z. Kuang, M. Wei, P. Torr, W. Zhang, and D. Lin, *Vision transformer with progressive sampling*, in IEEE/CVF International Conference on Computer Vision (ICCV), 2021.

[512] S. Yun, D. Han, S. J. Oh, S. Chun, J. Choe, and Y. Yoo, *Cutmix: Regularization strategy to train strong classifiers with localizable features*, in Proceedings of the IEEE/CVF international conference on computer vision, 2019, pp. 6023–6032.

[513] ———, *Cutmix: Regularization strategy to train strong classifiers with localizable features*, in ICCV, 2019, pp. 6023–6032.

[514] C. Zach, T. Pock, and H. Bischof, *A duality based approach for realtime tv-l1 optical flow*, in DAGM-Symposium, 2007.

[515] O. Zafrir, G. Boudoukh, P. Izsak, and M. Wasserblat, *Q8bert: Quantized 8bit bert*, in 2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Edition (EMC2-NIPS), IEEE, 2019, pp. 36–39.

[516] S. Zagoruyko and N. Komodakis, *Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer*, in ICLR, 2017.

[517] E. B. Zaken, S. Ravfogel, and Y. Goldberg, *Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models*, arXiv preprint arXiv:2106.10199, (2021).

[518] X. Zhai, A. Oliver, A. Kolesnikov, and L. Beyer, *S4l: Self-supervised semi-supervised learning*, in The IEEE International Conference on Computer Vision (ICCV), October 2019.

[519] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, *mixup: Beyond empirical risk minimization*, arXiv preprint arXiv:1710.09412, (2017).

[520] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, *mixup: Beyond empirical risk minimization*, 2018.

[521] ———, *mixup: Beyond empirical risk minimization*, in ICLR, 2018.

[522] J. Zhang, Y. Huang, W. Wu, and M. R. Lyu, *Transferable adversarial attacks on vision transformers with token gradient regularization*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2023, pp. 16415–16424.

[523] L. Zhang, G.-J. Qi, L. Wang, and J. Luo, *Aet vs. aed: Unsupervised representation learning by auto-encoding transformations rather than data*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 2547–2555.

[524] P. Zhang, X. Dai, J. Yang, B. Xiao, L. Yuan, L. Zhang, and J. Gao, *Multi-scale vision longformer: A new vision transformer for high-resolution image encoding*, in Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), October 2021, pp. 2998–3008.

[525] Q. Zhang, M. Chen, A. Bukharin, P. He, Y. Cheng, W. Chen, and T. Zhao, *Adaptive budget allocation for parameter-efficient fine-tuning*, 2023.

[526] R. Zhang, P. Isola, and A. A. Efros, *Colorful image colorization*, in European conference on computer vision, Springer, 2016, pp. 649–666.

[527] ———, *Split-brain autoencoders: Unsupervised learning by cross-channel prediction*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 1058–1067.

[528] S. Zhang, Z. Li, S. Yan, X. He, and J. Sun, *Distribution alignment: A unified framework for long-tail visual recognition*, in CVPR, 2021, pp. 2361–2370.

[529] Y. Zhang, B. Hooi, L. Hong, and J. Feng, *Test-agnostic long-tailed recognition by test-time aggregating diverse experts with self-supervision*, arXiv preprint arXiv:2107.09249, (2021).

[530] Y. Zhang, H. Ling, J. Gao, K. Yin, J.-F. Lafleche, A. Barriuso, A. Torralba, and S. Fidler, *Datasetgan: Efficient labeled data factory with minimal human effort*, in CVPR, 2021.

[531] Y. Zhang, K. Zhou, and Z. Liu, *Neural prompt search*, 2022.

[532] Z. Zhang and M. R. Sabuncu, *Generalized cross entropy loss for training deep neural networks with noisy labels*, arXiv preprint arXiv:1805.07836, (2018).

[533] H. Zhao, L. Jiang, J. Jia, P. Torr, and V. Koltun, *Point transformer*, in IEEE/CVF International Conference on Computer Vision (ICCV), 2021.

[534] H. Zhao, L. Jiang, J. Jia, P. H. Torr, and V. Koltun, *Point transformer*, in Proceedings of the IEEE/CVF international conference on computer vision, 2021, pp. 16259–16268.

[535] H. Zheng, P. He, W. Chen, and M. Zhou, *Mixing and shifting: Exploiting global and local dependencies in vision mlps*, arXiv preprint arXiv:2202.06510, (2022).

[536] S. Zheng, J. Lu, H. Zhao, X. Zhu, Z. Luo, Y. Wang, Y. Fu, J. Feng, T. Xiang, P. H. Torr, and L. Zhang, *Rethinking semantic segmentation from a sequence-to-sequence perspective with transformers*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2021.

[537] Z. Zhong, J. Cui, S. Liu, and J. Jia, *Improving calibration for long-tailed recognition*, in CVPR, Computer Vision Foundation / IEEE, 2021, pp. 16489–16498.

[538] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, *Learning deep features for discriminative localization*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 2921–2929.

[539] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva, *Learning deep features for scene recognition using places database*, in Advances in neural information processing systems, 2014, pp. 487–495.

[540] D. Zhou, B. Kang, X. Jin, L. Yang, X. Lian, Q. Hou, and J. Feng, *Deepvit: Towards deeper vision transformer*, in arXiv preprint arXiv:2103.11886, 2021.

[541] D. Zhou, B. Kang, X. Jin, L. Yang, X. Lian, Z. Jiang, Q. Hou, and J. Feng, *Deepvit: Towards deeper vision transformer*, arXiv preprint arXiv:2103.11886, (2021).

[542] Z. Zhou, X. Qiu, J. Xie, J. Wu, and C. Zhang, *Binocular mutual learning for improving few-shot classification*, in ICCV, 2021.

[543] D. Zhu, S. Lu, M. Wang, J. Lin, and Z. Wang, *Efficient precision-adjustable architecture for softmax function in deep learning*, IEEE Transactions on Circuits and Systems II: Express Briefs, 67 (2020), pp. 3382–3386.

[544] J. Zhu, Z. Wang, J. Chen, Y.-P. P. Chen, and Y.-G. Jiang, *Balanced contrastive learning for long-tailed visual recognition*, in CVPR, 2022, pp. 6908–6917.

[545] C. Zhuang, A. L. Zhai, and D. Yamins, *Local aggregation for unsupervised learning of visual embeddings*, in Proceedings of the IEEE International Conference on Computer Vision, 2019, pp. 6002–6012.

APPENDIX A

Appendix

A.1 ATS: Adaptive Token Sampling For Vision Transformers

A.1.1 Runtime

Throughput: While ATS is a super-light module, there is still a small cost associated with I/O operations. For a DeiT-S network with a single ATS stage, the sampling overhead is about 1.5% of the overall computation which is negligible compared to the large savings due to the dropped tokens. To further elaborate on this, we have reported the throughput (images/s) of the DeiT-S model with/without our ATS module in Table A.1. As it can be seen, the speed-up of our module is aligned with its GFLOPs reduction.

Batch Processing: While for most applications the inference is performed for a single image or video, ATS can also be used for inference with a mini-batch. To this end, we rearrange the tokens of each image so that the sampled tokens are in the lower indices. Then, we remove the last tokens completely to reduce the computation. This way, we only process m tokens, where $m = \max_i(K'_i + 1)$ over all images i of the mini-batch. In the worst case scenario (*e.g.* a very large minibatch), we will keep all $K + 1$ first tokens after rearrangement. This will still reduce the computation by a factor of $\frac{N+1}{K+1}$. For example, using a mini-batch of size 512 on the ImageNet validation set, m is 129 in Stage 7 of the DeiT-S+ATS model, which is smaller than the total number of tokens (197). Therefore, we discard at least 68 tokens in stage 7 even in a mini-batch setting. Moreover, for the fully connected layers in a transformer block, which requires most of the computation [305], we can flatten the mini-batch dimension and forward only non-zero tokens of the whole mini-batch in parallel through the fully connected layers.

A.1.2 The Effect of K

In Fig 2.5a, we varied the value of K to achieve different GFLOPs levels (Top-1 Accuracy vs. GFLOPs). In Fig. A.1, we study the effect of varying K in the ATS module of the single-stage DeiTS+ATS model with fine-tuning. Interestingly, even sampling only 48 tokens (2 GFLOPs) achieves 75% accuracy.

TABLE A.1. **Runtime comparison:** We run the models on a single RTX6000 GPU (CUDA 11.0, PyTorch 1.8, image size: 224×224). We average the value of throughput over 20 runs. We add ATS to multiple stages of the DeiT-S model and fine-tune the network on the ImageNet dataset.

Model	Params (M)	GFLOPs	Throughput	Top-1
Deit-S [431]	22.05	4.6	1010	79.8
Deit-S+ATS	22.05	**2.9**	**1403**	79.7

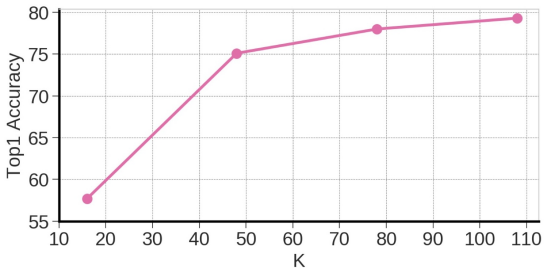


FIGURE A.1. **Effect of K:** We varied the value of K in the ATS module to study the effect of K on the top-1 accuracy. K=48 corresponds to 2 GFLOPs. The backbone model is DeiT-S pre-trained on ImageNet-1K.

A.1.3 ATS Integration Without Further Training

One of the most important aspects of our approach is that it can be added to any pre-trained off-the-shelf vision transformer. For example, our not fine-tuned multi-stage DeiT-S+ATS model (Fig 2.5c) has only a 0.6% (Table 2.1) top-1 accuracy drop while it has improved the efficiency by about 1.6 GFLOPs without any further training of the backbone model. We also observe the same performance on video data. As reported in Table A.2, our not fine-tuned XViT+ATS model has only a 1.1% top-1 accuracy drop while it has improved the efficiency by about 329 GFLOPs without any further training of the backbone model. This capability of our ATS module roots back in its adaptive inverse transform sampling strategy. Our ATS module samples informative tokens based on their contributions to the classification token. Uninformative tokens that only slightly contribute to the final prediction receive lower attention weights for the classification token. Therefore, the output classification token will be only marginally affected by removing such redundant tokens. On the other hand, the redundant tokens are less similar to the informative tokens and receive lower attention weights for those tokens in the attention matrix. Consequently, they do not contribute much to the value of informative tokens and eliminating them does not change the way informative tokens are contributing to the output classification token.

TABLE A.2. Our ATS module is added to XViT [48] pre-trained on Kinetics-600.

Model	Top-1	GFLOPs
XViT+ATS Not-Finetuned(16×)	83.4	**521**
XViT+ATS Finetuned(16×)	84.4	**521**
XViT(16×)	**84.5**	850

A.1.4 Attention Map Visualization

As shown in Fig. A.2, the attention maps become more focused on the birds and less on the background at the later stages, which is aligned with our observations on the sampled tokens at each stage.

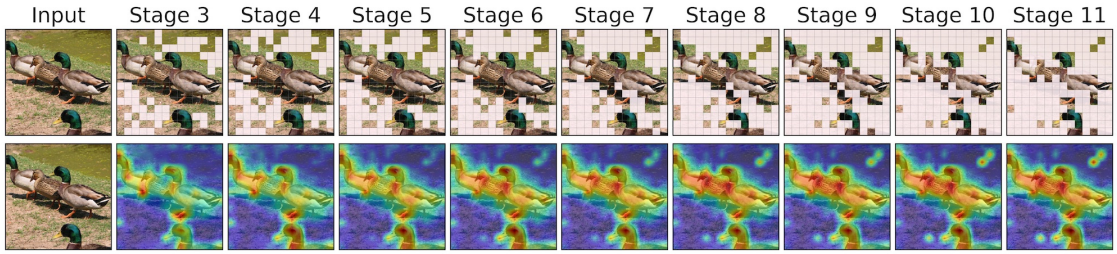


FIGURE A.2. Visualization of the sampled tokens and attention maps of a not fine-tuned multi-stage DeiT-S+ATS.

A.1.5 Implementation Details

In our experiments for image classification, we use the ImageNet [101] dataset with 1.28M training images and 1K classes. We evaluate our adaptive models, which are equipped with the ATS module, on 50K validation images of this dataset. In our experiments for action recognition, we use the Kinetics-400 [226] and Kinetics-600 [60] datasets containing short clips (typically 10 seconds long) sampled from YouTube. Kinetics-400 and Kinetics-600 consist of 400 and 600 classes, respectively. The versions of Kinetics-400 and Kinetics-600 used in our experiments consist of approximately 261k and 457k clips, respectively. Note that these numbers are lower than the original datasets due to the removal of certain videos from YouTube. Our networks for image classification are trained on 8 NVIDIA Quadro RTX 6000 GPUs and for action recognition on 8 NVIDIA A100 GPUs.

A.1.5.1 DeiT + ATS

Training To fine-tune our adaptive models, we follow the DynamicViT [359] training settings. We use the DeiT model's pre-trained weights to initialize the backbones of our adaptive network and train it for 30 epochs using AdamW optimizer. The learning rate and batch size are set to 5e-4 and 8×96, respectively. We use the cosine scheduler to train the networks. For both multi and single stage models, we set $K = 197$ during training.

Evaluation We use the same setup as [431] for evaluating our adaptive models. To evaluate the performance of our multi-stage DeiT-S+ATS model with different average GFLOPs levels of 3, 2.5, and 2, we set $K_n = \max(\lfloor \rho \times \#InputTokens_n \rceil, 8)$ in which ρ is set to 1, 0.87, 0.72, respectively, and n is the stage index. For

the single-stage model, we set K = 108, 78, 48 to evaluate the model with different average GFLOPs levels of 3, 2.5, and 2.

A.1.5.2 CvT + ATS

We integrate our ATS module into the 1^{st} to 9^{th} blocks of the 3^{rd} stage of the CvT-13 [476] and CvT-21 [476] networks. For both CvT models, we do not use any convolutional projection layers in the transformer blocks of stage 3.

Training To train our adaptive models, we follow most of the CvT [476] network's training settings. We use the CvT model's pre-trained weights to initialize the backbones of our adaptive networks and train them for 30 epochs using AdamW optimizer. The learning rate and batch size are set to 1.5e-6 and 128, respectively. We use the cosine scheduler to train the networks.

Evaluation To evaluate our CvT+ATS model, we use the same setup as [476].

A.1.5.3 PS-ViT + ATS

Training To fine-tune our adaptive models, we follow the PS-ViT [511] training settings. We use the PS-ViT model's pre-trained weights to initialize the backbones of our adaptive network and train it for 30 epochs using AdamW optimizer. The learning rate and batch size are set to 5e-4 and 8×96, respectively. We use the cosine scheduler to train the networks.

Evaluation To evaluate our CvT+ATS model, we use the same setup as [511].

A.1.5.4 XViT + ATS

We integrate our ATS module into the stages 3 to 11 of the XViT [48] network.

Training To train our adaptive model, we follow most of the XViT [48] network's training settings. We use the XViT model's pre-trained weights to initialize the backbone of our adaptive network and train it for 10 epochs using SGD optimizer. The learning rate and batch size are set to 1.5e-6 and 64, respectively. We use the cosine scheduler to train the networks.

Evaluation To evaluate our XViT+ATS model, we use the same setup as [48].

A.1.5.5 TimeSformer + ATS

We integrate our ATS module into the stages 3 to 5 of the TimeSformer [38] network.

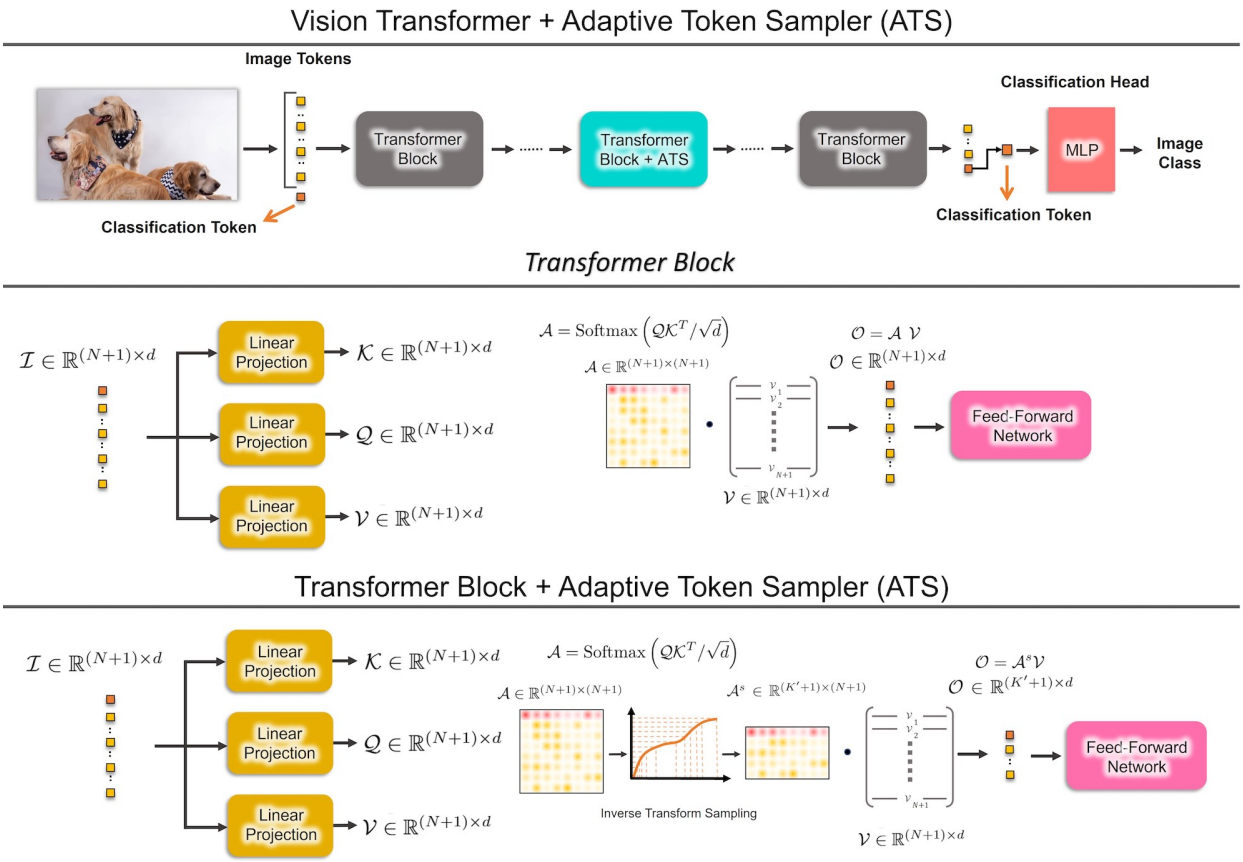


FIGURE A.3. The Adaptive Token Sampler (ATS) can be integrated into the self-attention layer of any transformer block of a vision transformer model (top). The ATS module takes at each stage a set of input tokens I. The first token is considered as the classification token in each block of the vision transformer. The attention matrix A is then calculated by the dot product of the queries Q and keys K, scaled by $\sqrt{d}$. Having selected the significant tokens, we then sample the corresponding attention weights (rows of the attention matrix A) to get A^s. Finally, we softly downsample the input tokens I to output tokens O using the dot product of A^s and V. Next, we forward the output tokens O through a Feed-Forward Network (FFN) to get the output of the transformer block.

Training To train our adaptive model, we follow most of the TimeSformer [38] network's training settings. We use the TimeSformer model's pre-trained weights to initialize the backbones of our adaptive networks and train it for 5 epochs using SGD optimizer. The learning rate and batch size are set to 5e-6 and 32, respectively. We use the cosine scheduler to train the networks.

Evaluation To evaluate our TimeSformer-HR+ATS and TimeSformer-L+ATS models, we use the same setup as [38].

A.1.5.6 Integrating ATS into a Transformer Block

Unlike a standard transformer block in vision transformers, we assign a score to each token and use inverse transform sampling to prune the rows of the attention matrix A to get A^s. Next, we get the output $O = A^s V$ and forward it to the Feed-Forward Network (FFN) of the transformer block. We visualize the details of our ATS module, which is integrated into a standard self-attention layer in Fig. A.3.

A.1.6 Ablation

A.1.6.1 Score Assignment

In Chapter 2.1, we analyzed the impact of using different tokens to calculate the significance scores S. In all of our experiments, we suggested keeping the classification token since the loss is defined on this token and discarding it may negatively affect the performance. To represent the importance of this token experimentally, we sum over the attention weights of all tokens (rows of the attention matrix) to find the most significant tokens. We show this in Fig. A.4 as Self-Attention Score (CLS Enforced). In contrast to our previous experiments, we allow ATS to remove the classification token when it is of low importance based on the significance scores S. We show the results of this experiment in Fig. A.4 as Self-Attention Score (CLS Not Enforced). As it can be seen in Fig. A.4, discarding the classification token reduces the top-1 accuracy.

TABLE A.3. Comparison of the inverse transform sampling approach with the top-K selection. We finetune and test two different versions of the multi-stage DeiT-S+ATS model: with (1) top-K token selection and (2) inverse transform token sampling. We report the top-1 accuracy of both networks on the ImageNet validation set. For the model with the top-K selection approach, we set $K_n = \lfloor 0.865 \times \#InputTokens_n \rceil$ where n is the stage index. For example, $K_3 = 171$ in stage 3.

Method	Top-1 acc	GFLOPs
Top-K	78.9	2.9
Inverse Transform Sampling	**79.7**	2.9

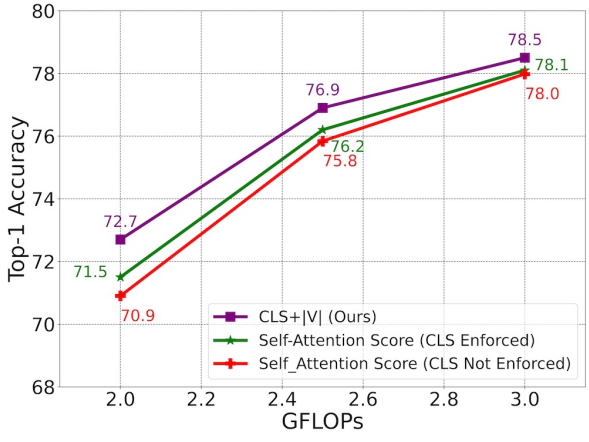


FIGURE A.4. Impact of allowing ATS to discard the classification token on the network's accuracy. The model is a single stage DeiT-S+ATS without finetuning.

A.1.6.2 Candidate Token Selection

As mentioned in Chapter 2.1, we employ the inverse transform sampling approach to softly downsample input tokens. We investigated this in Section 2.1.4. To better analyze it, we also evaluate the performance of our trained multi-stage DeiT-S+ATS model when picking the top K tokens with the highest significance scores S. To this end, we trained our DeiT-S+ATS network with the top-K selection approach and compared it to our DeiT-S+ATS model with the inverse transform sampling method. As it can be seen in Table A.3, our inverse transform sampling approach outperforms the top-K selection with and without training (Fig 2.5a). As discussed earlier, our inverse transform sampling approach does not hardly discard all tokens with lower significance scores and hence provides a more diverse set of tokens for the following layers. This sampling strategy also helps the model to gain a better performance after training, thanks to a more diversified token selection.

A.1.6.3 ATS Placement

To evaluate the effect of our ATS module's location within a vision transformer model, we add it to different stages of the DeiT-S network and evaluate it on the ImageNet validation set without finetuning the model. To have a better comparison, we set the average computation costs of all experiments to 3 GFLOPs. As it can be seen in Table A.4, integrating the ATS module into the first stage of the DeiT-S model results in a poor top-1 accuracy of 73.1%. On the other hand, integrating the ATS module into stage 3 results in a 78.5% top-1 accuracy. As mentioned before, earlier transformer blocks are more prone to predict noisier attention weights for the classification token. Therefore, integrating our ATS module into the first stage performs worse than incorporating it into the stage 3. Although the attention weights of the stage 6 are less noisy, we have to discard more tokens to reach the desired GFLOPs level of 3. For example in stages 0, 3, and 6, we set K to 130, 108, and 56, respectively. The highest accuracy is obtained when we integrate the ATS module into multiple stages of the DeiT-S model. This is because of the progressive token sampling that occurs in a multi-stage DeiT-S+ATS model. In other words, a multi-stage DeiT-S+ATS network can gradually decrease the GFLOPs by discarding fewer tokens in the earlier stages, while a single-stage DeiT-S+ATS model has to discard more tokens in the earlier stages to reach the same GFLOPs level. We also added the ATS module into all stages, yielding average GFLOPs of 2.6 and 76.9% top-1 accuracy.

TABLE A.4. Evaluating the integration of the ATS module into different stages of DeiT-S [431].

Stage(s)	0	3	6	3-11
Top-1 Accuracy	73.1	78.5	77.4	**79.2**

A.1.6.4 Adding ATS to Models with Other Token Pruning Approaches

To better evaluate the performance of our adaptive token sampling approach, we also add our module to the state-of-the-art efficient vision transformer EViT-DeiT-S [274]. EViT [274] introduces a token reorganization method that first identifies the top-K important tokens by computing token attentiveness between the tokens and the classification token and then fuses less informative tokens. Interestingly, our ATS module can also be added to the EViT-DeiT-S model and further decrease the GFLOPs, as shown in Table A.5. These results demonstrate the superiority of our adaptive token sampling approach compared to static token pruning methods. We integrate our ATS module into stages 4, 5, 7, 8, 10, and 11 of the EViT-DeiT-S backbone and fine-tune them for 10 epochs following our fine-tuning setups on the ImageNet dataset discussed earlier.

TABLE A.5. Evaluating the EViT-DeiT-S [274] model's performance when integrating the ATS module into it with $K_n = \lfloor 0.7 \times \#InputTokens_n \rceil$ where n is the stage index.

Model	Top-1 acc	GFLOPs
EViT-DeiT-S (30 Epochs) [274]	79.5	3.0
EViT-DeiT-S (30 Epochs)+ATS	79.5	**2.5**
EViT-DeiT-S (100 Epochs) [274]	79.8	3.0
EViT-DeiT-S (100 Epochs)+ATS	79.8	**2.5**

A.2 SimA: Simple Softmax-free Attention for Vision Transformers

A.2.1 Inference Time Comparison on GPU:

We compare execution time of SimA and other SOTA methods on edge devices in Figure 2.9. Additionally, we compare execution time of SimA, XCiT, and DeiT on GPU in Figure A.5.

Visualization: Figure A.6 provides more results similar to Figure 2.11. Please see Section 2.2.4.7 for details.

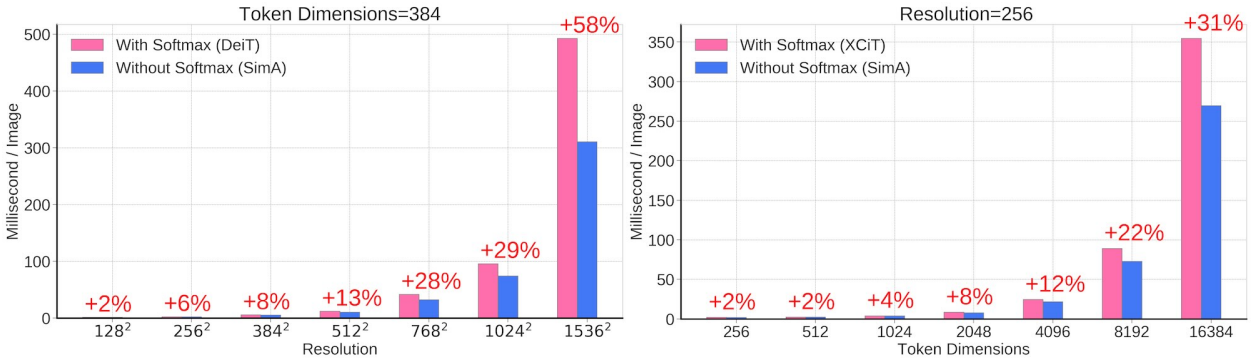


FIGURE A.5. **Effect of Softmax on inference time (GPU):** We evaluate performance of each model on a single RTX 8000 GPU with batch size of 8. When comparing the baseline to our method (SimA), we fix the order of (QK^TV) to have the same dot product complexity as the baseline. For example, when comparing with DeiT, if $N > D$, then it is more efficient to do $\hat{Q}(\hat{K}^TV)$ for our method, but we do $(\hat{Q}\hat{K}^T)V$ to have same complexity as DeiT($O(N^2D)$). We do this to solely evaluate the effect of Softmax on the computation time. **Left:** We fix the token dimension to 384 and increase the image resolution. At 1536×1536 resolution, DeiT is 58% slower than our method due to the overhead of $exp(.)$ function in Softmax. **Right:** We fix the resolution and increase the capacity of the model (dimensions of Q and K). With 8192 dimensions, XCiT is 22% slower due to Softmax overhead.

A.2.2 SimA without LPI:

Although XCiT [23] shows that LPI layer can improve the accuracy by 1.2 point, it limits the application of vanilla transformer (e.g., running masked auto encoder models like MAE [174] is not straightforward). To show that our method is not dependent on LPI, we train our model without LPI. We observe that the accuracy drops by 1.2 point (82.1% vs 80.9%). Hence, although LPI boosts the accuracy, our method has comparable performance without LPI.

A.2.3 Details of Linear Attention Comparison:

CosFormer with cosine re-weighting requires 4× more FLOPs compared to SimA in multiplying K and V matrices. Since CosFormer is developed for NLP, it assumes one dimensional indexing for the tokens.

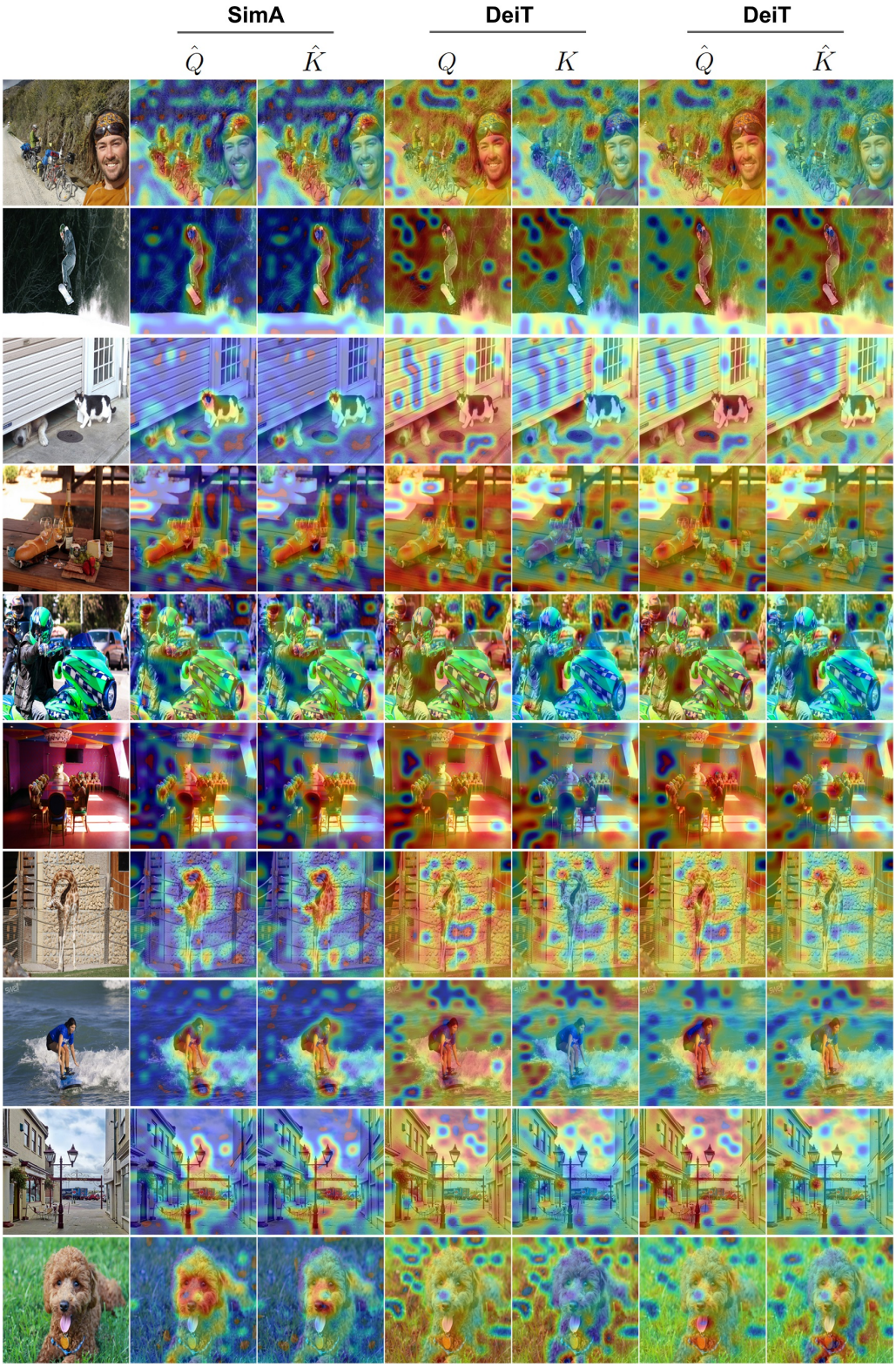


FIGURE A.6. **Our method (SimA):** We extract $\hat{Q}$ and $\hat{K}$ from layer 12 of transformer. We get ℓ_2-norm of each token for $\hat{Q}$ and $\hat{K}$, normalize it to range [0,1] and overlay it as a heatmap on the image. Interestingly, magnitude of tokens represent the significance of tokens in our method. Note that all images are randomly selected from MS-COCO test set without any visual inspection or cherry picking.

However, applying it to vision, we need to index the tokens with two indices to take advantage of the induced locality. To do so, one may introduce two cosine weights to Eq 10 of CosFormer [51]: one in x direction and the other one in y direction to come up with:

$$Q_{i,m}K^T_{j,n}cos(i-j)cos(m-n)$$

which can be expanded to:

$$Q_{i,m}K^T_{j,n}\Big(cos(i)cos(j)+sin(i)sin(j)\Big)\Big(cos(m)cos(n)+sin(m)sin(n)\Big)$$

which can be regrouped to:

$$\begin{aligned}
&=\Big(Q_{i,m}cos(i)cos(m)\Big)\Big(K^T_{j,n}cos(j)cos(n)\Big)\\
&+\Big(Q_{i,m}cos(i)sin(m)\Big)\Big(K^T_{j,n}cos(j)sin(n)\Big)\\
&+\Big(Q_{i,m}sin(i)cos(m)\Big)\Big(K^T_{j,n}sin(j)cos(n)\Big)\\
&+\Big(Q_{i,m}sin(i)sin(m)\Big)\Big(K^T_{j,n}sin(j)sin(n)\Big)
\end{aligned}$$

Hence, for every attention value, CosFormer needs 4 dot products between Q and K vectors while our method needs only one dot product. Hence, following Eq. 12 of the CosFormer paper, CosFormer needs 4 times more FLOPS compared to our method in calculating the attention values (multiplying Q, K, and V matrices).

A.3 CompRess: Self-Supervised Learning by Compressing Representations

A.3.1 More results for cluster alignment

For cluster alignment experiment (Section 2.3.4.3), we calculate the alignment for each category, sort them, and show in Figure A.7. Moreover, Figure A.8 is a larger version that is generated similar to Figure 2.14(c). Each row is a random cluster while images in the row are randomly sampled from that cluster with no manual selection or cherry-picking.

TABLE A.6. **Parameter and FLOPs comparison:** We report the number of floating point operations (FLOPs) and the number of parameters in a model below.

Model	FLOPs (G)	Params (M)
MobileNet-V2	0.33	3.50
AlexNet	0.77	61.0
ResNet-18	1.82	11.69
ResNet-50	4.14	25.56
ResNet-50x4	64.06	375.38

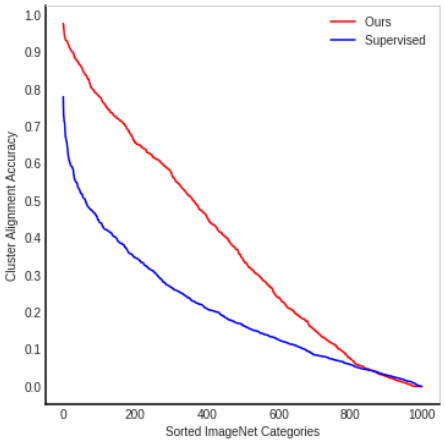


FIGURE A.7. **Cluster alignment accuracy:** We calculate the accuracy for each ImageNet category and then plot them after sorting.

A.3.2 Implementation details for the baselines

Non-compressed (MoCo): We use MoCo-v2 [76] from the official code [12] with AlexNet, ResNet-18, and MobileNet-V2 architectures for 200 epochs. We also train a longer baseline with ResNet-18 for 1000 epochs. All other hyperparameters are the same as the official version (m=0.999, lr=0.03).

CRD: We use the official code provided by the authors [3] and removed the supervised loss term. We use their default ImageNet hyperparameter of $lr = 0.05$ except for AlexNet student for which we use $lr = 0.005$ to make it converge.

CC: We calculate the ℓ_2 normalized embeddings for the entire training dataset and apply k-means clustering with $(k = 16,000)$ (which is adopted from [495]). This is equivalent to clustering with cosine similarity. We got slightly better results for cosine similarity compared to Euclidean distance. We use the FAISS GPU based k-means clustering implementation [9]. Finally, the student is trained to classify the cluster

assignments. As in [325, 495], we train the student for 100 epochs. We use lr = 0.1 for ResNet models and lr = 0.01 for MobileNet-V2 and AlexNet models. We use cosine learning rate annealing.

Reg: We use Adam optimizer with weight decay of $1e-4$ for 100 epochs, and batch size of 256. For MobileNet-V2 and ResNet-18, we use lr = 0.001, and for AlexNet lr = 0.0001. The lr is reduced by a factor of 10 at the 40-th and 80-th epochs. We use ADAM optimizer as performed better than SGD.

Reg-BN: It is similar to Reg except that we use SGD optimizer with lr = 0.1 instead of ADAM.

A.3.3 Details of Places experiments (Section 2.3.4.4)

We perform adaptive max pooling to get features with dimensions around 9K, and train a linear layer on top of them. Training is done for 90 epochs with lr = 0.01, batch size = 256, weight decay = $1e-4$, momentum = 0.9, and lr multiplied by 0.1 at 30, 60, and 80 epochs.

A.3.4 Details of PASCAL experiments (Section 2.3.4.4)

For classification, we train a single linear layer on top of a frozen backbone. We use SGD with learning rate = 0.01, batch size = 16, weight decay = $1e-6$, and momentum = 0.9. We train for $80,000$ iterations and multiply learning rate by 0.5 every $5,000$ iterations.

For object detection, we use SGD learning rate = 0.001, weight decay = $5e-4$, momentum = 0.9 and batch size = 256. We train for $15,000$ iterations and multiply learning rate by 0.1 every $5,000$ iterations. The training parameters are adopted form [325] and the code from [150].

A.3.5 Details of small data ImageNet experiments (Section 2.3.4.4)

We train a single linear layer on top a frozen backbone. We use SGD with learning rate = 0.05, batch size = 256, weight decay = $1e-4$, and momentum = 0.9. We use cosine learning rate decay and train for 30 and 60 epochs for 10 percent and 1 percent subsets respectively. The subsets and training parameters are adopted from [70].

FIGURE A.8. **Cluster Alignment:** Similar to Figure 2.14(c), we show 20 randomly selected images (columns) from 30 randomly selected clusters (rows) for our best AlexNet modal. This is done with no manual inspection or cherry-picking. Note that most rows are aligned with semantic categories.

A.4 NOLA: Compressing LoRA using Linear Combination of Random Basis

A.4.1 Measuring the rank of possible solutions in NOLA vs PRANC:

Choosing n basis vectors (d dimensional each) in PRANC will result in all possible learned matrices living in a n dimensional subspace. However, since NOLA with the same number of total parameters ($k+l=n$) uses $A \times B$ factorization, the possible solutions can live in a higher dimensional subspace. We do a simple experiment by sampling several random coefficient vectors, reconstructing the ΔW matrix, reshaping it to be a long (d^2)-dimensional vector, and measuring the rank of the covariance of samples to see how much of the whole space is covered by the samples. The results are shown in Figure A.9 for a simple experiment with varying d and n. As expected, NOLA can cover the whole space (full-rank) using a small number of parameters compared to PRANC. Note that the rank in this analysis is on the covariance of possible random samples of weight matrices and should not be confused with the rank in LoRA or NOLA formulation.

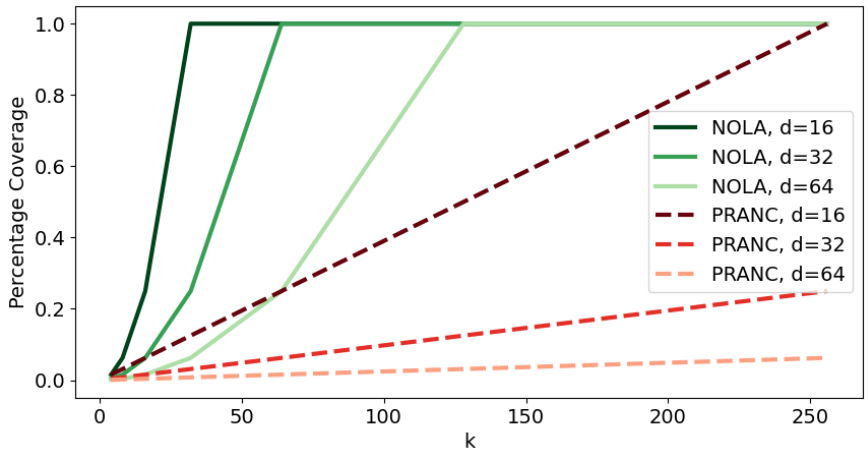


FIGURE A.9. Comparing the rank of samples in the solution subspace for PRANC and NOLA, given the same number of parameters, n. "Percentage Coverage" is the subspace rank divided by the max possible rank (d^2), so 1.0 denotes full rank. As expected, the coverage for PRANC increases linearly while it saturates very fast for NOLA.

A.4.2 NOLA in training from scratch:

MLP on MNIST (a toy experiment):

We believe NOLA is a better reparametrization than PRANC and can achieve local minimums that PRANC cannot achieve. To show this empirically, we perform a very simple experiment where we apply

a 2-layer MLP for the MNIST classification task. Since we want to measure which method is better at reaching the local minimums and not necessarily the generalization, we evaluate the models by the training loss. Also, we intentionally use a large number of neurons in the hidden layer to over-parameterize and increase the number of local minimums.

We use a linear layer from 784 features to 256 with bias followed by ReLU and a linear layer from 256 to 10 classes. We use all $60K$ samples for training to report the training loss. For both PRANC and NOLA, we use 32 parameters for each layer. Other hyperparameters are same for both: 200 epochs, 512 batch size with $lr = 0.05$. We use rank $r = 4$ for NOLA. PRANC has a final training loss of 0.87, while NOLA achieves a training loss of 0.71. This simple experiment empirically supports that NOLA has better representation power. Additionally, PRANC training finishes in 1152 seconds while NOLA finishes in 579 seconds. We will leave the theoretical study of this comparison for future work. Moreover, this experiment empirically shows that NOLA is a generic method, and its success is not dependent on the architecture of transformers or attention modules.

CNN on ImageNet100 and CIFAR-10:

Moreover, to compare the representation power of NOLA and PRANC, we train NOLA from scratch on an image classification task using a CNN architecture. For each convolution layer, we reshape all parameters of a layer into a matrix (close to square shape) and apply NOLA to the matrix. Then, we reshape it to the original shape of the convolution. Additionally, we train LoRA using a similar approach as NOLA. We follow a similar setup as [322] for our experiments on image classification.

Datasets and Architectures: We consider two architectures in our experiments: ResNet20 with $270K$ parameters, and ResNet18 [179] with $11M$ parameters. We train ResNet20 on CIFAR10 [248], and ResNet18 on ImageNet100 [377].

Results: We report result of ImageNet100 in Table A.8, and CIFAR10 in Table A.7. NOLA outperforms both PRANC and LoRA with a similar number of parameters.

Implementation Details: For ImageNet100 and ResNet18, we use $k = l = 2,000$ basis for each of 20 modules, and for the classifier (last linear layer), we used $k = l = 10,000$, resulting in a total of $100,000$ trainable parameters excluding $9,600$ batchnorm parameters. We use rank 64 for all layers. We train all models using Adam optimizer with a learning rate of 0.001 and batch size of 256 for 200 epochs. For CIFAR-10 and ResNet20, we use $k = l = 250$ basis for each convolutional module, and for the linear layer, we use $k = l = 1000$ parameters. We use batch size 256, Adam optimizer, and a learning rate of 0.001. We use a single NVIDIA-GeForce RTX 3090 for all experiments.

Training Time Comparison: We measure the training time of NOLA and PRANC on a single NVIDIA-GeForce RTX 3090 GPU and batch size of 256. Note that training time includes both forward and backward passes for each batch. On average, NOLA processes a batch in 228ms while PRANC does the same in 1070ms, so NOLA is 4.6 times faster than PRANC.

TABLE A.7. **Training On CIFAR10:** Result of our method on CIFAR10 dataset and ResNet20.

Method	# Params	Acc.
trained model	269,722	**88.92%**
PRANC	12,752	81.5%
LoRA	13,295	81.5%
NOLA	12,876	82.4%

TABLE A.8. **Training On ImageNet100:** Result of our method on ImageNet-100 dataset and ResNet18

Method	# Params	Acc.
trained model	11,227,812	**82.1%**
HashedNet [74]	129,200	52.96%
PRANC	119,200	61.08%
LoRA	150,000	63.50%
NOLA	109,600	64.66%

A.4.3 Ablation and details of NOLA on Vision Transformers:

Implementation detail: We consider learning rates of $5e-3$, $1e-3$ and $5e-4$ for LoRA, NOLA and Linear methods and $8e-5$, $5e-5$, $3e-5$ and $1e-5$ for Full-FT. The best settings is chosen based on the performance on validation set. For creation of k-shot dataset, we randomly sample without replacement from the train set. For each of these sets, we run with three different initializations of the networks. This process is repeated four times and the averaged values are reported.

Comparison between NOLA-QV and NOLA-MLP: We experiment with NOLA layer in both the attention and MLP modules of the vision transformer. We observe that applying NOLA on MLP performs better than that on attention block (Table A.9). Thus, we use NOLA-MLP as our default setting. Note that the number of trainable parameters remains the same in both versions. Unlike this, applying LoRA on MLP block would require significantly higher number of trainable parameters due to the increased dimensions of the weight matrices in MLP compared to those in attention block.

TABLE A.9. **Comparison between NOLA in MLP and attention blocks:** We observe that NOLA on MLP block is more effective. We choose this as our default setting.

Base Model		# Train Params	CIFAR-10		CIFAR-100		CUB-200-2011		Caltech-101	
			5	10	5	10	5	10	5	10
ViT-L	NOLA-QV	47K	87.0 (0.9)	91.6 (0.7)	74.8 (0.6)	80.4 (0.9)	75.3 (0.4)	81.7 (0.3)	87.9 (1.1)	90.6 (0.5)
	NOLA-MLP	47K	87.9 (1.3)	92.2 (0.5)	75.1 (0.6)	81.3 (0.8)	75.5 (0.6)	81.7 (0.4)	88.0 (1.2)	90.6 (0.5)

A.4.4 Results of NLG task on DART and WebNLG datasets:

In Table A.10, we report more results similar to Table 3.1 using GPT-2 M and GPT-2 L on DART [316] and WebNLG [147] datasets.

TABLE A.10. **DART and WebNLG Dataset**: Similar to Table 3.1 we compare NOLA to other methods. NOLA is on par or better with other methods with the same number of parameters.

Method	Adapted Layers	Adapter Rank	# Trainable Parameters	DART			WebNLG		
				BLEU↑	MET↑	TER↓	BLEU↑	MET↑	TER↓
GPT-2 M									
Finetune	All Layers	-	354.000M	46.2	0.39	0.46	46.5	0.38	0.53
$\text{Adapter}^{\text{L}}$	Extra Layers	-	0.370M	42.4	0.36	0.48	50.2	0.38	0.43
$\text{Adapter}^{\text{L}}$	Extra Layers	-	11.000M	45.2	0.38	0.46	54.9	0.41	0.39
$\text{Finetune}^{\text{Top2}}$	Last 2 Layers	-	24.000M	41.0	0.34	0.56	36.0	0.31	0.72
PreLayer	Extra Tokens	-	0.350M	46.4	0.38	0.46	55.1	0.41	0.40
LoRA	QV	4	0.350M	47.1	0.39	0.46	54.9	0.41	0.39
LoRA	QV	1	0.098M	46.4	0.38	0.48	53.5	0.40	0.40
NOLA (Ours)	QV	8	0.096M	47.0	0.38	0.48	53.9	0.40	0.40
NOLA (Ours)	MLP	8	0.096M	47.1	0.38	0.47	54.7	0.41	0.40
NOLA (Ours)	QV	8	0.048M	45.7	0.38	0.49	53.8	0.40	0.40
NOLA (Ours)	MLP	8	0.048M	45.5	0.38	0.49	53.0	0.40	0.40
GPT-2 L									
Finetune	All Layers	-	774.000M	47.0	0.39	0.46	55.5	0.42	0.42
$\text{Adapter}^{\text{L}}$	Extra Layers	-	0.880M	45.7	0.38	0.46	56.0	0.41	0.39
$\text{Adapter}^{\text{L}}$	Extra Layers	-	230.000M	47.1	0.39	0.45	57.7	0.43	0.39
PreLayer	Extra Tokens	-	0.770M	46.7	0.38	0.45	56.3	0.42	0.40
LoRA	QV	4	0.770M	47.5	0.39	0.45	57.1	0.43	0.38
LoRA	QV	1	0.184M	47.7	0.39	0.47	55.9	0.42	0.39
NOLA (Ours)	QV	8	0.144M	47.8	0.39	0.47	55.8	0.41	0.39
NOLA (Ours)	MLP	8	0.144M	47.8	0.39	0.47	56.0	0.42	0.39
NOLA (Ours)	QV	8	0.072M	46.4	0.38	0.48	55.5	0.41	0.38
NOLA (Ours)	MLP	8	0.072M	46.8	0.38	0.48	55.8	0.41	0.39

A.5 ISD: Self-Supervised Learning by Iterative Similarity Distillation

A.5.1 Transfer evaluation training details:

we freeze the backbone and forward train set images without augmentation (resize shorter side to 256, take a center crop of size 224, and normalize with ImageNet statistics). Then we train a linear layer on top of extracted features. We split each dataset to train, validation, and test set. We search for best lr in 10 log spaced values between -3 and 0 and weight decay in 9 log spaced values between -10 and -2, then we train linear layer with best parameters on train+validation set and evaluate it on test set.

TABLE A.11. The train, val, and test split sizes for transfer datasets are listed above. **Test split:** For DTD and Flowers datasets, we use the provided test sets. Otherwise, in case of Sun397, Cars, CIFAR-10, CIFAR-100, Food101, and Pets datasets, the val set provided in the dataset is used as the hold-out test set. Also, for Caltech-101, the hold-out test set is created by randomly sampling 30 images/class from the train set. **Val split:** For DTD and Flowers datasets, we use the provided val sets. Otherwise, the val set is created by a randomly sampled subset of the train set is used as the val set. We report the strategy for splitting val sets for different datasets: 5 samples/class for Caltech-101, 20% samples/class for Cars, 50 samples/class for CIFAR-100, 50 samples/class for CIFAR-10. 75 samples/class for Food101, 20 samples/class for Pets, 10 samples/class for Sun397. We attempt to be as close to the details provided in BYOL [159] as possible.

Dataset	Classes	Train samples	Val samples	Test samples	Accuracy measure	Test provided
Food101 [41]	101	68175	7575	25250	Top-1 accuracy	-
CIFAR-10 [245]	10	49500	500	10000	Top-1 accuracy	-
CIFAR-100 [245]	100	45000	5000	10000	Top-1 accuracy	-
Sun397 (split 1) [481]	397	15880	3970	19850	Top-1 accuracy	-
Cars [244]	196	6509	1635	8041	Top-1 accuracy	-
DTD (split 1) [90]	47	1880	1880	1880	Top-1 accuracy	Yes
Pets [333]	37	2940	740	3669	Mean per-class accuracy	-
Caltech-101 [130]	101	2550	510	6084	Mean per-class accuracy	-
Flowers [321]	102	1020	1020	6149	Mean per-class accuracy	Yes

TABLE A.12. We explore various hyper-parameters for ResNet-18 model. "s/s" refers to the setting where both views use strong augmentation while in "w/s" the teacher view uses weak augmentation while the student view uses strong augmentation. The projection layer is a 2-layer MLP with 1024 as hidden dim and 128 as the output dim. There is a BatchNorm layer followed by ReLU between the two layers. In step learning rate decay, the LR is reduced by a factor of 0.2 at 140 and 180 epochs. The training happens for 200 epochs. The 1st row uses the same setting as the ResNet-18 model in the main paper. The 6th row uses the same settings as the ResNet-50 model in the main paper.

	LR	m	Aug.	Proj.	τ_t	τ_s	NN	20-NN
1	step	0.999	s/s	✗	0.02	0.02	41.5	46.6
2	step	0.999	w/s	✗	0.02	0.02	41.7	46.7
3	step	0.99	s/s	✗	0.02	0.02	40.2	45.2
4	cosine	0.999	s/s	✗	0.02	0.02	40.9	45.7
5	cosine	0.99	w/s	✓	0.02	0.02	34.6	38.3
6	cosine	0.99	w/s	✓	0.02	0.2	39.4	44.4
7	cosine	0.99	w/s	✓	0.01	0.1	31.7	37.3
8	step	0.999	w/s	✗	0.02	0.2	6.0	8.0
9	step	0.999	w/s	✗	0.02	0.5	5.5	7.0
10	step	0.999	w/s	✗	0.03	0.3	2.9	3.9

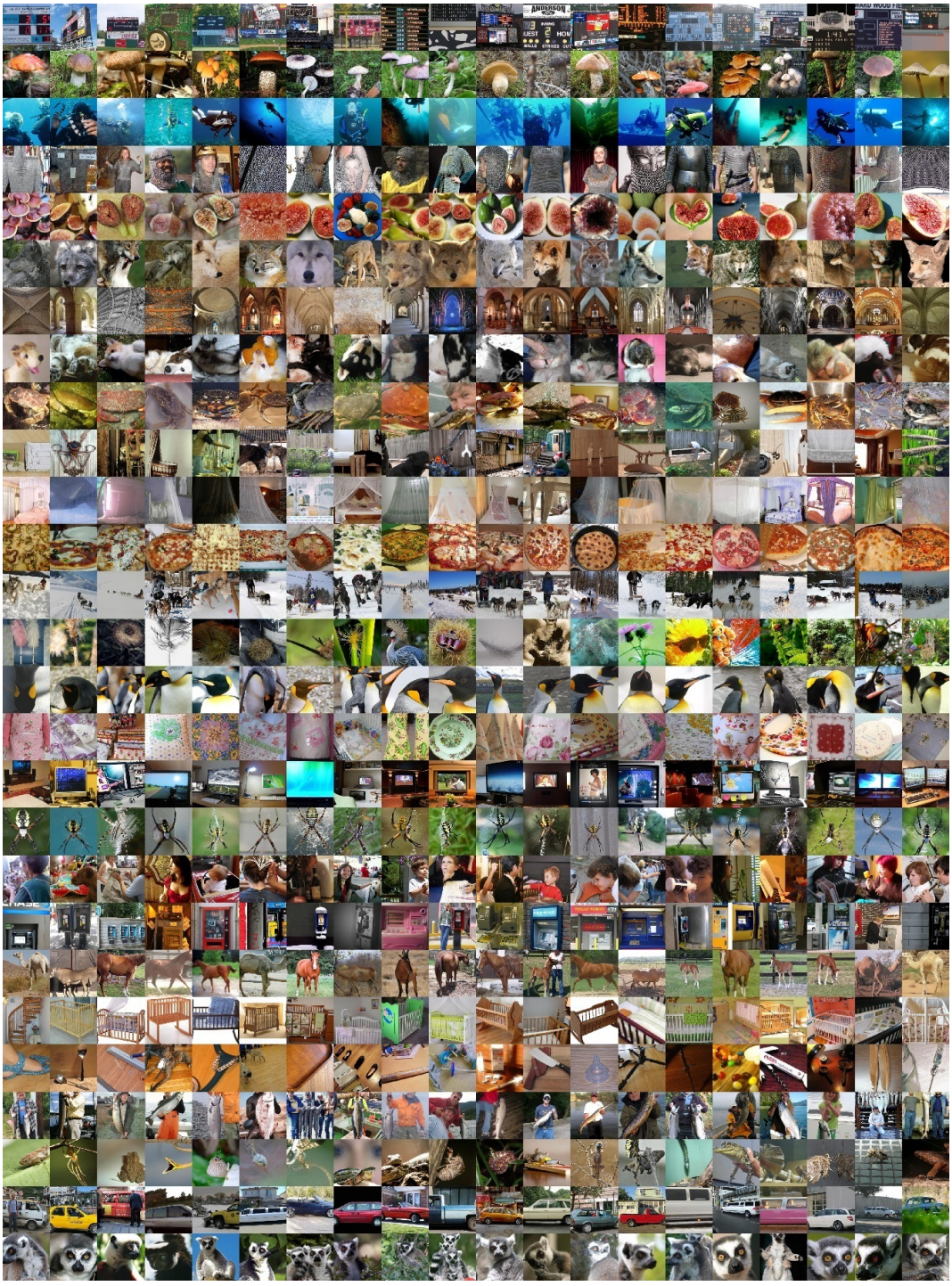


FIGURE A.10. **Random Clusters:** We cluster ImageNet dataset into 1000 clusters using k-means and show random samples from random clusters. We have no cherry-picking for this visualization. Interestingly, images from each row(each cluster) are semantically similar.

A.6 Mean Shift for Self-Supervised Learning

TABLE A.13. We list the sizes of train, val, and test splits of the transfer datasets. **Test split:** We use the provided test sets for Aircraft, DTD, and Flowers datasets. In case of Sun397, Cars, CIFAR-10, CIFAR-100, Food101, and Pets datasets, we use the provided val set as the hold-out test set. In case of Caltech-101, we use a random split of 30 images per category as the hold-out test set. **Val split:** We use the provided val sets for the datasets DTD and Flowers. For all other datasets, the val set is created by randomly sampling a subset of the train set. In order to be as close to BYOL [159] transfer setup as possible, we use the following val set splitting strategies for each dataset. Aircraft: 20% samples/class. Caltech-101: 5 samples/class. Cars: 20% samples/class. CIFAR-100: 50 samples/class. CIFAR-10: 50 samples/class. Food101: 75 samples/class. Pets: 20 samples/class. Sun397: 10 samples/class.

Dataset	Classes	Train samples	Val samples	Test samples	Accuracy measure	Test provided
Food101 [41]	101	68175	7575	25250	Top-1 accuracy	-
CIFAR-10 [245]	10	49500	500	10000	Top-1 accuracy	-
CIFAR-100 [245]	100	45000	5000	10000	Top-1 accuracy	-
Sun397 (split 1) [481]	397	15880	3970	19850	Top-1 accuracy	-
Cars [244]	196	6509	1635	8041	Top-1 accuracy	-
DTD (split 1) [90]	47	1880	1880	1880	Top-1 accuracy	Yes
Pets [333]	37	2940	740	3669	Mean per-class accuracy	-
Caltech-101 [130]	101	2550	510	6084	Mean per-class accuracy	-
Flowers [321]	102	1020	1020	6149	Mean per-class accuracy	Yes

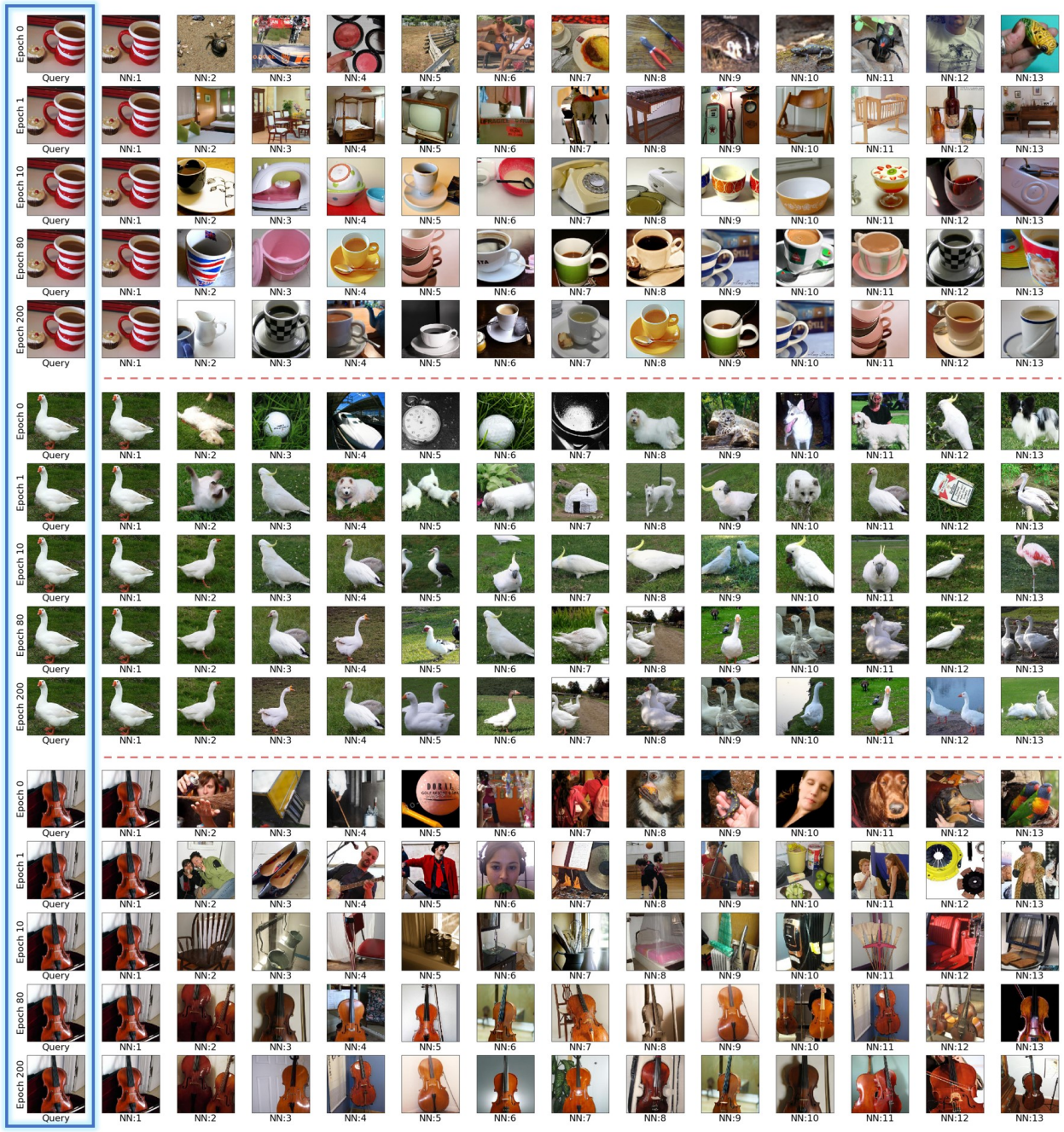


FIGURE A.11. **Nearest neighbors (NN) of the model at each epoch:** Similar to Figure 4.7.

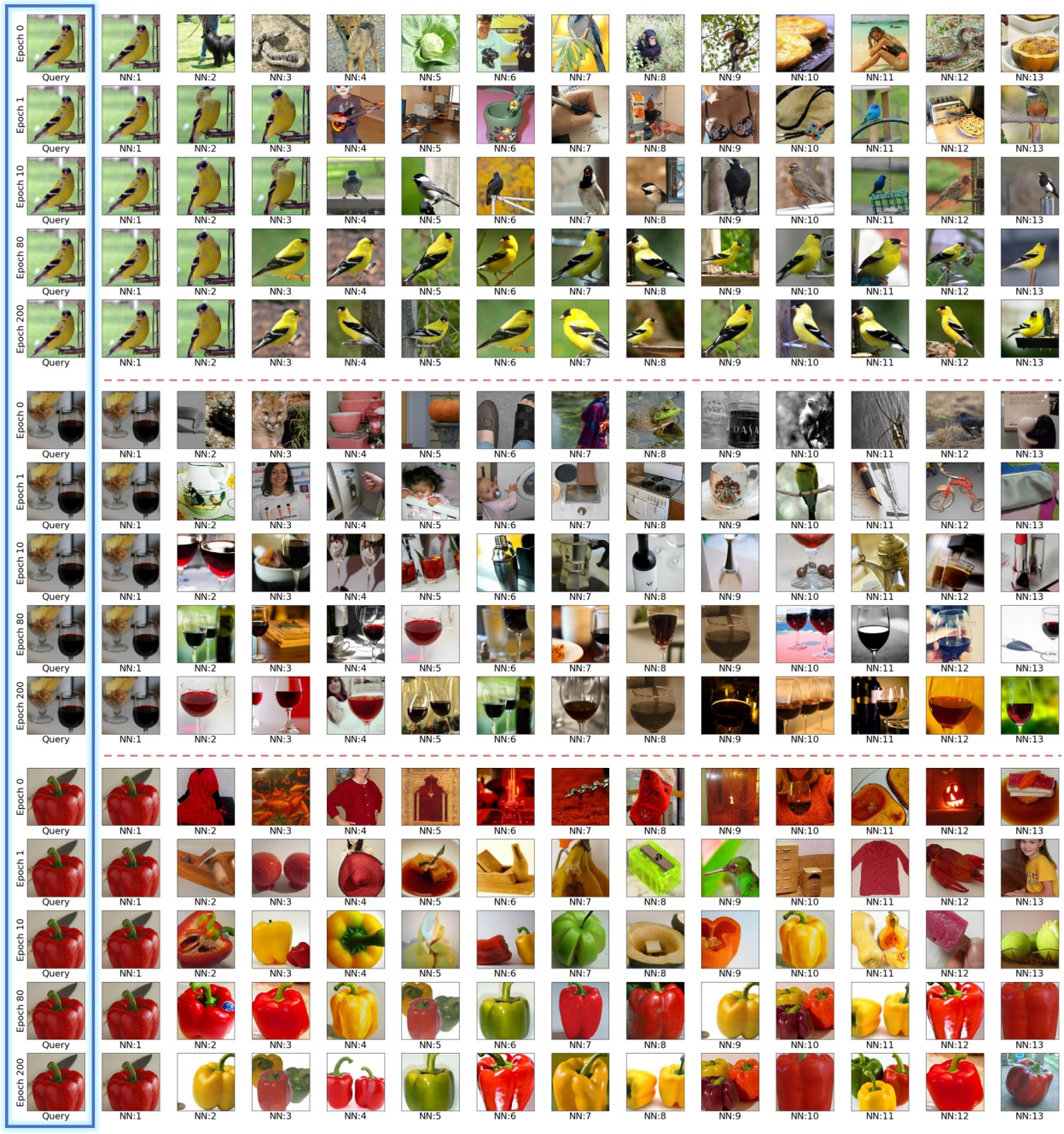


FIGURE A.12. **Nearest neighbors (NN) of the model at each epoch:** Similar to Figure 4.7.

FIGURE A.13. **Random Clusters:** We forward ImageNet training set through our ResNet-50 model and cluster them into 1000 clusters using k-means. We select 30 clusters randomly and show 20 randomly sampled images from each cluster without cherry-picking. Each row corresponds to a cluster. Note that semantically similar images are clustered together.

A.7 Constrained Mean Shift for Representation Learning

Here, we provide additional results and analysis on self-supervised (section A.7.1), cross modal constraint (section A.7.4), semi-supervised (section A.7.5) and supervised (section A.7.9) settings. Additional results include analysis of retrieved far neighbors (Figs. A.14 and A.15) and ablations to justify various design choices (tables A.18, A.19, A.20, A.21, A.23, A.25, A.27). More details on implementation (section A.7.10) and compute calculation (Eq. A.1, table A.24) are provided.

A.7.1 Results on Self-supervised Setting

A.7.1.1 Using far neighbors in unconstrained MSF:

In $\text{CMSF}_{\text{self}}$, we use augmented images from previous epoch to obtain distant neighbors. A trivial way to sample farther neighbors is to just increase the number of neighbors k in the original (unconstrained) MSF [242] method. Here we train MSF with $k = 500$ to compare with our $\text{CMSF}_{\text{self}}$. We train all models for 80 epochs. Settings are similar to our self-supervised settings in section 4.3.4.1. For fair comparison, we use memory-bank of size $256k$ for MSF while we use $128K$ for $\text{CMSF}_{\text{self}}$. Results are in Table A.14. While increasing k in MSF helps to sample far NNs, it degrades the accuracy. We hypothesize that this happens due to reducing purity of top-k in unconstrained MSF with increasing k (also shown in Fig. 4.14). This experiment shows that it is not trivial to to sample far NNs with good purity.

TABLE A.14. **Using far neighbors in unconstrained MSF [242]:** Using far NNs by trivially increasing k in MSF baseline degrades the accuracy. This is due to the low purity of far NNs in MSF when no constraint is utilized. However, $\text{CMSF}_{\text{self}}$ achieves high accuracy while using distant NNs.

	MSF [242] Top-k = 500	MSF [242] Top-k = 5	$\text{CMSF}_{\text{self}}$ Top-k = k' = 5
NN	35.8	49.7	51.4
20-NN	40.2	54.0	55.5

A.7.2 Effect of number of neighbors k

$\text{CMSF}_{\text{self}}$ uses top-k NNs as part of loss calculation. Here we study the effect of k in $\text{CMSF}_{\text{self}}$ performance. We set $k' = k$ in all models. Note that k' is the number of NNs retrieved from the second memory bank M'. We train all models for 80 epochs. All settings are similar to that of $\text{CMSF}_{\text{self}}$ in Section 3.1. Results are shown in Table A.15. Higher values of k and k' degrade model performance. We thus use $k' = k = 5$ in all our main experiments.

TABLE A.15. **Effect of k in top-k NNs sampling within M:** We set $k = k'$ in all models and varied k. We use memory bank of size $256k$. Increasing k degrades the accuracy of the model.

	k'=k=5	k'=k=10	k'=k=20	k'=k=50
NN	51.4	51.3	51.1	49.5
20-NN	55.5	55.3	55.3	53.8

A.7.3 Results with Different Architectures

In addition to the ResNet-50 architecture, we experiment with a smaller and a larger backbone architecture. We consider ResNet-18 and ResNet-101 networks. The results are shown in table A.16. The proposed $\text{CMSF}_{\text{self}}$ improves over MSF across different architectures. Note that the networks are trained only for 100 epochs on ResNet-101.

TABLE A.16. **Results with different backbone architectures:** We compare performance of our method with that of CMSF with ResNet architectures of different sizes. We observe that CMSF consistently outperforms MSF. * models were trained for 100 epochs instead of 200.

Method	ResNet-18	ResNet-50	ResNet-101*
MSF [242]	49.8	72.2	71.1
$\text{CMSF}_{\text{self}}$	51.7	73.0	71.9

A.7.4 Cross-modal Constraint

Fig. 6 of main submission showed that proposed CMSF_{sup} is more robust to noisy labels. Here, we explore another such noisy constraint: a pre-trained SSL model from another modality. We consider an unlabeled video dataset and use the RGB and optical flow inputs as the two different modalities. We first train two SSL models on the RGB and Flow modalities separately using InfoNCE method [169, 445]. Then we continue the training on one modality while freezing the other modality and using it as a constraint. In training the flow network using RGB network as constraint, we sample k' nearest neighbors in RGB's memory bank and then search for top-k nearest neighbors among those samples in the memory bank corresponding to Flow.

Implementation Details. Following [169], we use split-1 of UCF-101 [404] (13k videos) as the unlabeled dataset. We use similar augmentation and pre-processing as [169] and calculate optical-flow using unsupervised TV-L1 [514] algorithm. For cross-modal experiments, we use S3D [485] architecture with the input size of 128×128 pixels. We initialize from the pretrained weights of InfoNCE (400-epoch) released by [169].

We use following settings for our method: memory bank of size 8192, n = 10, k = 5, batch size 128, weight decay $1e-5$, initial lr of 0.001, and learning rate decay by factor of 10 at epoch 80. We train each modality for additional 100 epochs using PyTorch Adam optimizer. For a fair comparison, we run CoCLR using their official code by initializing it from the same model as ours. We use the code from [169] for linear evaluation.

Results: The results are shown in Table A.17. We report top-1 accuracy for linear classification and recall@1 for retrieval on the extracted features of frozen networks. All experiments use spatio-temporal 3D data either in RGB or flow format. At the end of 3 stages of training on Flow modality, our method outperforms CoCLR [169] baseline and MSF with 2 stages.

TABLE A.17. **Cross-modal constraint:** We initialize all models using an InfoNCE pre-trained model. In CMSF-cross modal, one of the modalities is used to constrain and train the other. Superscript indicate the constraint modality, subscript indicate the training modality. For example, in $\text{CMSF}^{\text{Flow}}_{\text{RGB}}$, we continue training CMSF on RGB modality while using frozen pretrained Flow model as the constraint. Note that CoCLR [169] also uses another modality as a constraint in the form of contrastive learning. We continue training InfoNCE SSL model for 200 epochs using MSF [242] for a fair comparison. We use S3D [485] architecture for all models. Models with the final round of training on Flow modality are highlighted with yellow and those on RGB are highlighted with blue. All rows with * contain results for the same model. Results are repeated for easier understanding of the table.

Model	Final modality	Epochs	R@1	Linear
InforNCE_{RGB}	RGB	400	35.5	47.9
$\text{InforNCE}_{RGB} \rightarrow \text{MSF}_{RGB}$	RGB	400+200	39.6	50.8
InforNCE_{Flow}*	Flow	400	45.3	66.1
$\text{InforNCE}_{Flow} \rightarrow \text{MSF}_{Flow}$	Flow	400+200	47.3	64.7
InforNCE_{Flow}*	Flow	400	45.3	66.1
$\text{InforNCE}_{Flow} \rightarrow \text{CoCLR}^{Flow}_{RGB}$	RGB	400+100	49.8	61.0
$\text{InforNCE}_{Flow} \rightarrow \text{CoCLR}^{Flow}_{RGB} \rightarrow \text{CoCLR}^{RGB}_{Flow}$	Flow	400+100+100	50.0	67.3
InforNCE_{Flow}*	Flow	400	45.3	66.1
$\text{InforNCE}_{Flow} \rightarrow \text{CMSF}^{Flow}_{RGB}$	RGB	400+100	45.8	58.1
$\text{InforNCE}_{Flow} \rightarrow \text{CMSF}^{Flow}_{RGB} \rightarrow \text{CMSF}^{RGB}_{Flow}$	Flow	400+100+100	54.1	71.2

A.7.5 Results on Semi-supervised Setting

Unless specified, we use the ImageNet100 dataset for all the ablations on the semi-supervised setting for faster experimentation.

A.7.5.1 Role of Confidence Threshold in Pseudo-labeling

We use a MLP classification head to predict pseudo-labels for the unlabeled set. As shown in Fig. A.14, the accuracy of the classifier is low in the initial stages and improves as training progresses. Using constraints from incorrectly labeled samples might affect the learning process. Thus, we use confidence (class probabilities) based thresholding to select the samples to be used for pseudo-labeling. Only those samples with confidence higher than the threshold are assigned a pseudo-label. Fig. A.14 shows that the accuracy of the classifier on the confident samples remains high throughout training, limiting the number of incorrect pseudo-labels. Results for threshold value (t) selection are shown in table A.18. As expected, $t = 0$ (i.e, no thresholding) performs poorly compared to higher threshold values. Pseudo-labeling accuracy increases with increasing value of t and the best result is observed for $t = 0.9$. While further increase in t could result in higher pseudo-labeling accuracy, it would also mean that fewer samples are assigned pseudo-labels. Thus, we use $t = 0.9$ in all our experiments on ImageNet100. Since ImageNet-1k has ten times more classes, we reduce the value to 0.85 for all our experiments on ImageNet-1k.

A.7.6 Effect of Caching on Pseudo-label Training

In addition to optimizing the query encoder network using CMSF loss, we train the pseudo-label classifier head at the end of each epoch of query encoder training. Each round of pseudo-label training entails 40 epochs of classifier head training on the supervised subset of the data (10%). While the time required for backward pass is minimal since only the MLP head is updated, forward pass through the encoder adds significant computational overhead. We thus employ encoder feature caching to overcome this issue. We experiment with two caching settings - offline caching and online caching. In offline caching, encoder features for all the supervised samples are calculated once at the beginning of pseudo-label training and kept fixed for the remaining 39 epochs. In online caching, encoder features for the supervised samples are cached for each mini-batch during the query network training. Similar to offline caching, these features are then fixed and used throughout the 40 epochs of pseudo-labeling network training. Offline technique requires one epoch of forward pass through the encoder, but has the advantage of using the most recent model parameters

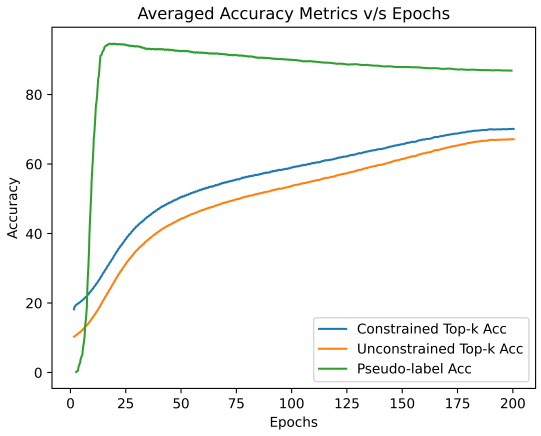


FIGURE A.14. **Unconstrained NN accuracy in semi-supervised:** For analysis, we track the pseudo-labeling accuracy and accuracy of top-k neighbors chosen with and w/o applying the pseudo-label based constraint during training. The more accurate constrained NNs provide a better training signal. Pseudo-label accuracy on confident samples remains high throughout training, decreasing slightly as more confident samples are added.

TABLE A.18. **Role of confidence threshold in pseudo-labeling (ImageNet100 results):** Using confidence based threshold to pseudo-label helps improve performance by eliminating noisy pseudo-labels. A higher threshold value results in higher pseudo-label accuracy but also limits the number of samples that participate in constraint selection. We set the value of t to 0.9 on the ImageNet100 dataset and to 0.85 on the more diverse (1000 classes) ImageNet-1k dataset.

Threshold (t)	1-NN	20-NN	Top-1
0	67.9	71.2	76.2
0.7	67.9	72.3	77.1
0.9	**69.0**	**72.7**	**77.5**

TABLE A.19. **Feature caching for pseudo-label classifier training (ImageNet100 results):** We experiment two different caching schemes for pseudo-label training - offline and online. In offline caching, the features are calculated once at the beginning each round of pseudo-label training while in the online setting, the features are cached for each mini-batch during query encoder training. Since both approaches have similar performance, we use the online version since it has minimal computational overhead.

Method	1-NN	20-NN	Top-1
Offline Caching	67.9	71.2	76.2
Online Caching	66.5	71.9	76.0

for feature calculation. Online caching results in features for different images being calculated using different encoder parameters. We observe that both these settings perform similarly on the ImageNet100 dataset (refer table A.19). We thus use the online version in all our experiments since it has almost no overhead. With this setting, pseudo-label training increases the training time of each epoch approximately by just 40 seconds.

A.7.7 Pseudo-label Classifier Selection

The classifier used to generate pseudo-labels plays a crucial role in obtaining effective constraint sets for $\text{CMSF}_{\text{semi}}$. We experiment with two classification techniques - k-NN classifier and MLP classifier trained with cross-entropy loss. Results on ImageNet100 dataset are shown in table A.20. k-NN classifier has lower pseudo-labeling accuracy and thus results in poorer performance. We additionally experiment with linear, two and three layer architectures for the MLP classifier head. As shown in table A.20, multi-layer head significantly outperform the linear classifier. Since there is minimal difference in performance of two and three layer MLPs, we use a two layer MLP head in all our experiments.

TABLE A.20. **Pseudo-label classifier selection (ImageNet100 results):** We experiment with different classifier methods and architectures for pseudo-label prediction. Linear layer or multi-layer perceptron (MLP) heads trained using cross-entropy loss on the supervised examples outperform a k-NN classifier. MLP classifiers achieve higher accuracy on the pseudo-labeling task on both the train and test sets. We use a two layer MLP head based classifier in all our experiments.

Pseudo-label Classifier	1-NN	20-NN	Top-1
k-NN Classifier	64.9	69.5	74.7
Linear Classifier	65.6	70.0	75.5
2 Layer MLP Head	67.9	71.2	76.2
3 Layer MLP Head	67.1	71.0	76.1

TABLE A.21. **Role of network fine-tuning on classification performance (ImageNet-1k results):** We evaluate the trained models using the linear evaluation technique commonly employed for evaluating self-supervised approaches and entire network fine-tuning as performed in semi-supervised methods. Both methods use 10% of the dataset as supervision. We observe an increase in classification performance when both the encoder and MLP classifier are fine-tuned.

Fine-tune Method	Top-1
Linear layer training	76.5
Full network fine-tune	76.9

A.7.8 Fine-tuning without Pseudo-labels

Since we do not explicitly optimize our encoder networks on the label classification task in the pre-training stage, we perform two-stage fine-tuning. We initially fine-tune the pretrained model on only the supervised samples and use the fine-tuned model to obtain pseudo-labels for the unsupervised ones. The combined data is then used to fine-tune the network again. In table A.21, we present results with just a single round of fine-tuning with the 10% supervised samples on ImageNet-1k. Two rounds of fine-tuning provides a small improvement in performance over the single-stage version.

A.7.9 Results on Supervised Setting

A.7.9.1 Coarse-grained ImageNet

CMSF_{sup} top-*k* groups together only top-*k* neighbors and thus can help in preserving the latent structure of the data compared to top-*all*. To verify this, we consider a dataset with coarse-grained labels where this difference is pronounced. ImageNet dataset was constructed using the WordNet hierarchy. Consider the subtree of WordNet that contains all the 1000 categories from ImageNet-1k as the leaf nodes. To obtain a coarse-grained version, we merge each category in the leaf node to its parent node. After merging, we further ensure that no two of the newly obtained super-classes are in the same path in the graph by merging the descendant into the ancestor class. The total number of classes is thus reduced from 1000 in ImageNet-1k to 93 in our ImageNet-coarse. We train $\text{CMSF}_{\text{self}}$and the baseline approaches in a supervised manner using the coarse labels and then evaluate on the fine-grained (i.e., original) labels on ImageNet-1k validation set. The training settings remain same as that of CMSF_{sup} in Section 4.3.4.2.

In Table A.22 we compare the top-*all*, top-1000 and top-*k* variants on the coarse grained version of ImageNet. We consider the top-1000 variant to limit the effect of dataset imbalance introduced due to the merging of classes. CMSF_{sup} top-*k* sees a minor drop in performance compared to training on ImageNet-1k. However, methods in which most or all samples in a class are explicitly brought closer - CMSF_{sup} top-*all* and top-1000, cross-entropy and supervised contrastive - see a huge drop in accuracy.

TABLE A.22. **Supervised learning on coarse grained ImageNet:** We train on the coarse grained version of ImageNet (93 super categories) and perform linear evaluation on the original ImageNet-1k validation set with fine-grained labels (1000 categories). CMSF_{sup} top-10 outperforms all other variants and baselines.

Train Dataset	ImageNet-1k Validation Set				
	Xent	SupCon	CMSF_{sup} top-*all*	CMSF_{sup} top-1000	CMSF_{sup} top-10
ImageNet-1k	77.2	**77.5**	75.7	-	76.4
ImageNet-coarse	61.4	58.7	67.0	71.0	**74.2**

A.7.9.2 Ablations

We explore different design choices and parameters of our method and baselines. We add the techniques used for our methods to the baselines to isolate the effect of different losses. The results are reported in Table A.23. Training and evaluation details are the same as in Section 4.3.4.2.

TABLE A.23. **Ablations of baselines and CMSF$_{sup}$:** All experiments use 200 epochs if not mentioned and use ImageNet-1k dataset. **(a)** More epochs does not improve transfer accuracy for Xent. Thus, the model available from PyTorch [16] (last row) has the best transfer accuracy; **(b)** We add components of our method to improve SupCon baseline. The baseline implementation of SupCon uses std. aug and 16k memory size and it does not include the target embedding u in the positive set. **(c)** We find that our method is not very sensitive to the size of memory bank or top-k in supervised settings; **(d)** Interestingly, excluding the target embedding u from C does not hurts the results. Note that when we do not include the target, the nearest neighbors are still chosen based on the distance to the target, so they will be close to the target.

Method	Mean Trans	Linear IN-1k
(a) Xent		
lr=0.05, cos, epochs=200, strong aug.	71.5	77.2
lr=0.05, cos, epochs=200, std. aug.	71.0	77.3
lr=0.10, cos, epochs=200, strong aug.	72.3	77.1
lr=0.05, cos, epochs=90, std. aug.	72.4	76.8
lr=0.10, cos, epochs=90, std. aug.	74.0	76.7
lr=0.10, step, epochs=90, std. aug.	74.9	76.2
(b) SupCon		
Base SupCon	77.2	77.9
+ change to strong aug.	77.9	77.4
+ add target to positive set	77.8	77.4
+ change to weak/strong aug.	77.8	77.2
+ increase mem size to 128k	78.4	77.5
(c) CMSF$_{sup}$		
top-1 (BYOL-asym)	74.3	69.3
mem=128k, top-2	78.4	76.2
mem=128k, top-10	80.1	76.4
mem=128k, top-20	79.9	76.3
mem=128k, top-*all*	80.1	75.7
mem=512k, top-10	79.9	76.2
mem=512k, top-20	80.1	76.3
(d) CMSF$_{sup}$		
target in top-10	80.1	76.4
target not in top-10	80.3	76.4

A.7.10 Implementation Details

A.7.10.1 Transfer Learning

We use the LBFGS optimizer (max_iter=20, and history_size=10) along with the Optuna library [21] in the Ray hyperparameter tuning framework [275]. Each dataset gets a budget of 200 trials to pick the best parameters on validation set. The final accuracy is reported on a held-out test set by training the model on the train+val split using the best hyperparameters. The hyperparameters and their search spaces (in loguniform)

TABLE A.24. **ResNet50 backbone training FLOPs calculation:** We provide the number of forward and backward passes per image (including multi-crops) and the total such passes for the entire training stage. Mean Shift and the proposed constrained mean shift methods have the least compute requirement among all approaches. Eq. A.1 provides the formula to calculate the total number of passes and FLOPs. In PAWS, *sup* refers to the size of the support set in the mini-batch.

Method	Unlabeled			Labeled			Mini-Batch	Iters per epoch	Epochs	Total Pass ($\times 10^8$)	FLOPs ($\times 10^{18}$)
	Fwd	Bwd	BS	Fwd	Bwd	BS					
Mean Shift [242]	2	1	256				768	5004	200	7.7	4
BYOL [160]	4	2	4096				24576	312	1000	76.7	40
SwAV [57]	3.1	3.1	4096				25395	312	800	63.4	37
SimCLRv2 [72]	2	2	4096				16384	312	800	40.9	16
UDA† [484]	2	1	15360	1	1	512	47104	40000		18.8	10
FixMatch† [403]	2	1	5120	1	1	1024	17408	250	300	13.1	70
MPL† [344]	3	2	2048	2	2	128	10752	500000		53.8	30
PAWS (sup=6720) [26]	3.1	3.1	4096	1	1	6720	38835	312	300	36.6	21
PAWS (sup=1680) [26]	3.1	3.1	256	1	1	1680	4947	5004	100	24.8	15
PAWS (sup=400) [26]	3.1	3.1	256	1	1	400	2387	5004	100	12.0	7
$\text{CMSF}_{\text{semi}}$-basic	2	1	256				768	5004	200	7.7	4
$\text{CMSF}_{\text{semi}}$	2	1	256				768	5004	200	7.7	4
$\text{CMSF}_{\text{semi}}$-mix prec.	2	1	768				2304	1668	200	7.7	4

are as follows: iterations $\in [0, 10^3]$, lr $\in [10^{-6}, 1]$, and weight decay $\in [10^{-9}, 1]$. We also show that we can reproduce the transfer results for BYOL [160] and SimCLR [70] with our framework. The features are extracted with the following pre-processing for all datasets: resize shorter side to 256, take a center crop of size 224, and normalize with ImageNet statistics. No training time augmentation was used.

A.7.10.2 Supervised Setting

Implementation Details of Baselines:

SupCon: The MLP architecture for SupCon baseline is: linear (2048x2048), batch norm, ReLU, and linear (2048x128). To optimize the SupCon baseline, following [230], we use the first 10 epochs for learning-rate warmup. For both SupCon and ProtoNW, the temperature is 0.1.

Prototypical Networks (ProtoNW): In order to further study the effect of contrast, we design another contrastive version of our top-*all* variation. We calculate a prototype for each class by averaging all its instances in the memory bank. Then, similar to prototypical networks [401], we compare the input with all prototypes by passing their temperature-scaled cosine distance through a SoftMax layer to get probabilities. Finally, we minimize the cross-entropy loss. Note that this method is still contrastive in nature because of the SoftMax operation.

TABLE A.25. **Noisy supervised setting on ImageNet-100:** Our method is more robust to noisy annotation compared to Xent and SupCon. The top-*all* variant suffers greater degradation compared to top-10 since all images from a single category are not guaranteed to be semantically related in the noisy setting.

Method	Noise	Food 101	CIFAR 10	CIFAR 100	SUN 397	Cars 196	Air-craft	DTD	Pets	Calt. 101	Flwr 102	**Mean Trans**	**Linear IN-100**
Xent	0%	53.6	81.9	61.1	37.8	25.7	29.5	56.9	69.7	70.2	82.3	56.9	85.7
SupCon	0%	61.5	88.7	69.0	49.1	51.6	48.2	65.4	81.0	87.0	89.8	69.1	86.9
$CMSF_{sup}$ top-*all*	0%	61.6	88.2	68.5	49.9	54.6	52.7	64.7	82.2	89.6	89.1	70.1	84.9
$CMSF_{sup}$ top-10	0%	62.6	86.8	66.2	50.5	54.7	51.0	64.6	82.4	88.5	90.4	69.8	85.0
Xent	5%	46.5	81.1	58.1	35.8	27.5	36.0	58.7	67.5	73.3	77.0	56.1	81.5
SupCon	5%	60.0	87.1	66.4	48.2	52.1	47.8	65.1	80.8	85.7	89.3	68.3	85.7
$CMSF_{sup}$ top-*all*	5%	60.3	87.5	66.4	49.1	55.5	53.0	64.8	80.9	87.3	89.9	69.5	84.4
$CMSF_{sup}$ top-10	5%	61.6	86.8	67.4	49.6	55.8	51.2	63.4	81.5	86.7	90.6	69.5	84.7
Xent	10%	44.1	79.5	56.1	32.4	26.1	34.5	56.1	69.7	72.5	75.1	54.6	79.6
SupCon	10%	58.8	85.8	66.4	47.0	50.6	47.7	65.3	79.8	85.0	89.1	67.6	84.0
$CMSF_{sup}$ top-*all*	10%	59.4	86.4	66.0	48.8	55.0	51.4	64.7	80.1	87.8	89.0	68.9	83.1
$CMSF_{sup}$ top-10	10%	60.9	87.2	66.9	49.4	54.2	51.4	65.5	80.6	88.5	90.0	69.5	83.8
Xent	25%	49.0	77.2	54.5	30.6	25.9	30.7	53.1	66.6	64.1	77.8	53.0	75.2
SupCon	25%	55.6	84.9	63.4	43.1	43.9	43.7	62.9	74.3	82.1	86.8	64.1	81.1
$CMSF_{sup}$ top-*all*	25%	56.4	85.7	64.2	46.0	53.6	49.6	62.7	74.2	85.2	87.4	66.5	78.8
$CMSF_{sup}$ top-10	25%	58.9	85.2	64.9	47.8	55.0	50.6	64.0	80.0	86.3	89.7	68.2	81.8
Xent	50%	44.4	72.3	51.3	31.1	21.4	24.9	46.0	57.4	56.0	73.0	47.8	67.8
SupCon	50%	30.8	64.9	38.9	24.2	13.6	20.5	45.5	55.2	60.1	59.2	41.3	69.0
$CMSF_{sup}$ top-*all*	50%	44.7	79.3	54.9	35.2	35.7	41.2	54.9	54.6	75.3	75.1	55.1	61.6
$CMSF_{sup}$ top-10	50%	58.7	85.7	64.2	47.5	51.6	50.5	62.0	77.3	86.8	70.1	65.4	80.1

A.7.10.3 Semi-supervised Setting

Pretraining: Similar to the self-supervised setting, we train the network for 200 epochs using SGD optimizer (batch size=256, lr=0.05, momentum=0.9, weight decay=1e-4). Ten nearest neighbors are chosen from the constraint set for loss calculation. The size of memory bank is set to 128000. We train the pseudo-label classifier using an additional SGD optimizer (batch size=256, lr=0.01, momentum=0.9, weight decay=1e-4) for 10 epochs at the end of each epoch of query encoder training. A confidence threshold value of 0.85 is used to assign pseudo-labels to the unlabeled samples.

Fine-tuning: In addition to pretraining, we use a two layer MLP atop the CNN backbone and fine-tune the entire network on the supervised subset for 20 epochs. This fine-tuned network is used to pseudo-label the unlabeled set with a confidence threshold of 0.9. Samples above the threshold are combined with the supervised set for a second round of fine-tuning for 20 epochs. We observe that nearly one third of the samples in the dataset have confidence higher than the threshold at the end of the first fine-tuning stage. We use a SGD optimizer (batch size=256, lr=0.005, momentum=0.9, weight decay=1e-4) for both the fine-tuning stages.

TABLE A.26. **Transfer dataset details:** Train, val, and test splits of the transfer datasets are listed in this table. **Test split:** We follow the details in [242]. For Aircraft, DTD, and Flowers datasets, we use the provided test sets. For Sun397, Cars, CIFAR-10, CIFAR-100, Food101, and Pets datasets, we use the provided val set as the hold-out test set. For Caltech-101, 30 random images per category are used as the hold-out test set. **Val split:** For DTD and Flowers, we use the provided val sets. For other datasets, the val set is randomly sampled from the train set. For transfer setup, to be close to BYOL [160], the following val set splitting strategies have been used for each dataset: Aircraft: 20% samples per class. Caltech-101: 5 samples per class. Cars: 20% samples per class. CIFAR-100: 50 samples per class. CIFAR-10: 50 samples per class. Food101: 75 samples per class. Pets: 20 samples per class. Sun397: 10 samples per class. **Accuracy measure:** *Top-1* refers to top-1 accuracy while *Mean* refers to mean per-class accuracy.

Dataset	Classes	Train samples	Val samples	Test samples	Accuracy measure	Test set provided
Food101 [41]	101	68175	7575	25250	Top-1	-
CIFAR-10 [245]	10	49500	500	10000	Top-1	-
CIFAR-100 [245]	100	45000	5000	10000	Top-1	-
Sun397 (split 1) [481]	397	15880	3970	19850	Top-1	-
Cars [244]	196	6509	1635	8041	Top-1	-
Aircraft [301]	100	5367	1300	3333	Mean	Yes
DTD (split 1) [90]	47	1880	1880	1880	Top-1	Yes
Pets [333]	37	2940	740	3669	Mean	-
Caltech-101 [130]	101	2550	510	6084	Mean	-
Flowers [321]	102	1020	1020	6149	Mean	Yes

The learning rate is multiplied by 0.1 at the end of epoch 15.

Calculation of forward and backward FLOPs: In figure 1 of the main submission, we present a plot of top-1 accuracy against total compute and resources for various semi-supervised approaches. Here (table A.24) we present the calculation of the forward and backward FLOPS for each of the methods. We set the backward FLOPs to be twice the forward number of FLOPs [178] for a single image and the total FLOPs to be the sum of forward and backward pass FLOPs for the entire training. We use a value of 3.9 GFLOPs for a single forward pass of 224×224 resolution image through the ResNet50 backbone [198]. Additional compute due to the use of multi-crops are accounted for. A scalar multiplier of $(\frac{K}{224})^2$ is used for images of resolution $K \times K$ (e.g., using one (96×96) image would be equivalent to 0.184 image of resolution (224×224)). However, we do not consider the floating point precision (mixed or full precision) in our calculations. We show that similar performance can be achieved by using both automatic mixed precision and full precision floating point during training (table 4, main submission) and thus focus the compute calculation on the total number of forward and backward passes. Eq. A.1 provides the formula to calculate the total number of training passes and FLOPs.

$$
\begin{aligned}
\text{Fwd mini-batch} &= (\text{Unlabeled fwd crops} * \text{Unlabeled batch-size}) \\
&\quad + (\text{Labeled fwd crops} * \text{Labeled batch-size}) \\
\text{Bwd mini-batch} &= (\text{Unlabeled bwd crops} * \text{Unlabeled batch-size}) \\
&\quad + (\text{Labeled bwd crops} * \text{Labeled batch-size}) \\
\text{Fwd passes} &= \text{Fwd mini-batch} * \text{Iterations per epoch} * \text{Epochs} \\
\text{Bwd passes} &= \text{Bwd mini-batch} * \text{Iterations per epoch} * \text{Epochs} \\
\text{Total FLOPs} &= (\text{Fwd passes} + 2 * \text{Bwd passes}) * (3.9 \times 10^9)
\end{aligned}
\tag{A.1}
$$

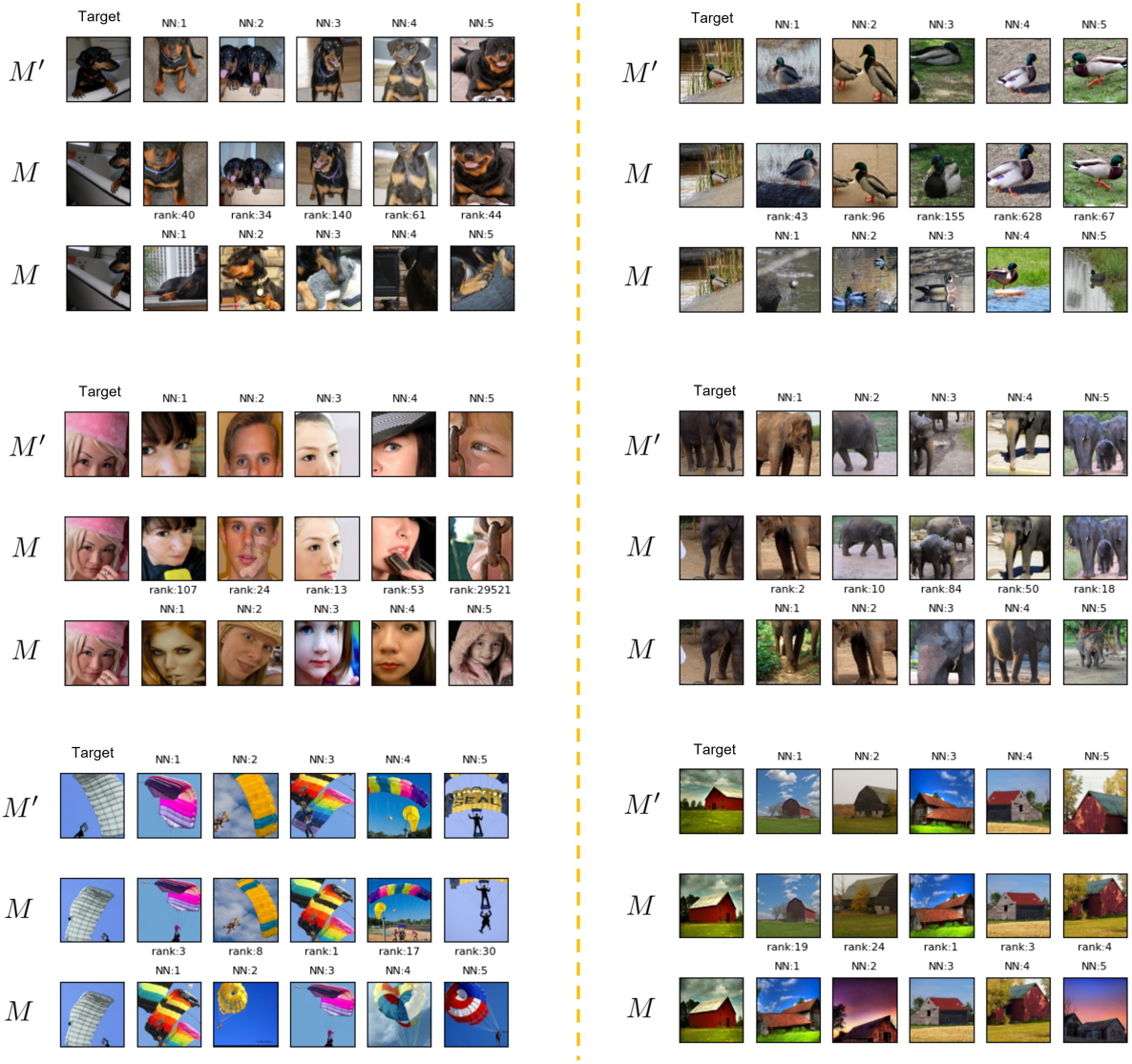


FIGURE A.15. **$\text{CMSF}_{\text{self}}$ nearest neighbor selection:** We use epoch 100 of $\text{CMSF}_{\text{self}}$ to visualize Top-5 NN from primary (M) and auxiliary (M') memory banks. M stores features for the current epoch while M' contains representations from a different augmentation of the same image instance from the previous epoch. First row shows the target image and its top-5 NNs from the auxiliary memory bank M'. Samples of the second row are the images in M corresponding to the ones in row 1. Thus, rows 1 and 2 contain different augmentations of the same image instances. We also report their rank in M in row 2. The last row contains the top-5 NNs in M. Note that constrained samples in M (second row), have high rank while they are semantically similar to the target.

A.8 Adversarial Attack on Compute of Efficient Vision Transformers

In sections A.8.1 and A.8.2, we provide visualizations of our learnt patches and token dropping respectively. In Sec. A.8.3, we provide additional details on our train and test settings.

A.8.1 Patch Visualization

In Fig. A.16, we visualize the optimized patches for each of the three efficient methods. All patches are of size 64×64, contributing to 16 of the 196 tokens for the input image. Surprisingly, a rectangular region in the patch for A-ViT , corresponding to one token, is almost entirely black.

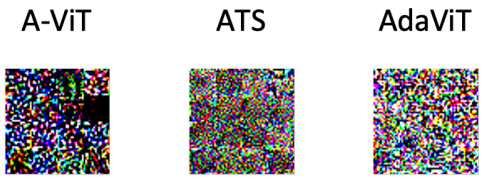


FIGURE A.16. **Visualization of optimized patch:** We show the learnt universal patches for each of the three efficient methods that we attack.

We optimize patches for A-ViT using different initializations and visualize them in Fig. A.17. All patches achieve Attack Success close to 100%. Presence of multiple universal adversarial patches highlights the vulnerability of the current efficient methods.

We show the evolution of the patch as training progresses in Fig. A.19. The patch is trained to attack A-ViT approach. We observe that the patch converges quickly, requiring less than an epoch for 100% Attack Success. The patch at 1000 iterations (0.1 epoch) is similar to that at 10000 iterations (1 epoch) in terms of both appearance and attack performance.

A.8.2 Visualization of Token Dropping

In Fig. A.18, we visualize dropped tokens in A-ViT-Small with and without our attack. Our attack significantly decreases the number of pruned tokens, resulting in more compute and energy consumption for the efficient transformer model.

A.8.3 Implementation Details

ATS Details: As in ATS [127], we replace layers 3 through 9 of ViT networks with the ATS block and set the maximum limit for the number of tokens sampled to 197 for each layer. We train the patch

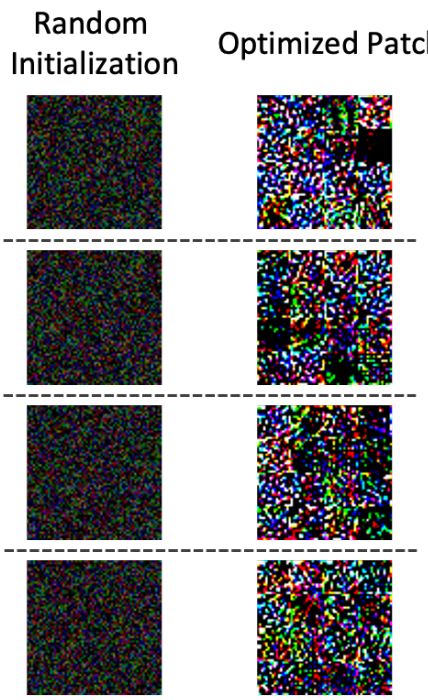


FIGURE A.17. **Optimized patches With different initializations:** Here, we show the optimized patches for A-ViT . A different initialization is used to train each of these patches. All patches achieve Attack Success close to 100%. Presence of multiple universal adversarial patches highlights the vulnerability of the current efficient methods.

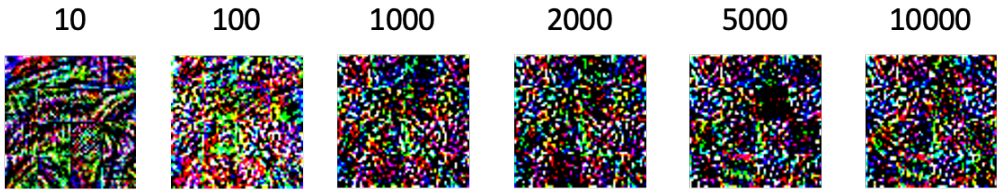


FIGURE A.18. **Visualization of patch optimization:** We train our patch to attack A-ViT and display the patch at various stages of optimization. We observe that the patch converges quickly. The patch at 1000 iterations (0.1 epoch) is similar to that at 10000 iterations (1 epoch) in terms of both appearance and attack performance.

for 2 epochs with a learning rate of 0.4 for ViT-Tiny and $lr = 0.2$ for ViT-Base and ViT-Small. We use a batch size of 1024 and different loss coefficients for each layer of ATS. For DeiT-Tiny we use $[1.0, 0.2, 0.2, 0.2, 0.01, 0.01, 0.01]$, for DeiT-Small we use $[1.0, 0.2, 0.05, 0.01, 0.005, 0.005, 0.005]$, and for DeiT-Base we use $[2.0, 0.1, 0.02, 0.01, 0.005, 0.005, 0.005]$ The weights are vastly different at initial and final layers to account for the difference in loss magnitudes across layers.

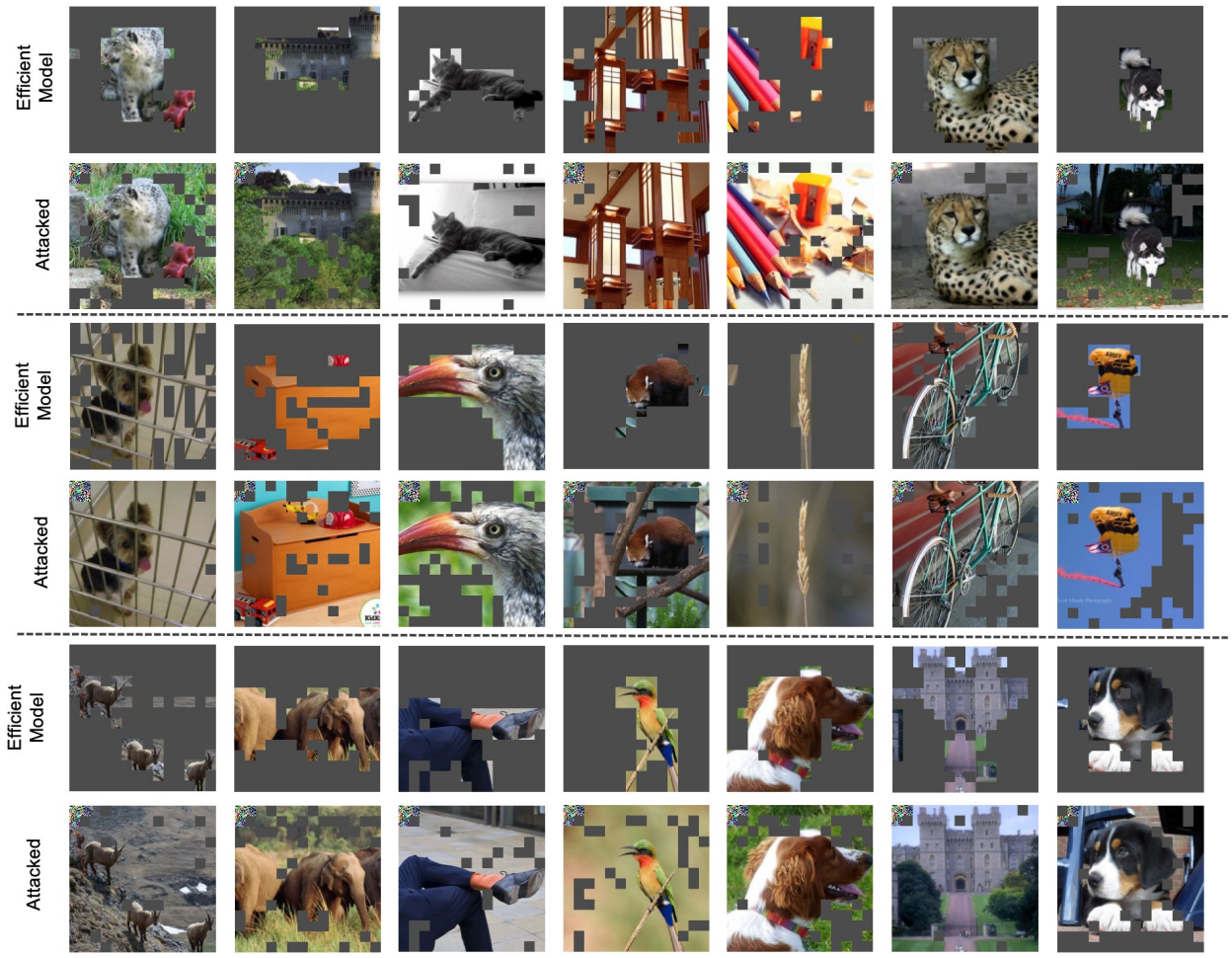


FIGURE A.19. **Visualization of our Energy Attack on Vision Transformers:** Similar to Figure 2 of the main submission, we visualize the A-ViT-Small with and without our attack. We use patch size of 32 for the attack (on the top-left corner). We show pruned tokens at layer 8 of A-ViT-Small. Our attack can recover most of the pruned tokens, resulting in increased computation and power consumption.

A-ViT Details: The patches are optimized for one epoch with a learning rate of 0.2 and a batch size of 512 (128 × 4GPUs) using AdamW [292] optimizer. We optimize the patches for 4 epochs for patch length 32 and below. For CIFAR-10 experiments, the images are resized from 32 × 32 to 256 × 256 and a 224 × 224 crop is used as the input. For the training of adversarial defense, we generate 5 patches per epoch of adversarial training and limit the number of iterations for patch generation to 500. The learning rate for patch optimization is increased to 0.8 for faster convergence.

AdaViT Details: We use a learning rate of 0.2 and a batch size of 128 with 4GPUs for patch optimization. We use AdamW [292] optimizer with no decay and train for 2 epochs with a patch size of 64 x 64. We train on the ImageNet-1k train dataset and evaluate it on the test set.

A.9 GeNIe: Generative Hard Negative Images Through Diffusion

A.9.1 Analyzing GeNIe, GeNIe-Ada's Class-Probabilities

The core aim of GeNIe and GeNIe-Ada is to address the failure modes of a classifier by generating *challenging* samples located near the decision boundary of each class pair, which facilitates the learning process in effectively enhancing the decision boundary between classes. As summarized in Table 6.5 and illustrated in Fig. 6.5, we have empirically corroborated that GeNIe and GeNIe-Ada can respectively produce samples X_r, X_{r^*} that are negative with respect to the source image X_S, while semantically belonging to the class T.

To further analyze the effectiveness of GeNIe and GeNIe-Ada, we compare the source class-probabilities $P(Y_S|X_r)$ and target-class probabilities $P(Y_S|X_r)$ of augmented samples X_r. To compute these class probabilities, we first fit an SVM classifier (as followed in UniSiam [295]) only on the labelled support set embeddings of each episode in the *mini*Imagenet test dataset. Then, we perform inference using each episode's SVM classifier on its respective X_r's and extract its class probabilities of belonging to its source class S and target class T. These per augmentation-sample source and target class probabilities are then averaged for each episode for each $r \in \{0.5, 0.6, 0.7, 0.8, 0.9\}$ in the case of GeNIe and for the optimal $r = r^*$ per sample in the case of GeNIe-Ada, plotted as density plots in Fig. 6.6, Fig. A.20, respectively. Fig. 6.6 illustrates that $P(Y_S|X_r)$ and $P(Y_T|X_r)$ have significant overlap in the case of $r \in \{0.6, 0.7\}$ indicating class-confusion for X_r.

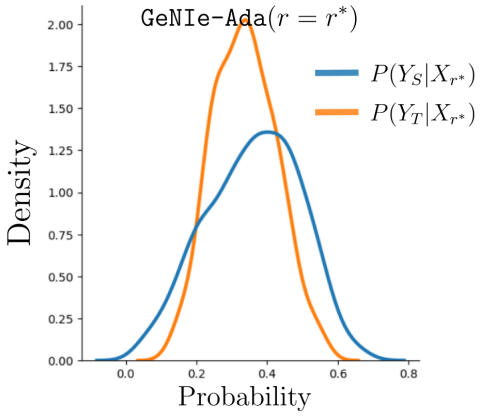


FIGURE A.20. Significant overlap between $P(Y_S|X_{r^*})$ and $P(Y_T|X_{r^*})$ indicates high class-confusion for augmented samples generated by GeNIe-Ada.

Furthermore, Fig. A.20 illustrates that when using the optimal $r = r^*$ found by GeNIe-Ada per sample, $P(Y_S|X_r)$ and $P(Y_T|X_r)$ significantly overlap around probability scores of 0.2 – 0.45, indicating class confusion for GeNIe-Ada augmentations. This corroborates with our analysis in Section 6.1.3.4, Table 6.5 and additionally empirically proves that the augmented samples generated by GeNIe for $r \in \{0.6, 0.7\}$ and GeNIe-Ada for $r = r^*$ are actually located near the decision boundary of each class pair.

A.9.2 Computational Complexity of GeNIe and GeNIe-Ada

In this section, we provide further details on the computational complexity of GeNIe across multiple noising ratios r and GeNIe-Ada when operating on a search space of $r \in [0.6, 0.8]$. Computational complexity has been reported in terms of the total number of inference/denoising-diffusion steps and the runtime in seconds per generated image. The runtime has been averaged over 10 different image-generations on an NVIDIA Tesla-V100 GPU with 16GB VRAM with 50 steps of denoising using a DPM scheduler with StableDiffusion v1.5. As can be seen in Tab. A.28, GeNIe is approximately $1/r$ times faster than the base diffusion model (referred to as the Txt2Img augmentation baseline). This empirically corroborates with the total number of denoising steps using in GeNIe vs. Txt2Img. Since, GeNIe-Ada scans for the best hard-negative in $r \in [0.6, 0.8]$, it incurs a computational cost of $\approx 2.2\times$ the Txt2Img. Note that the runtime for GeNIe-Ada reported in Tab. A.28 also includes the runtime of performing a batched forward pass through a ResNet-50 feature extraction backbone.

TABLE A.27. Train and test split details of the fine-grained datasets. We use the provided train set for few-shot task generation, and the provided test sets for our evaluation. Aircraft dataset uses the manufacturer hierarchy.

Dataset	Classes	Train samples	Test samples
CUB200 [452]	200	5994	5794
Food101 [41]	101	75750	25250
Cars [244]	196	8144	8041
Aircraft [301]	41	6,667	3333

TABLE A.28. Computational Complexity

Augmentation	Steps	Runtime [sec/img]
Txt2Img	T	4.12
GeNIe(r=0.5)	$0.5 \times T$	2.17
GeNIe(r=0.6)	$0.6 \times T$	2.59
GeNIe(r=0.7)	$0.7 \times T$	2.98
GeNIe(r=0.8)	$0.8 \times T$	3.46
GeNIe-Ada	$2.1 \times T$	9.22

A.9.3 Extra Computation of GeNIe-Ada

Given that GeNIe-Ada searches for the best hard-negative between multiple noise-ratios r's, it naturally requires a higher compute budget than txt2Img that only uses $r = 1$. For this experiment, we use GeNIe-Ada with $r \in \{0.6, 0.7, 0.8\}$ to compare with Txt2Img. Based on this, we only have 3 paths, with steps of 0.1), and for each of which we go through partial reverse diffusion process. E.g. for $r = 0.6$ we do 30 steps instead of standard 50 steps of Stable Diffusion. This practically breaks down the total run-time of GeNIe-Ada to approximately 2 times that of the standard reverse diffusion (GeNIe-Ada: total $r = 0.6 + 0.7 + 0.8 = 2.1$ vs Txt2Img total $r = 1$). Thus, to be fair, we generate twice as many Txt2Img augmentations as compared to GeNIe-Ada to keep a constant compute budget across the methods, following your suggestion. The

results are shown in Table A.29. As can be seen, even in this new setting, GeNIe-Ada offers a performance improvement of 0.8% to 1.9% across different backbones.

TABLE A.29. Few-shot classification comparison of GeNIe-Ada with `Txt2Img` on miniImagenet.

Method	ResNet-18		ResNet-34		ResNet-50	
	1-shot	5-shot	1-shot	5-shot	1-shot	5-shot
`Txt2Img`	76.9±1.0	86.5±0.9	77.1±0.8	86.7±1.0	77.2±1.3	86.8±0.9
`GeNIe-Ada`	77.7±0.8	87.4±1.0	78.3±0.9	87.8±0.9	79.1±1.1	88.4±1.2

A.9.4 Effect of Backbone for Noise Ratio Selector in `GeNIe-Ada`

To analyze the effect of the backbone feature extractor f_θ on selecting the optimal hard-negative using `GeNIe-Ada`, we use a pre-trained DeiT-B [430] instead of the UniSiam pretrained ResNet backbone. However, we still utilize the same ResNet backbone for few-shot classification. As shown in Tab. A.30, we notice a marginal improvement of upto 0.7% when using `GeNIe-Ada`+DeiT-B as compared to `GeNIe-Ada` which uses the UniSiam pre-trained ResNet backbone. This suggests that there is still potential to develop more effective strategies for selecting noise ratios to further enhance `GeNIe`. However, in this paper, we limit our exploration to `GeNIe-Ada` and leave these improvements for future work.

TABLE A.30. **Effect of Backbone for Noise Ratio Selector in `GeNIe-Ada`:** We evaluate the impact of the noise ratio selector used in `GeNIe-Ada` ($f_\theta(.)$). Note that in all experiments presented in this paper, we use the same backbone for $f_\theta(.)$ that is subsequently fine-tuned for few-shot classification tasks. However, to analyze the effect of $f_\theta(.)$ on sampled augmentations, we replace it with a more powerful backbone, specifically DeiT-B pretrained on ImageNet-1K. It is important to note that this is not a practical assumption; if DeiT-B were available for noise selection, it could also be used as the classifier in few-shot experiments, outperforming the weaker backbones employed in our study. Nevertheless, this experiment demonstrates that using a stronger backbone can result in more accurate selection of augmentations in `GeNIe`, thereby enhancing the final accuracy. To clarify, DeiT-B is utilized solely as $f_\theta(.)$ for sampling augmentations and not as the classifier. Therefore, the observed improvement is attributed exclusively to better augmentation sampling.

ResNet-18				
Augmentation	Noise Ratio Selector Backbone $f_\theta(.)$	Method [Classifier Backbone]	1-shot	5-shot
`GeNIe` (Ours)	-	UniSiam[ResNet18]	**75.5±0.6**	**85.4±0.4**
`GeNIe-Ada`	UniSiam[ResNet18]	UniSiam[ResNet18]	**76.8±0.6**	**85.9±0.4**
`GeNIe-Ada`	IN-1K[DeiT-B]	UniSiam[ResNet18]	**77.5±0.5**	**86.3±0.2**

ResNet-50				
Augmentation	Noise Selector Backbone $f_\theta(.)$	Method [Classifier Backbone]	1-shot	5-shot
`GeNIe`	-	UniSiam[ResNet50]	**77.3±0.6**	**87.2±0.4**
`GeNIe-Ada`	UniSiam[ResNet50]	UniSiam[ResNet50]	**78.6±0.6**	**87.9±0.4**
`GeNIe-Ada`	IN-1K[DeiT-B]	UniSiam[ResNet50]	**79.2±0.4**	**88.3±0.5**

A.9.5 Psuedocode of `GeNIe`:

As illustrated in Alg. 3, we provide a detailed pytorch-style pseudocode for `GeNIe`. First, a SDv1.5 pipeline initialized by loading all the components such as the VQ-VAE encoder and decoder, the CLIP text encoder and the DPM scheduler for the forward and reverse diffusion process. Then, the source image is

input to the encoder to encode the image into latent space for the diffusion model. Next, the encoded image is partially noised based on the noise ratio r using the scheduler. The diffusion model then de-noises the partially noised latent embedding for a total of NUM INFERENCE STEPS $\times r$ steps, with an additional input of a text prompt from a contradictory target class. Finally, the decoder decodes the de-noised latent embedding into the generated hard-negative image, that contains the low-level features of the source image and the class/category of the contradictory text-prompt.

Algorithm 3: PyTorch-style Pseudocode of GeNIe.

```
# StableDiffusionPipeline: Pre-trained diffusion model
# DPMSolverMultistepScheduler: Scheduler for forward and reverse diffusion
# encode_latents: Encodes an image into latent space
# decode_latents: Decodes latents back into an image

def AugmentGeNIe(source_image, target_prompt, percent_noise):
    NUM_INFERENCE_STEPS = 50 # Number of steps for reverse diffusion
    NUM_TRAIN_STEPS = 1000 # Number of steps for forward diffusion

    # Initialize the stable diffusion pipeline and scheduler
    pipe = StableDiffusionPipeline.from_pretrained("stable-diffusion-v1-5")
    scheduler = DPMSolverMultistepScheduler.from_config(pipe.scheduler.config)

    # Encode the source image into latent space
    latents = encode_latents(source_image)

    # Forward Diffusion
    noise = torch.randn(latents.shape) # Generate random noise
    timestep = torch.Tensor([int(NUM_TRAIN_STEPS * percent_noise)]) # Calculate timestep
    latents_noise = scheduler.add_noise(latents, noise, timestep) # Add noise to latents

    # Reverse Diffusion
    latents = pipe(
        prompt=target_prompt,
        percent_noise=percent_noise,
        latents=latents_noise,
        num_inference_steps=NUM_INFERENCE_STEPS
    )

    # Decode latents back into an augmented image
    augmented_image = decode_latents(latents)

    return augmented_image
```

A.9.6 How does GeNIe control which features are retained or changed?

We instruct the diffusion model to generate an image by combining the latent noise of the source image with the textual prompt of the target category. This combination is controlled by the amount of added noise and the number of reverse diffusion iterations. This approach aims to produce an image that aligns closely with the semantics of the target category while preserving the background and features from the source image that are unrelated to the target.

To demonstrate this, in Figure A.21, We are progessivley moving towards the two key components of GeNIe: (i) careful choice of r and (ii) contradictory prompt. The input image is a bird in a cage. The top row shows a Stable Diffusion model, unprompted. As can be seen, such a model can generate anything (irrespective of the input image) with a large r. Now prompting the same model with "a photo of a bird"

allows the model to preserve low-level and contextual features of the input image (up to $r = 0.7$ and 0.8), until for a large $r \geq 0.9$ it returns a bird but the context has nothing to do with the source input. This illustrates how a careful choice of r can help preserve such low-level features, and is a key idea behind GeNIe. However, we also need a semantic switch to a different target class as shown in the last row where a hardly seen image of a dog in a cage is generated by a combination of a careful choice of r and the contradictory prompt - leading to the full mechanics of GeNIe. This sample now serves as hard negative for the source image (bird class).

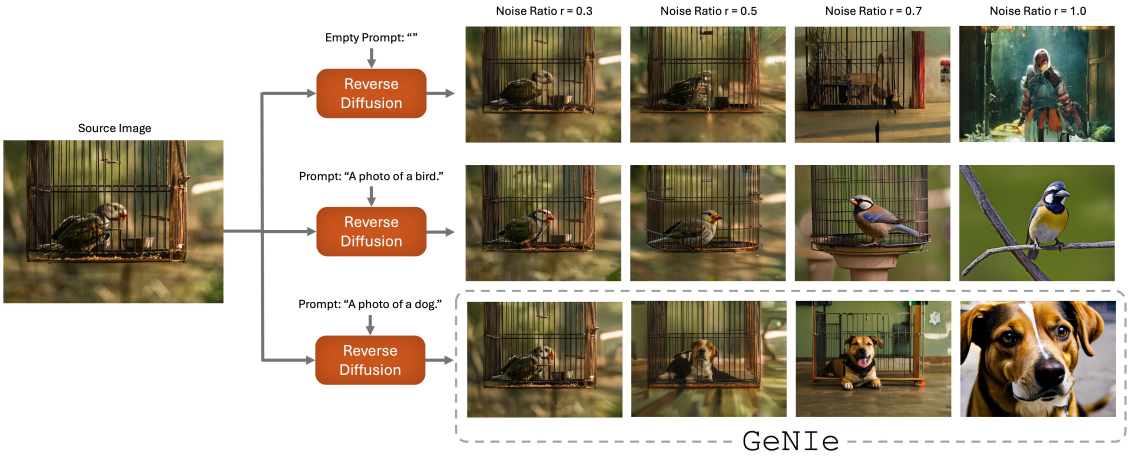


FIGURE A.21. **Key components of GeNIe:** (i) careful choice of r and (ii) contradictory prompt are two key idea behind GeNIe

A.9.7 Analyzing Noise Effects in Bi-Directional Transformations with GeNIe

To further explore the effect of noise ratio r in GeNIe, we conducted an experiment where GeNIe was applied twice to transform between a source image and a target category. For this experiment, images from the "mushroom" category were used as the source, and "volcano" served as the target category. In the first step, we applied GeNIe using a mushroom image as the source and a volcano prompt as the target. In the second step, we reversed the process: the GeNIe-generated volcano image from the first step was used as the source, with the target prompt set to mushroom. Importantly, using a smaller noise ratio, r during the generation of the volcano image helps preserve more low-level visual features from the original mushroom source image. Consequently, when the roles of source and target are flipped in the second step, the final image retains a stronger resemblance to the original mushroom source image for lower noise ratios. This phenomenon is visualized in Fig. A.22. As shown, a lower noise ratio during the first step results in the preservation of more visual features, leading to a final image that more closely resembles the original mushroom source.

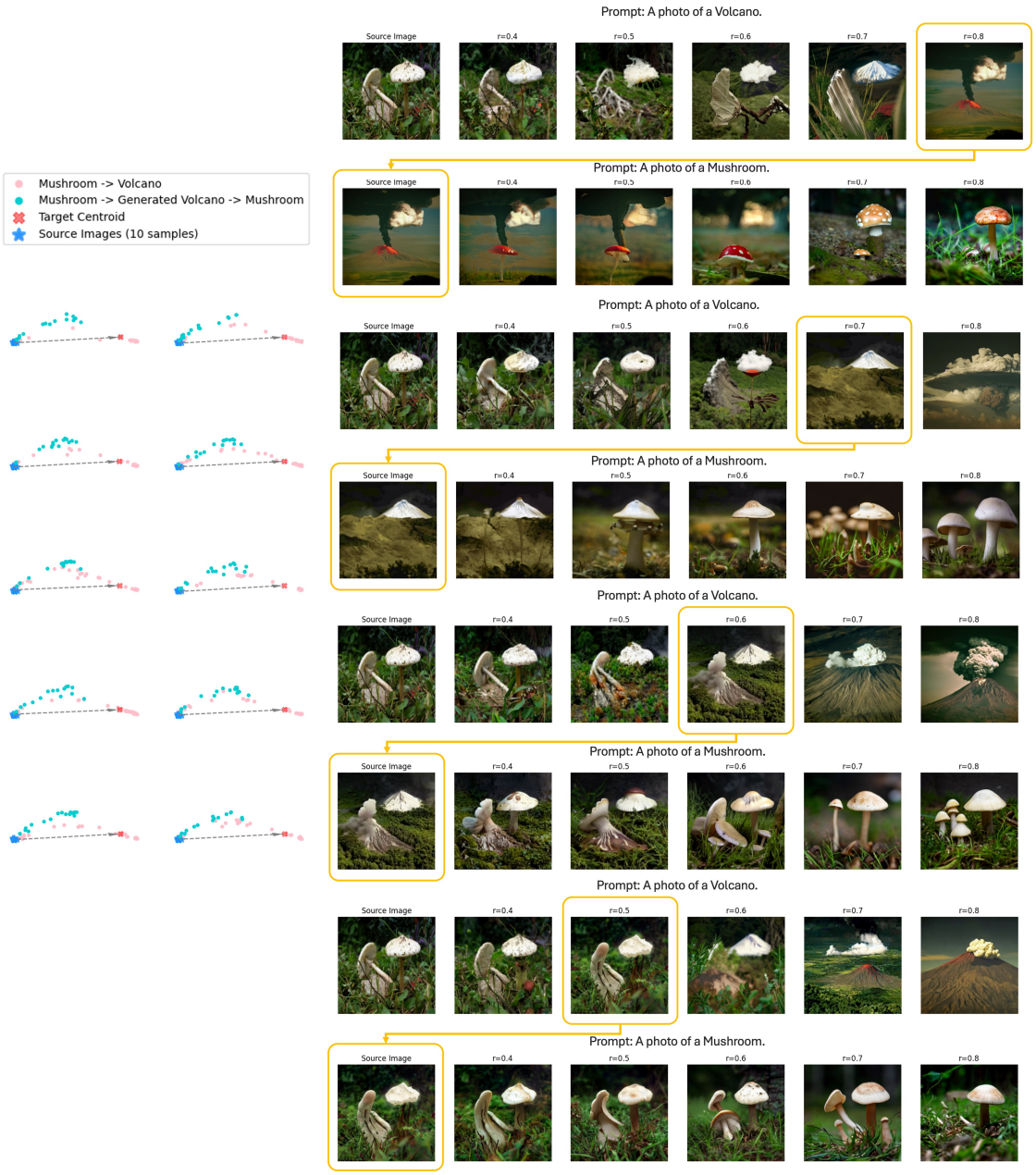


FIGURE A.22. **Trajectory of GeNIe augmentations:** To further analyze the effect of noise ratio r in GeNIe, we conducted an experiment using a set of augmentations generated from 10 different source images in the "mushroom" category, with a target label of "Volcano," across varying noise ratios. Similar to Fig. 6.5, all generated augmentations were processed through the DinoV2 ViT-G model, which serves as our oracle, to extract their embeddings. For visualization, we applied PCA to these embeddings. Next, we selected one augmentation with a specific noise ratio, (r), and used it as the source image in for the "volcano" category in GeNIe, with the target prompt set to "mushroom." As observed, using a lower noise ratio samples as the source for "volcano" preserves more low-level visual features from the original mushroom source image. Consequently, after a second round of applying GeNIe, the resulting augmentations (even rows) tend to more closely resemble the original source image (first image in the corresponding odd rows above). The left plot presents the embeddings of all 10 samples, while the right plot provides a detailed visualization of one sample, showcasing the impact of varying noise ratios used in the second step of applying GeNIe.

A.9.8 Few-shot Classification with ResNet-34 on *tiered*-Imagenet

We follow the same evaluation protocol here as mentioned in section 6.1.3.1. As summarized in Table A.31, `GeNIe` and `GeNIe-Ada` outperform all other data augmentation techniques.

TABLE A.31. ***tiered*-ImageNet:** Accuracies (% ± std) for 5-way, 1-shot and 5-way, 5-shot classification settings on the test-set. We compare against various SOTA supervised and unsupervised few-shot classification baselines as well as other augmentation methods, with UniSiam [295] pre-trained ResNet-34.

ResNet-34				
Augmentation	**Method**	**Pre-training**	**1-shot**	**5-shot**
Weak	MAML + dist [137]	sup.	51.7±1.8	70.3±1.7
Weak	ProtoNet [400]	sup.	52.0±1.2	72.1±1.5
Weak	UniSiam + dist [295]	unsup.	68.7±0.4	85.7±0.3
Weak	UniSiam [295]	unsup.	65.0±0.7	82.5±0.5
Strong	UniSiam [295]	unsup.	64.8±0.7	82.4±0.5
CutMix [513]	UniSiam [295]	unsup.	63.8±0.7	80.3±0.6
MixUp [521]	UniSiam [295]	unsup.	64.1±0.7	80.0±0.6
`Img2Img`^{L} [297]	UniSiam [295]	unsup.	66.1±0.7	83.1±0.5
`Img2Img`^{H} [297]	UniSiam [295]	unsup.	70.4±0.7	84.7±0.5
`Txt2Img` [180]	UniSiam [295]	unsup.	75.0±0.6	85.4±0.4
DAFusion [438]	UniSiam [295]	unsup.	64.1±2.1	82.8±1.4
`GeNIe` (Ours)	UniSiam [295]	unsup.	**75.7±0.6**	**86.0±0.4**
`GeNIe-Ada` (Ours)	UniSiam [295]	unsup.	**76.9±0.6**	**86.3±0.2**

A.9.9 Additional details of Long-Tail experiments

In Table A.32, we present a comprehensive version of Table 6.3 to benchmark the performance with different backbone architectures (e.g., ResNet50) and to compare against previous long-tail baselines.

Implementation Details of LViT: We download the pre-trained ViT-B of LViT [493] and finetune it with Bal-BCE loss proposed therein on the augmented dataset. Training takes 2 hours on four NVIDIA RTX 3090 GPUs. We use the same hyperparameters as in [493] for finetuning: 100 epochs, lr = 0.008, batch size of 1024, CutMix and MixUp for the data augmentation.

Implementation Details of VL-LTR: We use the official code of VL-LTR [423] for our experiments. We use a pre-trained CLIP ResNet-50 backbone. We followed the hyperparameters reported in VL-LTR [423]. We augment only "Few" category and train the backbone with the VL-LTR [423] method. Training takes 4 hours on 8 NVIDIA RTX 3090 GPUs.

TABLE A.32. **Long-Tailed ImageNet-LT:** We compare different augmentation methods on ImageNet-LT and report Top-1 accuracy for "Few", "Medium", and "Many" sets. † indicates results with ResNeXt50. ⋆: indicates training with 384 resolution so is not directly comparable with other methods with 224 resolution. On the "Few" set and LiVT method, our augmentations improve the accuracy by 11.7 points compared to LiVT original augmentation and 4.4 points compared to `Txt2Img`.

Method	Many	Med.	Few	Overall Acc
ResNet-50				
CE [100]	64.0	33.8	5.8	41.6
LDAM [52]	60.4	46.9	30.7	49.8
c-RT [223]	61.8	46.2	27.3	49.6
τ-Norm [223]	59.1	46.9	30.7	49.4
Causal [416]	62.7	48.8	31.6	51.8
Logit Adj. [311]	61.1	47.5	27.6	50.1
RIDE(4E)† [464]	68.3	53.5	35.9	56.8
MiSLAS [537]	62.9	50.7	34.3	52.7
DisAlign [528]	61.3	52.2	31.4	52.9
ACE† [50]	71.7	54.6	23.5	56.6
PaCo† [99]	68.0	56.4	37.2	58.2
TADE† [529]	66.5	**57.0**	43.5	58.8
TSC [268]	63.5	49.7	30.4	52.4
GCL [267]	63.0	52.7	37.1	54.5
TLC [260]	68.9	55.7	40.8	55.1
BCL† [544]	67.6	54.6	36.6	57.2
NCL [265]	67.3	55.4	39.0	57.7
SAFA [192]	63.8	49.9	33.4	53.1
DOC [456]	65.1	52.8	34.2	55.0
DLSA [490]	67.8	54.5	38.8	57.5
ResLT [97]	63.3	53.3	40.3	55.1
PaCo [98]	68.2	58.7	41.0	60.0
LWS [222]	62.2	48.6	31.8	51.5
Zero-shot CLIP [352]	60.8	59.3	58.6	59.8
DRO-LT [386]	64.0	49.8	33.1	53.5
VL-LTR [423]	77.8	67.0	50.8	70.1
Cap2Aug [375]	78.5	**67.7**	51.9	70.9
`GeNIe-Ada`	**79.2**	64.6	**59.5**	**71.5**
ViT-B				
LiVT* [493]	76.4	59.7	42.7	63.8
ViT [111]	50.5	23.5	6.9	31.6
MAE [173]	74.7	48.2	19.4	54.5
DeiT [434]	70.4	40.9	12.8	48.4
LiVT [493]	73.6	56.4	41.0	60.9
LiVT + $\texttt{Img2Img}^L$	74.3	56.4	34.3	60.5
LiVT + $\texttt{Img2Img}^H$	73.8	56.4	45.3	61.6
LiVT + `Txt2Img`	**74.9**	55.6	48.3	62.2
LiVT + `GeNIe` (r=0.8)	74.5	56.7	50.9	62.8
LiVT + `GeNIe-Ada`	74.0	**56.9**	**52.7**	**63.1**

A.9.10 More Visualizations

Additional qualitative results resembling the style presented in Fig. 6.4 are presented in Fig. A.25, and more visuals akin to Fig. 6.2 can be found in Fig. A.23. Moreover, we also present more visualization similar to the style in Fig. 6.5 in Fig. A.24.

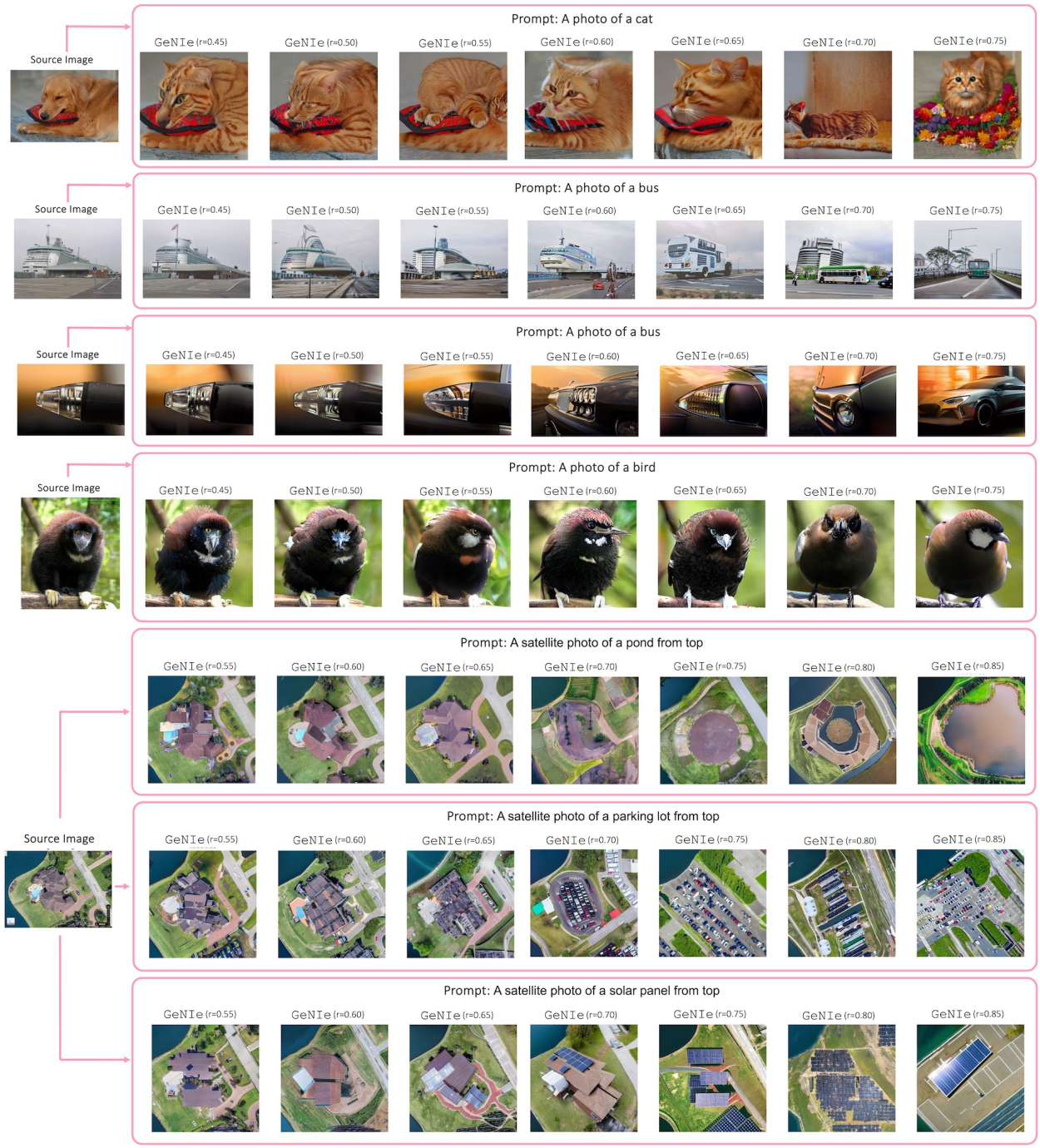


FIGURE A.23. **Effect of noise in GeNIe:** Akin to Fig. 6.2, we use GeNIe to create augmentations with varying noise levels. As is illustrated above, a reduced amount of noise leads to images closely mirroring the semantics of the source images, causing a misalignment with the intended target label.

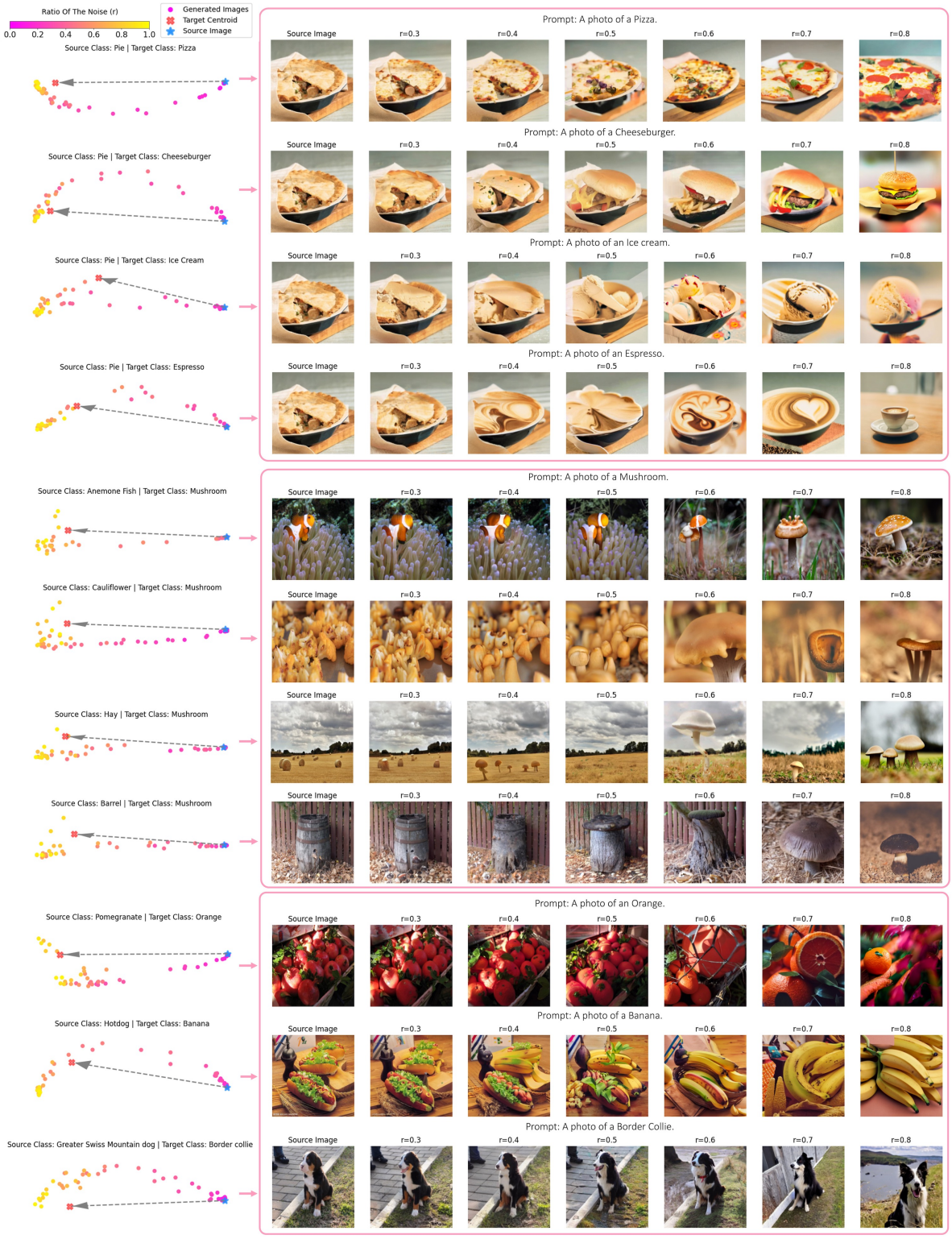


FIGURE A.24. **Effect of noise in GeNIe:** Similar to Fig. 6.5, we pass all the generated augmentations through the DinoV2 ViT-G model, which acts as our oracle model, to obtain their associated embeddings. Subsequently, we employ PCA for visualization purposes. The visualization reveals that the magnitude of semantic transformations is contingent upon both the source image and the specified target category.

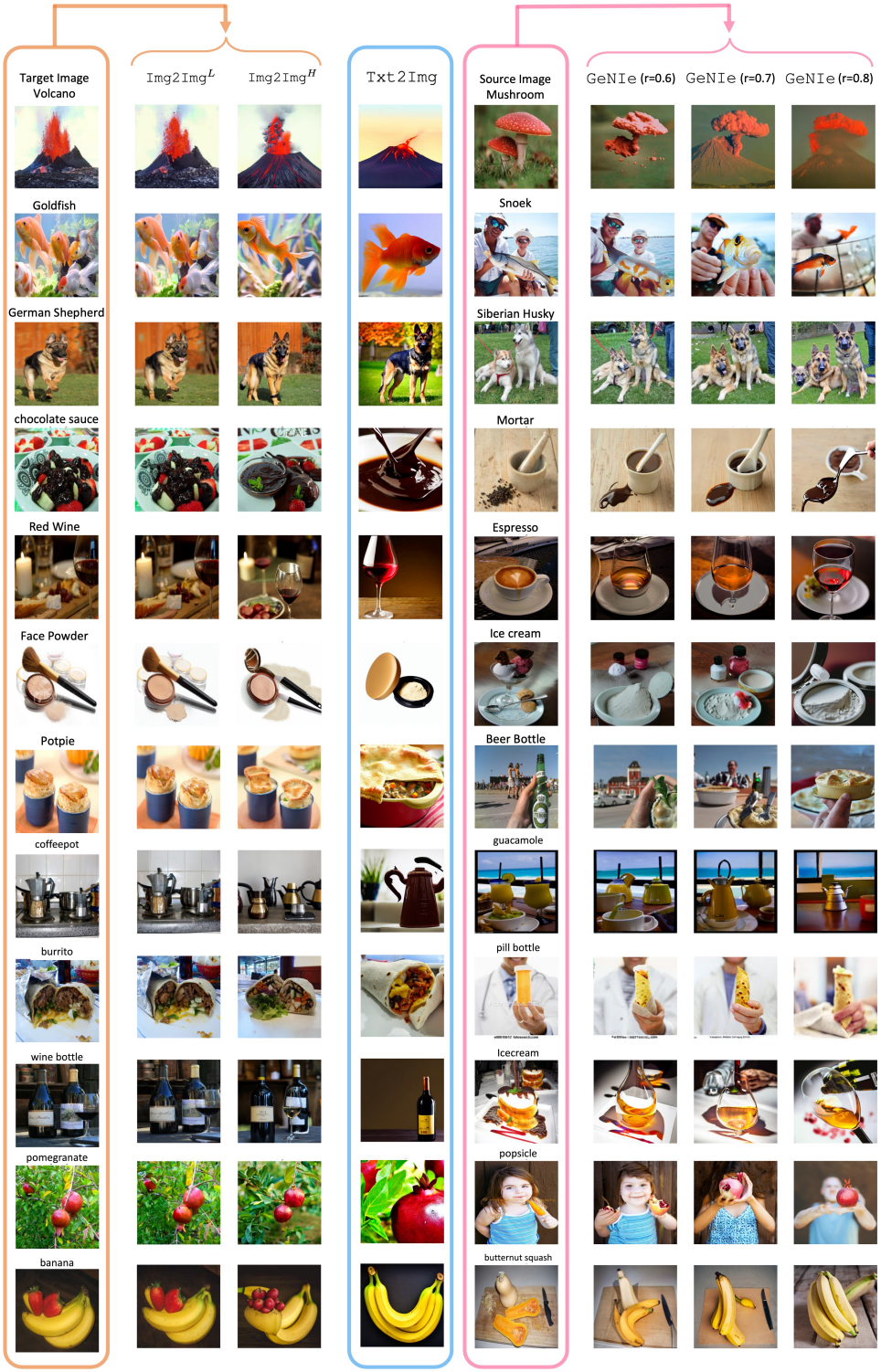


FIGURE A.25. **Visualization of Generative Samples:** More visualization akin to Fig. 6.4. We compare GeNIe with two baselines: **Img2Img^L augmentation** uses both image and text prompt from the same category, resulting in less challenging examples. **Txt2Img augmentation** generates images based solely on a text prompt, potentially deviating from the task's visual domain. **GeNIe augmentation** incorporates the target category name in the text prompt along with the source image, producing desired images with an optimal amount of noise, and balancing the impact of the source image and text prompt.