UC Davis
UC Davis Electronic Theses and Dissertations

Title
Efficiency in Computer Vision: From Compute and Memory to Robustness

Permalink
https://escholarship.org/uc/item/3nw3t486

Author
Kadur Lakshminarasimha Murthy, Navaneet Kadur Lakshminarasimha

Publication Date
2024

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

Efficiency in Computer Vision: From Compute and Memory to Robustness

By

NAVANEET KADUR LAKSHMINARASIMHA MURTHY
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

Chen-Nee Chuah

Rogerio Feris

Committee in Charge

2024

© Navaneet Kadur Lakshminarasimha Murthy, 2024. All rights reserved.

Dedicated to my family

Contents

Abstract

Deep learning has been the biggest success story in the past decade. It is now part of nearly every facet of our lives. Along with growth in popularity, deep learning has also seen growth in terms of model and data sizes and the scale of their training. The introduction of transformer architecture a few years ago has proved to be an inflection point and foundation models have taken off since then. The large vision and language models of today contain hundreds of billions of parameters, are trained on trillions of tokens on thousands of GPUs. They are also being deployed at a scale never seen before, including in real-time and safety critical applications. Very soon, each person could have their own customized LLM model to act as their virtual self.

The big improvements in performance of these models also comes with bigger demands in terms of energy, compute, memory and resources both at the training and inference stages. The general purpose nature of foundation models also implies that they need to be continuously updated on new data. Thus, 'training is a one and done process' cannot be a justification for their huge demands. The training and use of these models also comes with a huge carbon footprint, and can have adverse impact on the environment. Thus, there is both a huge opportunity and a need to develop more efficient models.

There are multiple perspectives to efficiency in deep learning, with compute/energy, speed, memory, data and hardware/resources being the most important ones. Their fates are often intertwined, for instance, smaller models require lesser data and computations and are thus faster and can be run on less expensive hardware. The scale of models can also be a limiting factor in many real-time applications. Making them nimble can unlock a host of new applications.

My goal here is to design and develop such efficient deep learning systems, primarily for computer vision applications, increasing their positive impacts and accessibility. My focus will be on the compute and memory efficiency of models with an eye on their robustness and reliability. I present solutions that reduce the training time and hardware requirements of self-supervised representation learning methods of vision and an easy way to distill them to smaller networks. On the memory front, my work includes a way to effectively fine-tune large vision and language models for a specific downstream task with a tiny fraction of the original model as additional parameters. Ideas from fine-tuning can also be easily adopted in active learning and is part of my future work. Similarly, I also target a recent groundbreaking approach

for novel view synthesis, reducing its memory bottleneck by compressing it. I intend to extend these works to similarly improve the training and inference times of diffusion models for image generation. While most of my research is on making methods more efficient, it is also necessary to stop and analyse their robustness. My work on adversarial attack on efficient transformers opens up new avenues for research to try and develop better attacks and defenses that target the efficiency of these models. I hope that my work contributes to democratizing AI and has a net positive impact on the planet.

Acknowledgments

I would like to thank my advisor Dr. Hamed Pirsiavash in guiding me through this journey. His focus on simplicity helped shape my approach to novel research problems. I have been fortunate to work on interesting and diverse problems throughout my PhD and I thank Hamed for the freedom to pursue my ideas. Even as an academic lab, I have been able to work on ideas that require huge computational resources thanks to his efforts.

Lab members form a huge part of a graduate student's life. I was extremely lucky to have folks that I consider my friends for life and not just lab-mates. I would like to particularly thank Soroush for his support in all my projects. It was a joy to work with him and I will dearly miss our brainstorming sessions. I would also like to thank Ajinkya, Akshay, Kossar, Vipin and Aniruddha who helped me in both lab and life and kept the PhD journey going.

I would like to thank my dissertation and qualifying exam committee members, Chen-nee Chuah, Rogerio Feris, Muhao Chen and Jiawei Zhang. Their feedback has been critical in shaping this document. I would like to thank them for accommodating all the difficult schedules and tight deadlines and help making this a success.

I am forever indebted to my family, especially my parents, brother and sister-in-law. Their help and support in this period has been invaluable. A PhD journey is quite a lonely one and it was exacerbated in my case due to the Covid epidemic. Support from my family kept me going through those difficult periods. Thanks to my baby niece for helping me with the final push to graduate.

Lastly, I would like to thank the Department of Computer Science and UC Davis for their administrative support and for the excellent research environment.

CHAPTER 1

Introduction

Deep learning has grown from being state-of-the-art solution on narrow problems to being ubiquitous in nearly every field in a matter of a few years. Large multi-modal models are poised to take over the world and essentially alter the way humans live. Nearly every problem in computer vision, from image segmentation to 3D reconstruction to motion tracking, now relies on deep neural networks. It is employed in real-time and safety critical applications like autonomous navigation for vehicles, medical imaging and quality control in manufacturing in aerospace and electronics.

Scale is all you need: This surge in popularity and effectiveness of deep learning models is thanks in large part to their scale. The models are ever larger, trained longer on more and more data and dedicated hardware. In what seems like ages ago, AlexNet [170] showed the impact of scale by training one of the earliest deep convolutional networks on ImageNet dataset to achieve a step change in computer vision. This 'ImageNet moment' was then achieved and surpassed in the field of natural language processing with the advent of transformer architecture. In their seminal work [306], Vaswani *et al*. showed that transformers could outperform state-of-the-art recurrent and convolutional neural networks on the task of language translation. The architecture is simple, easy to parallelize and can be scaled to large number of parameters. Building on top of this, the generative pretrained transformer (GPT) series of models by OpenAI showed that scale is the key to unlock the potential of deep networks, resulting in surprising emergent properties [318]. Model sizes have grown from a few million parameters in AlexNet to trillions of parameters [63] in the latest large language models (LLMs). Figure 1.1(left) [211] shows the model parameter count over the years. The growth of model size is considered to be a new Moore's law, with the rate exceeding even that of the growth in semiconductors. LLMs and their multi-modal variants are now named based on their size, with even the smallest of models containing a few billion parameters.

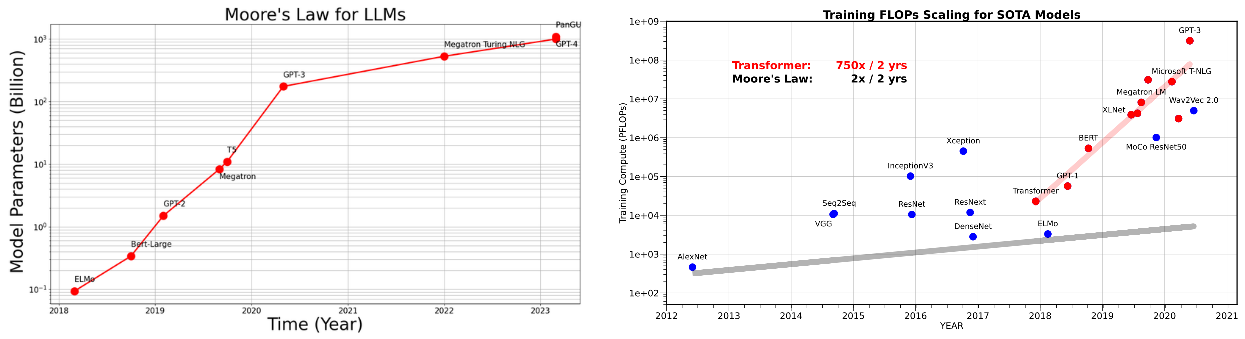


FIGURE 1.1. **Growth in model size (left) and compute (right)**. Both the parameter counts of large language models and the compute required to train them has grown exponentially in the last few years, outpacing the Moore's law prediction for semiconductors! Figures from [211](left) and [96](right).

Despite its impact, model size is just one part of the deep learning success story. Another equally important contributor is the scale of datasets used in the training of these models. ImageNet [65] was one of the first large scale datasets in computer vision with more than a million labelled images. The growth of photo sharing via internet and social media in the recent times has resulted in datasets like YFCC100M [292], Public Multimodal Dataset (PMD) [272], LAION-5B [268] with orders of magnitude higher number of samples. Similarly, large datasets like Common Crawl [1] and RefinedWeb [288], mostly obtained by scraping the web, have been instrumental in the development of LLMs.

Although the size of datasets has grown, it is important to note that these are not carefully curated and can thus be extremely noisy. Many of them, especially image and videos datasets, are also weakly annotated. In such a scenario, self-supervised and semi-supervised learning approaches play a pivotal role in obtaining effective models. Contrastive learning based approaches [106, 119, 228] vastly improved representation learning from unannotated image collections. These approaches relied on a simple idea of mapping similar samples in image space to similar regions in the latent space and to push apart dissimilar images in the latent space. In the absence of supervised labels, similar images can be obtained by augmenting images with operations such as cropping and blurring. Although simple in principle, these methods have been highly effective, nearly matching supervised training performance on ImageNet classification! These techniques have also been adapted to semi-supervised and few-shot learning set-ups, achieving similar improvements. Another line of work uses masked auto-encoding (e.g. MAE [118]) to inpaint images as a way of representation learning. Analogously, LLMs use next word (token) prediction from given context as the primary objective for training. Datasets for this can be easily constructed without extra annotations by masking parts of the text.

While the large scale of models and datasets comes with the obvious benefits, it also has significant drawbacks. Training costs of these models has also been increasing exponentially (refer Figure 1.1 right) and can have far reaching impacts on the environment. For instance, Meta LLaMA 3 [4], a recent open-source LLM from Meta has model variants with around 8, 70 and 400 billion parameters and is trained on 15 trillion tokens. The 400B parameter model training would require more than 10^25 flops. The estimated carbon footprint for training just the two smaller variants is 2290 tonnes of CO2eq. The models were trained on custom-built 24000 GPU clusters adding up to millions of GPU hours. Large vision and language models thus require huge amounts of resources in terms of hardware and electricity to train and can have noticeable impact on the environment. More energy and resources will also need to be dedicated to run inference on these models, especially if they are scaled to the tasks of web search. The huge financial costs required to train and run these models have also limited their accessibility. Unlike a few years ago, most of the state-of-the-art models are now from the industry and not academia. One might need a small GPU cluster just to run inference on some of the larger LLMs while it can take a few seconds on a GPU to generate a single image using diffusion. It is thus extremely important to consider various approaches to make them more efficient.

Efficiency in deep learning: Efficiency in deep learning has five primary aspects - compute, speed, memory, data and resources. They are often tied, and making a model more efficient on one of them usually has a positive effect on one or more of the rest. A distinction is also made between these properties at training and inference stages as they may not always be correlated. Reduction in model size often has the biggest impact on all these and is thus well studied in the literature. Model pruning, knowledge distillation and quantization are some common techniques to reduce model size and thus, compute and memory.

In this thesis, we focus on compute and memory efficient solutions for diverse computer vision tasks. For compute efficiency, we look at knowledge distillation [218] and a more efficient self-supervised learning [219] approach. Both these methods are set in the low/no-labeled data regime. Furthermore, they reduce the GPU memory requirement during training and help democratize AI. This is particularly important since it allowed the training of state-of-the-art representation learning models on consumer grade hardware. Additionally, our work Genie [159] also focuses on few-shot learning settings but with a data perspective, employing an off-the-shelf image generation network that alleviates the need for large amounts of data.

For memory efficiency, we focus on reducing model size. In Compact3D [217], we compress a specific novel view synthesis method to make it nearly 40× smaller and twice faster. In NOLA [157], we propose a new parameter efficient fine-tuning method to fine-tune large language and vision models on task-specific datasets. Our solution can reduce the parameter count of the fine-tuned module by as much as 95% on

LLMs without compromising the performance. This enables running inference with not just one but tens of thousands of LLM variants on a single GPU!

Finally, we also analyse the robustness of a class of compute efficient algorithms. Adversarial attacks [103] have famously shown that deep convolutional networks are susceptible to carefully designed imperceptible perturbations in input images. However, most of the works on adversarial attacks target the model performance on a given task. Here [216], we attack the compute efficiency of adaptive inference networks. These networks adaptively process an input, avoiding the processing of unimportant regions in an image and thus saving compute. Our attack forces them to pay attention to all parts of the image, thus making them inefficient. As the scale of networks and data grow, so will the development of efficient solutions. Our work highlights the need for further studies on efficiency adversarial attacks and design of defenses to make them more robust.

CHAPTER 2

Compute Efficiency in Computer Vision in Training and Inference

Here, we consider the training and inference efficiency of self-supervised representation learning (SSL) methods. For efficient training, we design an approach (Constrained Mean Shift) that needs lesser compute per iteration and also has a faster convergence. The approach can be generalized to work well in settings with vastly diverse availability of labeled data, going from completely self-supervised to fully supervised settings. The method also makes it feasible to train self and semi-supervised approaches with lesser hardware requirements, increasing their accessibility. For efficient inference, we consider knowledge distillation from a large teacher network to a smaller student network. Our work, SimReg, shows that a simple regression based knowledge distillation for self-supervised representation learning is on par with state-of-the-art approaches.

2.1 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

2.1.1 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.

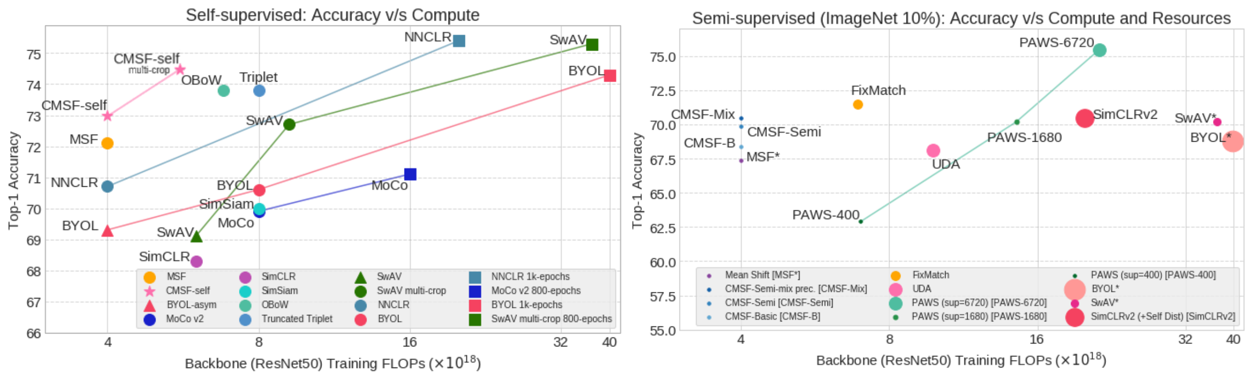


FIGURE 2.1. **Accuracy vs. training compute on ImageNet with ResNet50:** We report the total training FLOPs for forward and backward passes through the CNN backbone. **(Left) Self-supervised:** All methods are for 200 epochs. CMSF$_{\text{self}}$ achieves competitive accuracy with considerably lower 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.

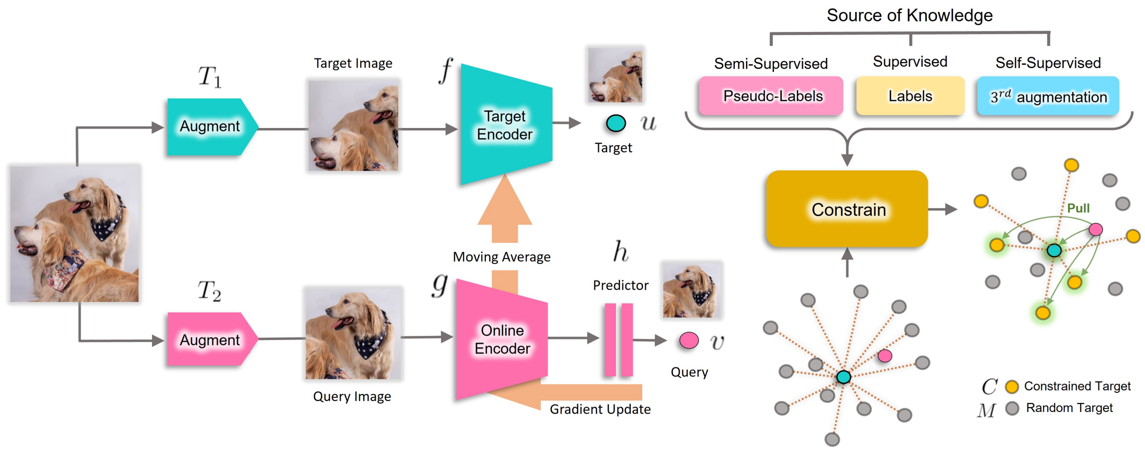


FIGURE 2.2. **Our method (CMSF):** We augment an image twice and pass them through online and target encoders followed by ℓ_2 normalization to get u and v. Mean-shift [162] encourages v to be close to both u and its nearest neighbors (NN). To make NNs diverse, we constrain the NN search space based on 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 2.3 for constructing the constrained set.

Most recent SSL methods, *e.g.*, MoCo [120] and BYOL [107], 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 [177, 252, 296] and the negative set by increasing the set size [120] or mining effective samples [143, 147, 311], but have largely ignored possibility of utilizing additional positive images. More recently, [15, 73, 162] expand the positive set using nearest neighbors. Inspired by classic mean-shift algorithm, MSF [162] 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. 2.4 and Fig. 2.5). Purity is defined as the percentage of the NNs belonging to the same category as the query image.

We generalize MSF [162] 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. 2.1). It outperforms MSF [162] in SSL, cross-entropy in supervised (with clean or noisy labels), and PAWS [14] 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 [162], to utilize additional sources of knowledge to constrain the NN search space.
(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.

2.1.2 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 [352], jigsaw puzzle [224], counting [225], and rotation prediction [98]. Another class of SSL methods is based on instance discrimination [72]. The idea is to classify each image as its own class. Some methods adopt the idea of contrastive learning for instance discrimination [37, 38, 39, 48, 120]. BYOL [107] 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 [360], samples are clustered using k-means and samples within a cluster are brought closer together compared to cross-cluster samples. MSF [162] and MYOW [15] generalize BYOL by regressing target view and its NNs. NNCLR [73] extends SimCLR to use NNs as positives. CLD [315] integrates grouping using instance-group discrimination. Affinity diffusion [141] 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 [162] to use an additional source of knowledge to constrain the NN search space for the target view. CoCLR [113] and Cl-InfoNCE [302] 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 [278, 354]. [214, 282, 301, 334] address the issue of hard labeling, *e.g.*, (one-hot labels) with label smoothing, [18, 91, 130] replace hard labels with prediction of pre-trained teacher, and [344, 349] propose an augmentation strategy to train on combination of instances and their labels. Another line of work [100, 263] is to learn representations with good kNN performance. SupCon [152] and [326] improve upon [100] 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 [348] uses rotation prediction based loss on the unlabeled set along with cross-entropy loss on the labeled set. Similarly, SuNCEt [13] combines SimCLR [48] and SwAV [38] methods with supervised contrastive loss. Pseudo-labeling is frequently used in semi-supervised learning. In Pseudo-Label [174], 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 [49], a teacher network is pre-trained using SimCLR [48] and fine-tuned with supervised labels. The teacher is then distilled to a student network using pseudo-labels on the unlabeled set. FixMatch [276] uses pseudo-labels obtained using a weakly augmented image to train a strongly augmented version of the same image. UDA [330] leverages strong data augmentation techniques in enforcing this consistency in pseudo-labels across augmentations. MPL [241] 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 [14] 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 [162] 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 [58, 267, 320] and contrastive loss [30, 275]. Metric learning methods perform well on tasks like image retrieval [324] and few-shot learning [274, 309]. Prototypical networks [274] is similar to a contrastive version of our method with top-*all*.

2.1.3 Method

Similar to MSF [162], 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

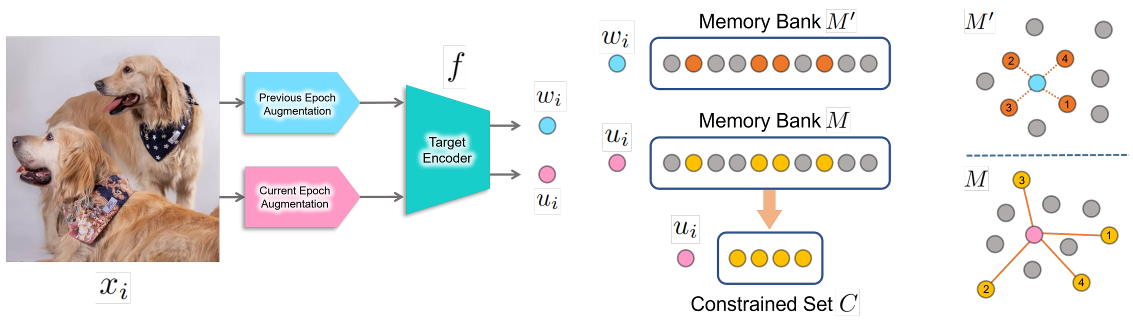


FIGURE 2.3. **CMSF$_{\text{self}}$:** The indices of the NNs of the previous epoch's memory bank M' are used to construct the constrained set C from the current memory bank M.

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 $h(.)$ [107] 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 2.1.3.1, 2.1.3.2, and 2.1.3.3. 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 [107] when $k = 1$ and to self-supervised mean-shift [162] 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.

2.1.3.1 Self-Supervised Setting

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. 2.3). 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 2.1-Right and Fig. 2.5).

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 2.1-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.

2.1.3.2 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. 2.6), 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.

2.1.3.3 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.

2.1.4 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 [121] 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. While we focus on single crop setting in most of our experiments, we also report the results for multiple crop inputs in the SSL setting. Following SwAV [38], we use four additional crops of 96x96 resolution as input. These are used as inputs only to the online encoder and not the target encoder. The momentum value of CMSF for the moving average key encoder is 0.99. The 2-layer MLP architecture for 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

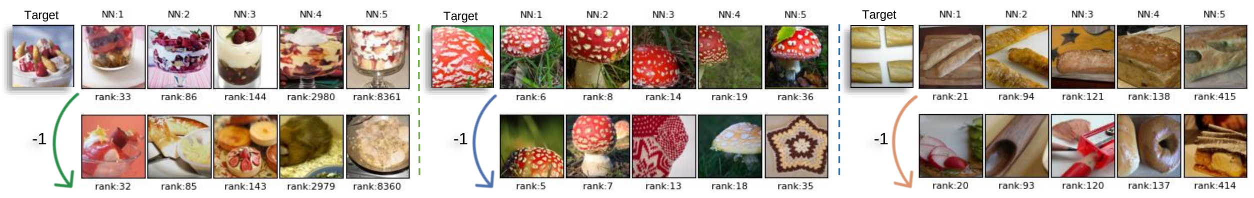


FIGURE 2.4. **Nearest neighbor selection on constrained memory bank:** First row shows top-5 NNs of target in constrained set C and their corresponding rank in the unconstrained memory bank M obtained using an intermediate checkpoint (epoch 100). While they are not the closest samples to the target (higher 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.

forward and backward passes through the network (that is also done in BYOL): the increase in time is 0.7% for MSF [162] and 2.1% for 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 [160]. Additionally, we report k-nearest neighbor ($k = 1, 20$) evaluation for the SSL setting as in [160]. The transfer performance is evaluated on the following datasets: Food101 [28], SUN397 [327], CIFAR10 [166], CIFAR100 [166], Cars196 [163], Aircraft [203], Flowers (Flwrs102) [221], Pets [233], Caltech-101 (Calt101) [83], and DTD [59] (additional details in supplementary material.)

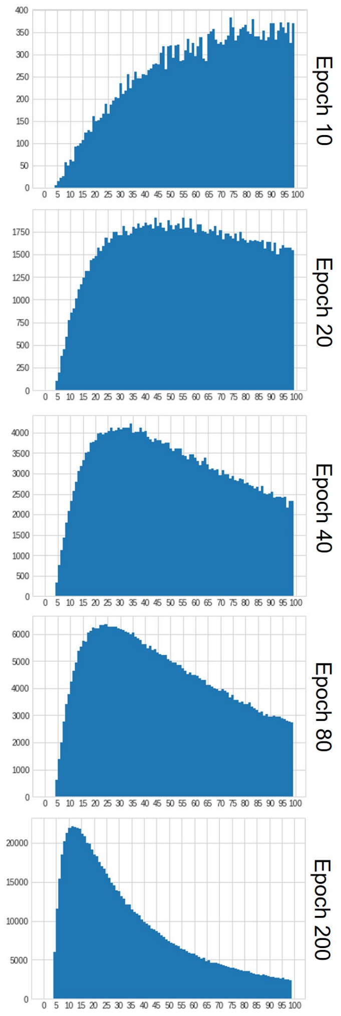
2.1.4.1 Self-Supervised Learning (CMSF$_{\text{self}}$)

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.

Results on ImageNet: Results of CMSF$_{\text{self}}$ are shown in Table 2.1. CMSF$_{\text{self}}$ outperforms MSF baseline with a larger memory bank, which we believe is due to pulling together far yet semantically similar

TABLE 2.1. **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 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 5th 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 CMSF$_{\text{self}}$. Also, the median of these ranks are shown in Figure 2.5. 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	[9]	256	100	-	-	76.2	71.4	74.8
Random-init	-	-	-	-	-	5.1	1.5	2.0
SeLa-v2 [339]	[38]	4096	400	✓	✗	67.2	-	-
SimCLR [48]	[48]	4096	1000	✓	✗	69.3	-	-
SwAV [38]	[38]	4096	400	✓	✗	70.1	-	-
DeepCluster-v2 [37]	[38]	4096	400	✓	✗	70.2	-	-
SimSiam [53]	[53]	256	400	✓	✗	70.8	-	-
MoCo v2 [120]	[51]	256	800	✗	✗	71.1	57.3	61.0
CompRess [160]	[160]	256	1K+130	✗	✗	71.9	63.3	66.8
InvP	[310]	256	800	✗	✗	71.3	-	-
BYOL [107]	[107]	4096	1000	✓	✗	74.3	62.8	66.9
SwAV [38]	[38]	4096	800	✓	✓	75.3	-	-
NNCLR	[73]	4096	1000	✗	✗	**75.4**	-	-
SimCLR [48]	[53]	4096	200	✓	✗	68.3	-	-
SwAV [38]	[53]	4096	200	✓	✗	69.1	-	-
MoCo v2 [120]	[53]	256	200	✓	✗	69.9	-	-
SimSiam [53]	[53]	256	200	✓	✗	70.0	-	-
NNCLR [73]	[73]	4096	200	✗	✗	70.7	-	-
BYOL [107]	[53]	4096	200	✓	✗	70.6	-	-
SwAV [38]	[53]	256	200	✓	✓	72.7	-	-
Truncated Triplet [311]	[311]	832	200	✓	✗	73.8	-	-
OBoW [97]	[97]	256	200	✗	✓	73.8	-	-
CMSF$_{\text{self}}$ (128K)	-	256	200	✗	✓	**74.4**	**62.3**	**66.2**
MoCo v2 [120]	[51]	256	200	✗	✗	67.5	50.9	54.3
CO2 [317]	[317]	256	200	✗	✗	68.0	-	-
BYOL-asym [107]	[162]	256	200	✗	✗	69.3	55.0	59.2
ISD [290]	[290]	256	200	✗	✗	69.8	59.2	62.0
MSF (1M) [162]	[162]	256	200	✗	✗	72.4	62.0	64.9
MSF (256K) [162]	[162]	256	200	✗	✗	72.2	62.1	65.1
CMSF$_{\text{self}}$ (128K)	-	256	200	✗	✗	**73.0**	**63.2**	**66.4**



samples (Fig. 2.4). We use MSF with 2x larger memory bank for fair comparison. 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. 2.1. We observe a good trade-off in terms of accuracy and compute for CMSF$_{\text{self}}$. Our best performance is obtained with the multi-crop version but at the cost of increased compute.

Evaluation on ImageNet subsets: Following [48,125], we evaluate the pre-trained models on the ImageNet

TABLE 2.2. **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 [162], SimCLR and Xent (1000 epoch) from [48], and BYOL from [107].

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$_{\text{sup}}$ 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$_{\text{sup}}$ 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
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**

classification task with limited labels. We report results with 1% and 10% labeled subsets of ImageNet (Table 2.4). 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.

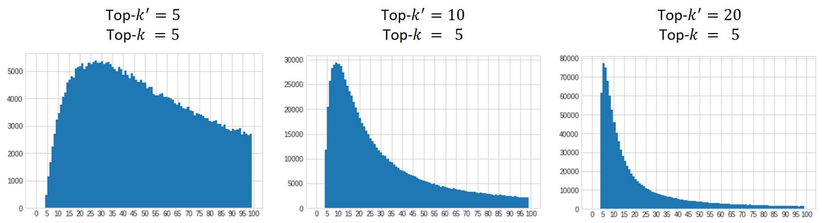
Transfer learning: We follow the procedure in [48, 107] for transfer evaluation (refer to Table 2.2). Hyperparameters for each dataset are tuned independently based on the validation set accuracy and final accuracy is reported on the held-out test set (more details in supplementary). CMSF$_{\text{self}}$ achieves SOTA average performance among methods trained for 200 epochs.

Purity of constrained samples: In 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 2.1-Right and Fig. 2.5). Also, as shown in Fig. 2.5, 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.

Effect of k': In 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 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.

TABLE 2.3. **Effect of k' in sampling NN from M':** In 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. The histogram shifts left, *i.e.*, overlap between constrained and unconstrained NN sets increases with increasing value of k'.

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



As observed in Table 2.3-right, the overlap between constrained and unconstrained NN set increases with increasing value of k'. Note that in a case where $k' = \infty$, CMSF$_{\text{self}}$ will be identical to the MSF baseline.

2.1.4.2 Supervised Learning

Evaluation: Unlike cross-entropy (Xent [22,180,258]) baseline, SupCon [152], ProtoNW [274] 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.

Results: Results on IN-1k dataset are shown in Table 2.2. 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$_{\text{sup}}$ 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 [293] (Fig. 2.6). We observe a significantly higher degradation in performance of Xent baseline and CMSF$_{\text{sup}}$ top-*all* compared to CMSF$_{\text{sup}}$ 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

TABLE 2.5. **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**

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 2.5 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.

2.1.4.3 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 [348], we perform two stages of fine-tuning with supervised and pseudo-labels. We fine-tune the backbone network with two MLPs (as in PAWS [14]) on the 10% labeled set for 20 epochs and pseudo-label the train set. Samples above confidence threshold (nearly 30% of dataset) are

TABLE 2.6. **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. 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 [62]. * TPUs are used.

Method	Epochs	Batch Size	GPUs	FLOPs (x10^{18})	Top-1
Self-supervised Pre-training					
Mean Shift [162]	200	256	4	4	67.4
BYOL [107]	1000	4096	512*	40	68.8
SwAV [38]	800	4096	64	37	70.2
SimCLRv2 [49]	800	4096	128*	16	68.4
Semi-supervised Pre-training					
SimCLRv2 (+Self Dist) [49]	1200	4096	128*	20	70.5
UDA† [330]	800	15872	64*	10	68.1
FixMatch† [276]	300	6144	32*	7	71.5
MPL† [241]	800	2048	-	30	73.9
PAWS (support=6720) [14]	300	4096	64	21	75.5
PAWS (support=1680) [14]	100	256	8	15	70.2
PAWS (support=400) [14]	100	256	4	7	62.9
CMSF$_{\text{semi}}$-basic	200	256	4	4	68.6
CMSF$_{\text{semi}}$	200	256	4	4	69.9
CMSF$_{\text{semi}}$-mix precision	200	768	4	4	70.5

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 [241, 276, 330].

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 [14] with a higher batch size of 768 since it has lower memory requirement.

Results: 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 [62] (Table 2.6, Fig. 2.1).

PAWS with a support set size of 6720 outperforms other approaches. However, this requires significantly higher compute (4.8× FLOPs) and resources (64 GPUs) compared to 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.

2.1.5 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.

2.2 SimReg: Regression as a Simple Yet Effective Tool for Self-supervised Knowledge Distillation

Feature regression is a simple way to distill large neural network models to smaller ones. We show that with simple changes to the network architecture, regression can outperform more complex state-of-the-art approaches for knowledge distillation from self-supervised models. Surprisingly, the addition of a multi-layer perceptron head to the CNN backbone is beneficial even if used only during distillation and discarded in the downstream task. Deeper non-linear projections can thus be used to accurately mimic the teacher without changing inference architecture and time. Moreover, we utilize independent projection heads to simultaneously distill multiple teacher networks. We also find that using the same weakly augmented image as input for both teacher and student networks aids distillation. Experiments on ImageNet dataset demonstrate the efficacy of the proposed changes in various self-supervised distillation settings.

2.2.1 Introduction

There has been a tremendous improvement in deep learning methodologies and architectures in the last few years. While this has lead to significant improvements in performance on various computer vision tasks, it has also resulted in complex and deep networks that require high compute during inference [121, 140, 142, 346]. Various specialized architectures [134, 144, 265, 285] have been proposed to minimize the inference time and memory requirements of the model to be deployed. Knowledge distillation [34, 130] has been proposed as an effective technique to compress information from larger but effective models (teachers) to lighter ones (students).

With availability of large scale unlabeled datasets, self-supervised learning (SSL) has received great attention in recent times. Several SSL methods achieve close to supervised performance on the benchmark ImageNet object classification task [37, 107]. Unlike supervised models, the outputs of a self-supervised network are latent feature vectors and not class probabilities. An additional module is generally trained atop the pretrained SSL models using supervision to perform the downstream task. Conventional knowledge distillation methods proposed for supervised classification are thus not applicable for distillation from self-supervised networks. A simple way to handle this is to directly regress the teacher latent features. Recent works [79, 161] have proposed more complex solutions that try to capture the structure of the teacher latent space and are shown to outperform the regression baselines.

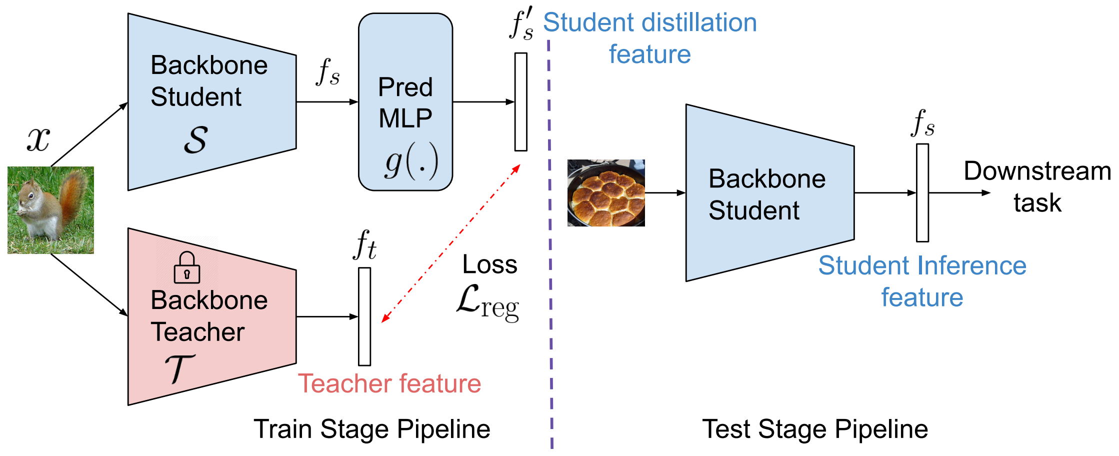

x
Backbone Student $\mathcal{S}$
f_s
Pred MLP $g(.)$
f_s' Student distillation feature
Backbone Teacher $\mathcal{T}$
f_t
Loss $\mathcal{L}_{\text{reg}}$
Teacher feature
Train Stage Pipeline
Backbone Student
f_s
Downstream task
Student Inference feature
Test Stage Pipeline

FIGURE 2.7. **Proposed distillation pipeline:** We propose a simple modification of using a MLP prediction module during distillation. The module is discarded during inference. Surprisingly, performance of backbone features f_s is better than those from MLP output, f_s', though f_s' more closely matches the teacher. A deeper MLP helps improve distillation performance.

Use of a multi-layer perceptron (MLP) head atop CNN backbone model has been shown to help self-supervised models prevent overfitting to the SSL task and generalize better to downstream applications [48, 52, 54, 107]. Such modules are used only during SSL pre-training and are not part of inference network. In this work, we consider the task of distilling self-supervised models. We employ a similar *prediction head* atop the student backbone network to effectively mimic the teacher. As in SSL, the prediction module is discarded after distillation and thus, there is no change in the time and memory required during inference (refer Fig. 2.7). We empirically demonstrate that doing so does not hurt classification performance. Counter-intuitively, we observe that the features from the backbone network outperform those from the final layer of the prediction head though the final layer best matches the teacher. Unlike in SSL, overfitting to the training task (i.e, exactly mimicking the teacher) benefits distillation [25] and it is not clear why generalization could be better at intermediate layers where the similarity with teacher is reduced. Our finding suggests that we require a deeper analysis to understand how well the student models mimic the teacher in general and how knowledge distillation works. Crucially, it also enables us to use a deeper prediction head to achieve lower train and test error leading to better downstream performance without increasing the student capacity.

We empirically show that the above observation generalises to distillation with different teacher and student settings and to other self-supervised distillation technique. Our simple regression model with a MLP prediction head outperforms complex state-of-the-art approaches that require the use of memory banks and tuning of temperature parameter. Our work serves as an important benchmark for future self-supervised distillation works. The use of MLP heads also facilitates effective distillation from multiple SSL teacher networks. Additionally, we demonstrate that using the same augmented image with weak augmentation for both student and teacher networks results in better student models. Since aggressive augmentation is necessary for effective self-supervised learning [48, 52] but hurts their ability to generalize [243], our approach could be used to learn better SSL models. To summarize, our contributions are simple changes to architecture and augmentation strategy of distillation networks that not only achieve state-of-the-art performance on SSL model distillation but also question our current understanding of knowledge distillation.

2.2.2 Related Works

Supervised knowledge distillation: Bucilua *et al.* [34] and Hinton *et al.* [130] pioneered the use of knowledge distillation for compressing information. The methods used the teacher prediction logits as soft-labels in addition to the supervised label to regularize the student model. [234] minimizes the divergence between the student and teacher probability distributions. Several works ([128, 155, 257]) utilize intermediate teacher outputs in distillation. FitNets [257] match both the final and intermediate teacher representations while [155] transfers knowledge from the attention maps of the teacher. RKD [231] transfers mutual relations instead of instance wise distillation. [312] proposes directly regressing the final teacher features with a modified loss function that strictly matches the direction of the features but allows flexibility in terms of feature magnitude.

Self-supervised representation learning (SSL): Earlier works on SSL ([37, 69, 98, 224, 225, 237]) learn effective representations by solving pretext tasks that do not require supervised labels. Recently, works based on contrastive learning ([17, 38, 48, 52, 120, 209, 294]) have gained focus. In contrastive learning, the distances between representations of positive pairs are minimized while those between negative pairs are maximized. The positive and negative pairs are generally constructed by utilizing multiple augmentations of each image. BYOL [107] is closer to knowledge distillation, where the distance between teacher and student representations are minimized. The inputs to the two networks must be different augmentations of the same image and the teacher network is obtained as a moving average of the student. Similar to our work, [107] employs MLP head atop the student network to predict the teacher features.

Distillation of self-supervised models: In [226], the student mimics the unsupervised cluster labels predicted by the teacher. CRD [295] maximizes a lower bound of the mutual information between the teacher and student networks. However, it additionally uses supervised loss for optimization. CompRess [161] and SEED [79] are specifically designed for compressing self-supervised models. In both these works, student mimics the relative distances of teacher over a set of anchor points. Thus, they require maintaining large memory banks of anchor features and tuning temperature parameters. As in regression, proposed prediction heads can also be used to improve CompRess and SEED.

2.2.3 Knowledge Distillation

We first consider the supervised model distillation formulation proposed in [130]. The teacher is trained on the task of object classification from images. Let X be the set of images, Y the set of corresponding class labels and c the total number of classes. Consider a teacher network with $f_t = T(x)$, $f_t \in \mathbb{R}^c$ as the output vector (logits) corresponding to input image x. The predicted class probabilities can be obtained by applying softmax operation $\sigma(.)$ atop the vector f_t.

$$\hat{y}_t = \sigma(f_t; \tau_t) = \frac{e^{f_t/\tau_t}}{\sum_i e^{f_t^i/\tau_t}} \tag{2.1}$$

where f_t^i is the i^{th} dimensional output of the feature vector and τ_t is the temperature parameter. The teacher network is trained using the image-label pairs in a supervised fashion with standard cross-entropy loss. The trained teacher network is to be distilled to a student network. Once trained, the teacher network parameters are frozen during the distillation process. Let be the student network, $f_s = S(x)$, $f_s \in \mathbb{R}^c$ the feature vector corresponding to input image x and $\hat{y}_s = \sigma(f_s; \tau_s)$ the predicted class probability vector. Knowledge distillation loss is given by

$$L_{\text{KD}}(\hat{y}_t, \hat{y}_s) = \sum_{j=1}^{c} \hat{y}_t^j \log(\hat{y}_s^j) \tag{2.2}$$

The student is trained using a combined objective function involving supervised cross entropy loss on student features L_{CE} and distillation loss L_{KD}:

$$L = \lambda L_{\text{CE}} + (1 - \lambda)\tau_s^2 L_{\text{KD}} \tag{2.3}$$

where λ is a hyperparameter that determines the relative importance of each loss term. Since τ_t is generally set to 1, KD loss is multiplied by a factor of τ_s^2 to match the scale of gradients from both loss terms.

2.2.3.1 Distillation of Self-supervised Models

The value of λ in Eq. 2.3 can be set to 0 if the class labels are not available during student distillation. However, the formulation cannot be directly employed to distill from self-supervised teacher networks since the teacher outputs are latent representations and not logits or class probability vectors. Thus, to distill from such teachers, we simply regress the final feature vector of the teacher. Let $f_t = T(x)$, $f_t \in \mathbb{R}^d$ and $f_s = S(x)$, $f_s \in \mathbb{R}^m$. Since it is not necessary for the student and teacher representation dimensions to be the same, we use a linear projection of the student feature to match the dimensions.

$$f_s' = W^T f_s + b; \;\; W \in \mathbb{R}^m \times \mathbb{R}^d, \, b \in \mathbb{R}^d \tag{2.4}$$

The distillation objective is then given by $L = L_{\text{reg}} = \mathrm{d}(f_t, f_s')$ where $\mathrm{d}(.)$ is a distance metric. Here, we consider squared Euclidean distance of l_2 normalized features as the metric.

2.2.3.2 Prediction Heads for Regression Based Distillation

For a more effective matching of the teacher latent space, we propose a non-linear prediction head $g(.)$ atop the student backbone network in place of the linear projection in Eq. 2.4. During training, the student feature is then obtained as $f_s' = g(f_s)$ where $g(.)$ is modeled using a multi-layer perceptron (MLP). Each layer in $g(.)$ is given by a linear layer with bias followed by batch-normalization and a non-linear activation function (we use ReLU non-linearity in all our experiments). The number of such layers is a hyperparameter to be optimized. The dimension of the last layer output matches that of the teacher. There is no non-linearity in the final layer to prevent constraining the output space of the student network. During inference, the prediction head $g(.)$ is removed and the output of the student network is obtained as $f_s = S(x)$ (refer Fig. 2.7). Thus, there is no change in the architecture or the number of parameters of the model to be deployed. In our experiments, we demonstrate that the use of such MLP heads plays a crucial role in improving downstream performance. Surprisingly, we also observe that preserving the prediction heads during inference is not necessarily beneficial and might result in reduction in performance.

2.2.3.3 Multi-teacher Distillation

The prediction heads are particularly beneficial in distillation from multiple teacher networks. Independent deep non-linear projections of the student backbone features can be employed during distillation to match each of the teachers. Let f_t^k be the output vector of the k^{th} teacher and $f_s^k = g^k(f_s)$ that of the corresponding student prediction head $g^k(.)$. The multi-teacher distillation objective for K teachers is given by:

$$L = \frac{1}{K}\sum_k \mathrm{d}(f_t^k, f_s^k) \tag{2.5}$$

The prediction heads are trained by the loss term from corresponding teachers while the backbone S is trained using the summation in Eq. 2.5.

2.2.4 Experiments

We consider distillation of pretrained self-supervised models. We consider four such methods for teacher networks - MoCo-v2 [52], BYOL [107], SwAV [37] and SimCLR [48]. We use the official publicly released models for all the teacher networks (details in suppl.). We also use a ResNet-50 model trained with supervised labels (provided by PyTorch in [7]) as a teacher. All teacher training and student distillation is performed on the train set of ImageNet. We consider different teacher and student backbone network architectures. For the prediction head, we experiment with linear, 2 and 4 layer MLPs. Let the dimension of the student backbone output be m and that of teacher d. Similar to the prediction head in [107], the MLP dimensions are $(m, 2m, m, 2m, d)$.

Implementation details: We use SGD optimizer with cosine scheduling of learning rate and momentum of 0.9. Initial learning rate is set to 0.05. As in [161] the networks are trained for 130 epochs with batch size of 256. Cached teacher features are utilized for faster distillation in experiments with SimCLR, BYOL and SwAV teachers. We publicly release the code*.

Datasets: We primarily evaluate the performance of distilled networks on ImageNet [259] classification task. Additionally, for transfer performance evaluation, we consider the following datasets: Food101 [29], CIFAR10 [167], CIFAR100 [167], SUN397 [328], Cars [164], Aircraft [204], DTD [60], Pets [232], Caltech-101 [84] and Flowers [222]. We train a single linear layer atop the frozen backbone network for transfer evaluation.

Metrics: We use k-nearest neighbour (k-NN) and linear evaluation on all tasks. We also report mean squared error (MSE) between the student and teacher features over the test set. For k-NN evaluation, k=1 and 20 are considered and cosine similarity is used to calculate NNs. We employ FAISS [3] GPU library to

*Code is available at `https://github.com/UCDvision/simreg`

TABLE 2.7. **Role of MLP Heads.** We train three models with varying number of layers in prediction head and use the features from *final* MLP layer of each prediction model for evaluation. Deeper models more closely match the teacher (lower MSE) and achieve better classification performance (1 and 20 Nearest Neighbour and Linear evaluation).

Train and Inference Arch	1-NN	20-NN	Linear	MSE
MobileNet-v2+4L-MLP	**54.5**	**58.7**	**68.5**	**0.090**
MobileNet-v2+2L-MLP	54.0	58.0	67.9	0.097
MobileNet-v2+Linear	50.8	55.1	58.3	0.149

TABLE 2.8. **Effect of MLP Heads on inference.** We train a single model with 4 layer MLP head and perform evaluation using features from different layers (pre-MLP, intermediate MLP layer and MLP output). Since the final layer outputs are trained to mimic the teacher, MSE with teacher features is lowest at 4L-MLP while that at 2L-MLP is extremely high (dimension of BB and teacher are different, hence MSE is not reported). However, features from backbone and intermediate layer (+2L-MLP) outperform those from final layer (+4L-MLP) on classification, contrary to the notion that features with lower MSE generalize better.

Train	Backbone(BB)+4L-MLP								
Inference	Backbone(BB)			BB+2L-MLP			BB+4L-MLP		
Metric	1-NN	Linear	MSE	1-NN	Linear	MSE	1-NN	Linear	MSE
ResNet-18	55.3	65.7	-	**56.0**	**66.4**	1.99	53.4	65.2	**0.1**

perform fast k-NN evaluation. For linear evaluation, a single linear layer is trained atop the features from the network to be evaluated. As in [161], the inputs to the linear layer are normalized to unit l_2 norm and then each dimension is shifted and scaled to have unit mean and zero variance. The layer is trained for 40 epochs using SGD with learning rate of 0.01 and momentum of 0.9.

2.2.4.1 Baseline Approaches

Regression: In addition to proposed MLP prediction head based regression (termed *SimReg-MLP*), we consider two additional regression baseline methods proposed in [161] termed 'Regress' and 'Regress-BN'. While Regress distills from unnormalized teacher features, Regress-BN uses batch-norm layer atop the final student and teacher features during distillation. Unlike Regress-MLP, both these approaches use a linear prediction head.

CompRess: CompRess [161] is designed to distill specifically from self-supervised models. Given a set of anchor points, the student is encouraged to have the same similarities with the anchors as that of the teacher. The anchor point features can either be common features from a teacher memory bank (CompRess-1q) or features from individual memory banks for teacher and student (CompRess-2q). We additionally implement CompRess with our MLP prediction head, termed CompRess-1q-MLP and CompRess-2q-MLP.

TABLE 2.9. **Comparison of SSL distillation methods on ImageNet classification.** Our regression method with MLP head (SimReg-4L-MLP) is comparable to or better than the complex state-of-the-art approaches, especially on the linear evaluation metric. We also observe that CompRess-1q and 2q are improved when MLP heads are utilized. Interestingly, regression gets a significantly higher boost compared to CompRess upon addition of MLP layers. Note that the MLPs are used only during training and the inference network architecture remains the same for all approaches making the comparison fair. * metrics from CompRess [161].

Teacher	MoCo-v2 ResNet-50		MoCo-v2 ResNet-50		SimCLR ResNet-50x4	
Student	MobileNet-v2		ResNet-18		ResNet-50	
Method	1-NN	Linear	1-NN	Linear	1-NN	Linear
Teacher*	57.3	70.8	57.3	70.8	64.5	75.6
Supervised*	64.9	71.9	63.0	69.8	71.4	76.2
Regress*	38.6	48.0	41.7	52.2	-	-
Regress-BN*	48.7	62.3	47.3	58.2	-	-
CC [226]*	50.2	59.2	51.1	61.1	55.6	68.9
CRD [295]*	36.0	54.1	43.7	58.4	-	-
CompRess-2q [161]*	54.4	63.0	53.4	61.7	63.0	71.0
CompRess-1q [161]*	54.8	65.8	53.5	62.6	63.3	71.9
CompRess-2q-4L-MLP	**56.3**	67.4	54.4	64.0	**62.5**	73.5
CompRess-1q-4L-MLP	55.5	67.1	**54.9**	64.6	60.9	72.9
SimReg-4L-MLP	55.5	**69.1**	54.8	**65.1**	60.3	**74.2**

SEED [79] proposes similarity based distillation similar to [161] but uses pre-trained teacher models with significantly lower number of training epochs and performance. Further, it requires access to the projection heads used atop teacher networks used only during SSL training and not inference, These parameters are generally not publicly released, making the setting less replicable. Thus we provide comparisons with only CompRess [161].

Contrastive Representation Distillation (CRD): CRD [295] uses a contrastive formulation to bring corresponding teacher and student features closer while pushing apart those from unrelated pairs. While the paper considered a supervised setup and loss term utilizing labels, we use the formulation with just the contrastive loss as proposed in [161].

Cluster Classification (CC): In CC [226], the student predicts unsupervised labels obtained by clustering samples using teacher features. We report metrics for CC and CRD from [161].

2.2.5 Results

Role of Prediction Head: A deeper prediction head results in a student with higher representational capacity and thus a model that better matches the teacher representations. Table 2.7 shows results for models with a common MobileNet-v2 [265] backbone and different prediction head architectures. The prediction head is used during both student training and evaluation. We observe that a deeper model has lower MSE with teacher features and better classification performance. However, a deeper model also implies greater inference time and memory requirements. The student architecture is fixed based on deployment needs and thus requirement of larger model goes against the very essence of distillation. To analyze performance at different layers of the prediction head, we train a single ResNet-18 [121] student with all intermediate dimensions of MLP equal to that of the output. Surprisingly, a model trained with MLP prediction head performs well on downstream task even when the prediction head is discarded during inference (Table 2.8). The performance using features from backbone network is slightly better than that from the final layer outputs whenever a MLP head is used (more results in suppl.). More importantly, this observation enables us to use deeper prediction heads for distillation in place of linear layers without any concerns about altering the student architecture or increasing inference time.

Comparison with existing approaches: In all the remaining experiments, we use SimReg-4L-MLP with the prediction head used only during distillation. We compare the proposed regression method with other baselines and self-supervised distillation methods in tables 2.9 and 2.10. Surprisingly, our simple regression performs comparably or even outperforms the state-of-the-approaches on all settings and metrics. On linear evaluation, we outperform previous methods (without MLP) by **3.3**, **2.5** and **2.3** points respectively on MobileNet-v2, ResNet-18 and ResNet-50 students. Our observation generalizes to similarity based distillation too. Use of MLP prediction head also consistently improves the classification performance of both the CompRess variants (table 2.9). Note that the linear metrics of our student model on MobileNet-v2 and ResNet-50 are just 1.7 and 1.4 points below the corresponding teacher accuracies. Our ResNet-50 model distilled from SimCLR teacher outperforms a ResNet-50 model trained from scratch using SimCLR (69.3% [48]) by **4.9** points.

Transfer Learning: Since an important goal of self-supervised learning is to learn models that generalize well to new datasets and tasks, we evaluate the transfer learning performance of our distilled networks. The results in table 2.11 suggest that the regression model transfers as well as or better than the state-of-the-approaches on most datasets. Among CompRess variants, CompRess-2q-MLP is generally better on ImageNet classification (table 2.9) but transfers poorly (table 2.11) compared to CompRess-1q-MLP. However, the same SimReg model performs comparably or outperforms them both in ImageNet and transfer tasks.

TABLE 2.10. **ImageNet Evaluation with different teacher networks.** We distill from two pretrained ResNet-50 SSL models, BYOL and SwAV to ResNet-18 students. When distilled from these stronger teacher networks, SimReg is significantly better than both CompRess variants on all metrics. Both SimReg and CompRess contain MLP head only during training.

Teacher	BYOL ResNet-50			SwAV ResNet-50		
Method	1-NN	20-NN	Linear	1-NN	20-NN	Linear
Teacher	62.8	66.8	74.3	60.7	64.8	75.6
CompRess-2q-4L-MLP	56.0	60.6	65.2	53.2	58.1	63.9
CompRess-1q-4L-MLP	55.4	60.0	65.2	52.4	57.1	63.4
SimReg-4L-MLP	**56.7**	**61.6**	**66.8**	**54.0**	**59.3**	**65.8**

TABLE 2.11. **Transfer learning results on multiple classification tasks.** Since the teacher networks are self-supervised, generalization of learnt features to other datasets is important. SimReg is significantly better than CompRess-2q and comparable to CompRess-1q on most datasets. All methods employ 4 layer MLP heads only during distillation.

Arch	ResNet-50	MobileNet-v2			ResNet-18		
Method	Teacher	Comp-2q -4L-MLP	Comp-1q -4L-MLP	SimReg -4L-MLP	Comp-2q -4L-MLP	Comp-1q -4L-MLP	SimReg -4L-MLP
Food	72.3	71.4	72.5	**73.1**	61.7	**65.9**	65.4
CIFAR10	92.2	90.3	90.4	**91.2**	87.3	**89.3**	88.6
CIFAR100	75.1	73.9	74.5	**76.1**	68.4	**71.9**	70.2
SUN	60.2	58.0	58.1	**59.4**	54.3	56.0	**57.1**
Cars	50.8	60.3	**63.1**	62.4	37.2	**44.1**	42.3
Aircraft	53.5	57.7	**59.7**	58.7	42.3	**47.8**	45.8
DTD	75.1	71.7	71.3	**74.5**	69.3	**71.2**	70.9
Pets	83.6	**86.7**	86.3	85.6	84.0	**84.4**	83.9
Caltech	89.3	91.1	91.5	**91.7**	87.3	**90.1**	89.2
Flowers	91.3	94.3	**95.4**	95.1	86.4	**91.3**	90.9
Mean	74.3	75.5	76.3	**76.8**	67.8	**71.2**	70.4

In addition to transfer learning on different datasets on the classification task, we consider the task of object detection. We distill a ResNet-50 teacher to multiple ResNet-18 student networks with different MLP heads. The MLP heads are not part of the model fine-tuned for detection. Following [120], we use Faster-RCNN [255] with `R18-C4` backbone architecture for the detection task. All methods are trained on the VOC `trainval07+12` and tested on `test07` subsets of the PASCAL VOC [76] dataset using the code from [6]. We report the standard AP_{50}, AP(COCO-style) and AP_{75} metrics in table 2.12. Unlike in classification tasks, we find that the use of deeper MLP heads during distillation does not aid detection performance. The performance of different distillation architectures is nearly identical on the detection task.

TABLE 2.12. **Transfer learning for object detection on Pascal VOC dataset.** Student models with different MLP head architectures are used to perform distillation on ImageNet dataset and the backbone with R18-C4 architecture is fine-tuned on PASCAL VOC. Unlike in classification tasks, the performance of different distillation architectures is nearly identical.

Method	AP_{50}	AP	AP_{75}
SimReg-4L-MLP	74.0	45.4	47.8
SimReg-2L-MLP	74.2	45.5	47.4
SimReg-Linear	73.6	45.1	47.9

TABLE 2.13. **Role of augmentation strength:** During distillation either the 'same' augmented image or two 'different' augmentations of a single image are used as inputs to the teacher and student networks. The augmentations strength is varied for both the settings. We find that the performance is best when the same image with weak augmentation is used. This is significant since using different and stronger augmentations improve classification performance of SSL models but decrease their generalizability [243].

Aug Type	Aug Strength		ImageNet			Transfer
	Teacher	Student	1-NN	20-NN	Linear	Linear
Same	Weak	Weak	**54.8**	**59.9**	**65.1**	**70.4**
Same	Strong	Strong	53.4	59.0	64.3	70.3
Different	Weak	Weak	54.7	59.8	64.6	70.0
Different	Weak	Strong	51.1	56.5	62.0	68.9
Different	Strong	Strong	50.3	56.0	61.4	68.7

Effect of Data Augmentation: As shown in SEED [79] and CompRess [161], for a given student architecture, distillation from larger models trained using a particular method is better than directly training the student using the same method. Use of different and strong augmentations in contrastive SSL approaches has been shown to hurt generalization performance [243]. Here, we show that when distilling models, the best performance is obtained when the same augmented image with a weaker augmentation (details in suppl.) is used as input to both teacher and student networks (table 2.13). This suggests that our method can be used to improve generalizability of SSL models.

Multi-teacher Distillation: We train a single student model from multiple teacher networks trained with different SSL methods. Regression with a 4 layer MLP head significantly outperforms one with linear prediction (table 2.14).

Distillation of Supervised Teacher: All the previous teacher networks were trained in a self-supervised manner. We additionally analyze the distillation from a teacher trained with supervision (table 2.15). Note that the distillation remains unsupervised and only the backbone CNN features of the teacher are regressed.

TABLE 2.14. **Multi-teacher distillation on ImageNet.** We train a single student model (ResNet-18) from multiple SSL teacher networks (ResNet-50) using a common backbone network and a separate prediction head for each teacher. Networks with 4 layer prediction heads can better match each of the teachers and thus vastly outperform those with a linear head on both k-NN and linear evaluation metrics.

Training Inference			ResNet-18+Linear ResNet-18			ResNet-18+4L-MLP ResNet-18		
MoCo-v2	BYOL	SwAV	1-NN	20-NN	Linear	1-NN	20-NN	Linear
✓	✓		50.9	56.5	62.6	56.4	61.3	66.3
✓		✓	49.9	55.6	61.8	55.1	60.4	65.4
	✓	✓	52.0	57.7	64.8	56.5	61.4	67.0
✓	✓	✓	51.1	56.7	63.2	56.5	61.6	66.9

TABLE 2.15. **Distillation of supervised teacher.** Here, we analyze the role of MLP heads when distilling the features of a teacher trained using supervision. Note that the distillation remains unsupervised and only the backbone CNN features are regressed. Similar to distillation of SSL teachers, we observe that the use of a deep MLP head during training significantly improves classification performance on ImageNet classification task.

Teacher	Student Arch (Inference)	Prediction Head (Train)	1-NN	20-NN	Linear
Supervised ResNet-50	MobileNet-v2 Backbone	4L-MLP	63.77	67.87	**73.5**
		2L-MLP	**64.7**	**69.3**	**73.5**
		Linear	55.4	62.0	67.5

Similar to distillation of self-supervised teachers, we observe that the use of a deep MLP head during training significantly improves performance on the ImageNet classification task.

2.2.6 Conclusion

Distilling knowledge with deeper student networks leads to better downstream performance. We surprisingly find that intermediate layer outputs of a distilled student model have better performance compared to the final layer, though final layer is trained to mimic the teacher representations. Thus, we use a prediction MLP head only for optimizing the distillation objective and achieve boosts in performance with just the backbone network during inference. We believe studying the reasoning for this effect is an interesting future work. Our work also serves as an improved benchmark for future self-supervised distillation works. Additionally, we show that using the same weakly augmented image for both teacher and student aids distillation.

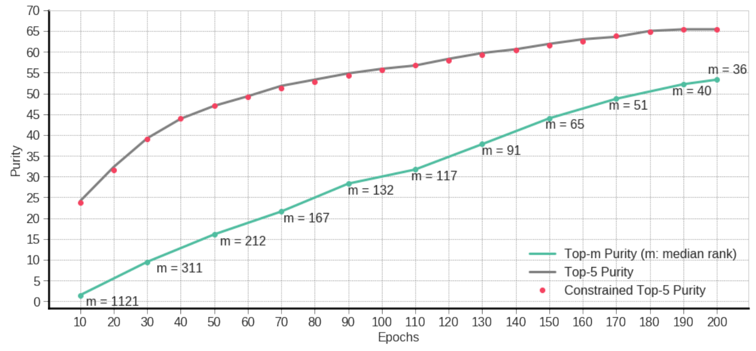


FIGURE 2.5. **Purity of constrained samples:** During training of CMSF$_{\text{self}}$, we plot purity of the top-5 samples in unconstrained set M (in black) and that of the top-5 samples in constrained set C (in red). The red curve is not significantly below the black one suggesting that the purity 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 5th 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.

TABLE 2.4. **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. CMSF$_{\text{self}}$ outperforms MSF on top-1 metric in both 1% and 10% settings.

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 [210]	✓	800	-	-	57.2	83.8
CO2 [317]	✓	200	-	-	71.0	85.7
SimCLR [48]	✓	1000	48.3	65.6	75.5	87.8
InvP [310]	✓	800	-	-	78.2	88.7
BYOL [107]	✓	1000	53.2	68.8	78.4	89.0
SwAV [38]	✓	800	**53.9**	**70.2**	**78.5**	**89.9**
MoCo v2 [51]	✗	800	51.5	63.6	77.6	86.1
BYOL [107]	✗	1000	55.7	**68.6**	80.0	**88.6**
CompRess [160]	✗	1K+130	**59.7**	67.0	**82.3**	87.5
MoCo v2 [51]	✗	200	43.6	58.4	71.2	82.9
BYOL-asym	✗	200	47.9	61.3	74.6	84.7
ISD [290]	✗	200	53.4	63.0	78.8	85.9
MSF [162]	✗	200	55.5	66.5	**79.9**	87.6
CMSF$_{\text{self}}$	✗	200	**56.4**	**67.5**	79.8	**87.7**

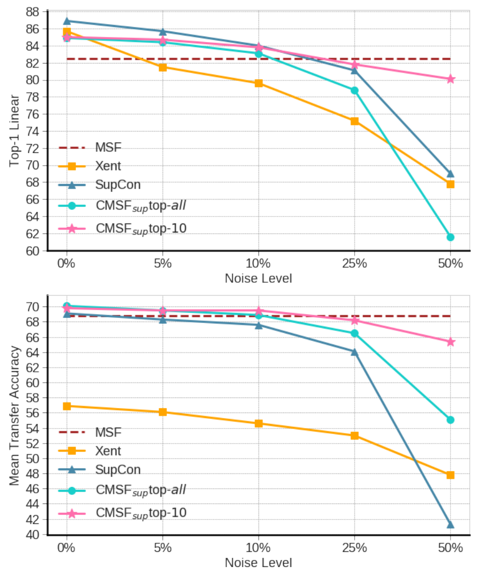


FIGURE 2.6. **Noisy supervised setting on ImageNet-100:** Our method is more robust to noisy annotation compared to Xent and SupCon. Also, using top-*all* degrades the results since all images from a single category are not guaranteed to be semantically related due to noisy labels. Mean Transfer Accuracy is the average over 10 transfer datasets.

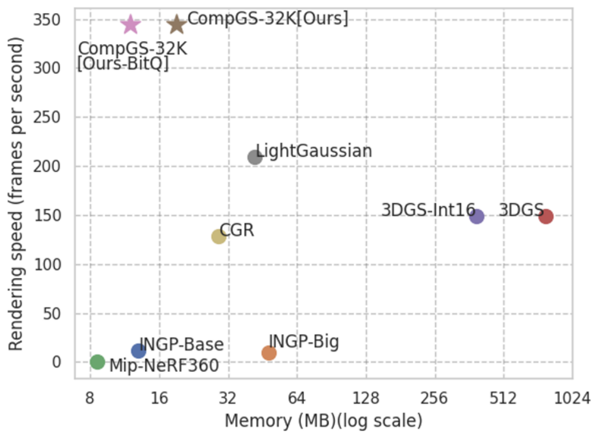
CHAPTER 3

Memory Efficiency of Models in 3D, Vision and NLP Applications

Here, we focus our attention on reducing storage memory of models for diverse tasks - novel view synthesis, generative tasks for large language models (LLM) and image classification. For the task of novel view synthesis, we consider 3D Gaussian splatting [151], a recent technique which achieves stellar reconstructions at real-time speeds. In our work Compact3D, we address its shortcoming related to model size by compressing the model parameters through vector quantization. For LLM and image classification tasks, we develop NOLA, a novel parameter efficient fine tuning (PEFT) method based on LoRA. LoRA provides a way to use a small number of parameters to fine-tune large vision and language models. NOLA further reduces the size of the fine-tuned model and makes it feasible to run tens of thousands of LLM variants on a single GPU.

3.1 CompGS: Smaller and Faster Gaussian Splatting with Vector Quantization

3D Gaussian Splatting (3DGS) is a new method for modeling and rendering 3D radiance fields that achieves much faster learning and rendering time compared to SOTA NeRF methods. However, it comes with a drawback in the much larger storage demand compared to NeRF methods since it needs to store the parameters for several 3D Gaussians. We notice that many Gaussians may share similar parameters, so we introduce a simple vector quantization method based on K-means to quantize the Gaussian parameters while optimizing them. Then, we store the small codebook along with the index of the code for each Gaussian. We compress the indices further by sorting them and using a method similar to run-length encoding. Moreover, we use a simple regularizer to encourage zero opacity (invisible Gaussians) to reduce the storage and rendering time by a large factor through reducing the number of Gaussians. We do extensive experiments on standard benchmarks as well as an existing 3D dataset that is an order of magnitude larger than the standard benchmarks used in this field. We show that our simple yet effective method can reduce the storage cost for 3DGS by 40× to 50× and rendering time by 2× to 3× with a very small drop in the quality of rendered images. Our code is available here: https://github.com/UCDvision/compact3d



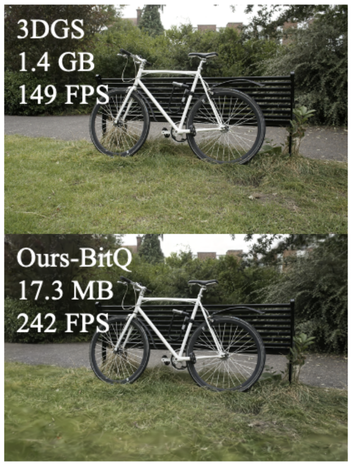


FIGURE 3.1. **Inference speed vs. memory comparison.** All methods except INGP achieve comparable PSNR that are reported in Table 3.1. CompGS, our compressed version of 3DGS, maintains the speed and performance of 3DGS while reducing its size to the levels of NeRF based approaches. We achieve around 45× compression and 2.5× inference speed up with little drop in performance (CompGS-32K). A bit quantized version of this (Ours-BitQ) compresses it further to a total compression of 65× with hardly noticeable difference in quality.

3.1.1 Introduction

Recently, we have seen great progress in radiance field methods to reconstruct a 3D scene using multiple images captured from multiple viewpoints. NeRF [208] is probably the most well-known method that employs an implicit neural representation to learn the radiance field using a deep model. Although very successful, NeRF methods are very slow to train and render. Several methods have been proposed to solve this problem; however, they usually come with some cost in the quality of the rendered images.

The Gaussian Splatting method (3DGS) [151] is a new paradigm in learning radiance fields. The idea is to model the scene using a set of Gaussians. Each Gaussian has several parameters including its position in 3D space, covariance matrix, opacity, color, and spherical harmonics of the color that need to be learned from multiple-view images. Thanks to the simplicity of projecting 3D Gaussians to the 2D image space and rasterizing them, 3DGS is significantly faster to both train and render compared to NeRF methods. This results in real-time rendering of the scenes on a single GPU (ref. Fig. 3.1). Additionally, unlike the implicit representations in NeRF, the 3D structure of the scene is explicitly stored in the parameter space of the Gaussians. This enables many operations including editing the 3D scene directly in the parameter space.

One of the main drawbacks of the 3DGS method compared to NeRF variants is that 3DGS needs at least an order of magnitude more parameters compared to NeRF. This increases the storage and communication requirements of the model and its memory at the inference time, which can be very limiting in

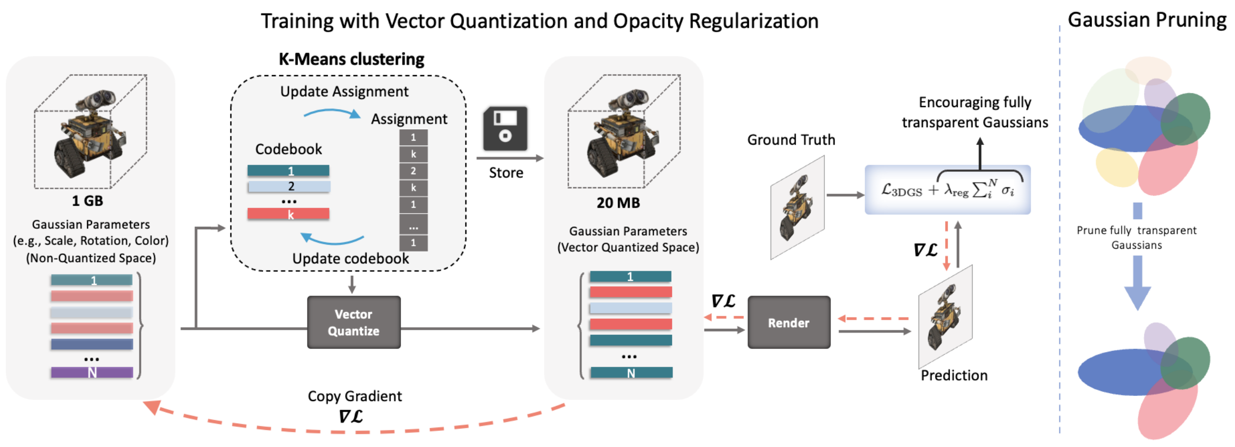

Training with Vector Quantization and Opacity Regularization
Gaussian Pruning
K-Means clustering
Update Assignment
Assignment
Codebook
Update codebook
Store
1 GB
Gaussian Parameters (e.g., Scale, Rotation, Color) (Non-Quantized Space)
20 MB
Gaussian Parameters (Vector Quantized Space)
Vector Quantize
Render
Ground Truth
Prediction
Encouraging fully transparent Gaussians
$\mathcal{L}_{3DGS} + \lambda_{reg} \sum_i^N \sigma_i$
$\nabla\mathcal{L}$
Copy Gradient
Prune fully transparent Gaussians

FIGURE 3.2. **Overview of CompGS vector quantization:** We compress 3DGS using vector quantization of the parameters of the Gaussians. The quantization is performed along with the training of the Gaussian parameters. Considering each Gaussian as a vector, we perform K-means clustering to represent the N Gaussians in the model with k cluster centers (codes). Each Gaussian is then replaced by its corresponding code for rendering and loss calculation. The gradients wrt centers are copied to all the elements in the corresponding cluster and the non-quantized versions of the parameters are updated. Only the codebook and code assignments for each Gaussian are stored and used for inference. To further reduce the storage and inference time, we regularize opacity in the loss to encourage fully transparent Gaussians. CompGS maintains the real-time rendering property of 3DGS while compressing it by an order of magnitude.

many real-world applications involving smaller devices. For instance, the large memory consumption may be prohibitive in storing, communicating, and rendering several radiance field models on AR/VR headsets.

We are interested in compacting 3DGS representations without sacrificing their rendering speed to enable their usage in various applications including low-storage or low-memory devices and AR/VR headsets. Our main intuition is that several Gaussians may share some of their parameters (e.g. covariance matrix). Hence, we simply vector-quantize parameters while learning and store the codebook along with the index for each Gaussian. This can result in a huge reduction in the storage. Also, it can reduce the memory footprint at the rendering time since the index can act as a pointer to the correct code freeing the memory needed to replicate those parameters for all Gaussians.

To this end, we use simple K-means algorithm to vector quantize the parameters at the learning time. Inspired by various quantization-aware learning methods in deep learning [248], we use the quantized model in the forward pass while updating the non-quantized model in the backward pass. To reduce the computation overhead of running K-means, we update the centroids in each iteration, but update the assignments less frequently (e.g., once every 100 iterations) since it is costly. Moreover, since the Gaussians are a set of non-ordered elements, we compress the representation further by sorting the Gaussians based on one of the quantized indices and storing them using the Run-Length-Encoding (RLE) method. Furthermore, we employ a simple regularizer to promote zero opacity (essentially invisible Gaussians), resulting in a significant reduction in storage and rendering time by reducing the number of Gaussians. Our final model is 40× to 50× smaller and 2× to 3× faster during rendering compared to 3DGS.

3.1.2 Related Work

Novel-view synthesis methods: Early deep learning techniques for novel-view synthesis used CNNs to estimate blending weights or texture-space solutions [87, 123, 256, 291, 359]. However, the use of CNNs faced challenges with MVS-based geometry and caused temporal flickering. Volumetric representations began with Soft3D [239], and subsequent techniques used deep learning with volumetric ray-marching [127, 273]. Mildenhall et al. introduced Neural Radiance Fields (NeRFs) [208] to improve the quality of synthesized novel views, but the use of a large Multi-Layer Perceptron (MLP) as the backbone and dense sampling slowed down the process a lot. Successive methods sought to balance quality and speed, with Mip-NeRF360 achieving top image quality [19]. Recent advances prioritize faster training and rendering via spatial data structures, encodings, and MLP adjustments [55, 88, 93, 124, 213, 253, 284, 325, 340]. Notable methods, like InstantNGP [213], use hash grids and occupancy grids for accelerated computation with a smaller MLP, while Plenoxels [88] entirely forgo neural networks, relying on Spherical Harmonics for directional effects. Despite impressive results, challenges in representing empty space, limitations in image quality, and rendering speed persist in NeRF methods. In contrast, 3DGS [151] achieves superior quality and faster rendering without implicit learning [19]. However, the main drawback of 3DGS is its increased storage compared to NeRF methods which may limit its usage in many applications such as edge devices. We are able to keep the quality and fast rendering speed of 3DGS method while providing reduced model storage by applying a vector quantization scheme to Gaussian parameters.

Bit quantization: Reducing the number of bits to represent each parameter in a deep neural network is a commonly used method to quantize models [112, 146, 165] that result in smaller memory footprints. Representing weights in 64 or 32-bit formats may not be crucial for a given task, and a lower-precision quantization can lead to memory and speed improvements. Dettmers et al. [66] show 8-bit quantization is sufficient for large language models. In the extreme case, weights of neural networks can be quantized using binary values. XNOR [249] examines this extreme case by quantization-aware training of a full-precision network that is robust to quantization transformations.

Vector quantization: Vector quantization (VQ) [75, 95, 101, 105] is a lossy compression technique that converts a large set of vectors into a smaller codebook and represents each vector by one of the codes in the codebook. As a result, one needs to store only the code assignments and the codebook instead of storing all vectors. VQ is used in many applications including image compression [61], video and audio codec [178, 205], compressing deep networks [57, 101], and generative models [108, 250, 304]. We apply a similar method to compressing 3DGS models.

Deep model compression. Model compression tries to reduce the storage size without changing the accuracy of original models. Model compression techniques can be divided to 1) model pruning [110, 112, 305, 322] that aims to remove redundant layers of neural networks; 2) weight quantization [146, 165, 223], and 3) knowledge distillation [16, 41, 130, 160, 242], in which a compact student model is trained to mimic the original teacher model. Some works have applied these techniques to volumetric radiance fields [64, 184, 340]. For instance, TensoRF [40] decompose volumetric representations via low-rank approximation.

Compression for 3D scene representation methods. Since NeRF relies on dense sampling of color values and opacity, the computational costs are significant. To increase efficiency, methods adopt different data structures such as trees [314, 340], point clouds [238, 333], and grids [40, 88, 213, 269, 279, 284]. With grid structures training iterations can be completed in a matter of minutes. However, dense 3D grid structures may require substantial amounts of memory. Several methods have worked on reducing the size of such volumetric grids [40, 213, 284, 286]. Instant-NGP [213] uses hash-based multi-resolution grids. VQAD [284] replaces the hash function with codebooks and vector quantization. Another line of work decomposes 3D grids into lower dimensional components, such as planes and vectors, to reduce the memory requirements [40, 138, 286]. Despite reducing the time and space complexity of the 3D scenes, their sizes are still larger than MLP-based methods. VQRF [183] compresses volumetric grid-based radiance fields by adopting the VQ strategy to encode color features into a compact codebook.

While we also employ vector quantization, we differ from the above approaches in the method employed for novel view synthesis. Unlike the NeRF based approaches described above, we aim to compress 3DGS which uses a collection of 3D Gaussians to represent the 3D scene and does not contain grid like structures or neural networks. We also achieve a significant amount of compression by regularizing and pruning the Gaussians based on their opacity.

Concurrent works: Some very recent works developed concurrently to ours [78, 99, 176, 212, 220] also propose vector quantization and pruning based methods to compress 3D Gaussian splat models. LightGaussian [78] uses importance based Gaussian pruning and distillation and vector quantization of spherical harmonics parameters. Similarly, CGR [176] masks Gaussians based on their volume and transparency to reduce the number of Gaussians and uses residual vector quantization for scale and rotation parameters. In CGS [220], highly sensitive parameters are left non-quantized while the less sensitive ones are vector quantized.

3.1.3 Method

Here, we briefly describe the 3DGS [151] method for learning and rendering 3D scenes and explain our vector quantization approach for compressing it.

Overview of 3DGS: 3DGS models a scene using a collection of 3D Gaussians. A 3D Gaussian is parameterized by its position and covariance matrices in the 3D space. $G(x) = e^{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)}$ where $x-\mu$ is the position vector, μ is the position, and Σ is the 3D covariance matrix of the Gaussian. Since the covariance matrix needs to be positive definite, it is factored into its scale (S) and rotation (R) matrices as $\Sigma = RSS^TR^T$ for easier optimization. In addition, each Gaussian has an opacity parameter σ. Since the color of the Gaussians may depend on the viewing angle, the color of each Gaussian is modeled by a Spherical Harmonics (SH) of order 3 in addition to a DC component.

Given a view-point, the collection of 3D Gaussians is efficiently rendered in a differentiable manner to get a 2D image by α-blending of anisotropic splats, sorting, and using a tile-based rasterizer. Color of a pixel is given by $C = \sum_i c_i\alpha_i \prod_{j=1}^{i-1}(1-\alpha_j)$ where c_i is the color of the i^{th} Gaussian and α_i is the product of the value of the Gaussian at that point and its learned opacity σ_i. At the training time, 3DGS minimizes the loss between the groundtruth and rendered images in the pixel space. The loss is ℓ_1 loss plus an SSIM loss in the pixel space. 3DGS initializes the optimization by a point cloud achieved by a standard SfM method and iteratively prunes the ones with small opacity and adds new ones when the gradient is large. 3DGS paper shows that it is extremely fast to train and is capable of real-time rendering while matching or outperforming SOTA NeRF methods in terms of rendered image quality.

Compression of 3DGS:

We compress the parameters of 3DGS using vector quantization aware training and reduce the number of Gaussians by regularizing the opacity parameter.

Vector quantization: 3DGS requires a few million Gaussians to model a typical scene. With 59 parameters per Gaussian, the storage size of the trained model is an order of magnitude larger than most NeRF approaches (e.g., Mip-NeRF360 [19]). This makes it inefficient for some applications including edge devices. We are interested in reducing the number of parameters. Our main intuition is that many Gaussians may have similar parameter values (e.g., covariance). Hence, we use simple vector quantization using K-means algorithm to compress the parameters. Fig. 4.1 provides an overview of our approach.

Consider a 3DGS model with N Gaussians, each with a d dimensional parameter vector. We run K-means algorithm to cluster the vectors into K clusters. Then, one can store the model using K vectors of size d and N integer indices (one for each Gaussian). Since $N >> K$, this method can result in a large compression ratios. In a typical scene, N is a few millions while K is a few thousands.

However, clustering the model parameters after training results in performance degradation, hence, we perform quantization aware training to ensure that the parameters are amenable to quantization. In learning 3DGS, we store the non-quantized parameters. In the forward pass of learning 3DGS, we quantize the parameters and replace them with the quantized version (centroids) to do the rendering and calculate the loss. Then, we do the backward pass to get the gradients for the quantized parameters and copy the gradients to the non-quantized parameters to update them. We use straight-through estimator proposed in STE [24]. After learning, we discard the non-quantized parameters and keep only the codebook and indices of the codes for Gaussians.

Since the number of Gaussians N is typically in millions, cost of performing K-means at every iteration of training can be prohibitively high. K-means has two steps: updating centroids given assignments, and updating assignments given centroids. We note that the latter is more expensive while the former is a simple averaging. Hence, we update the centroids after each iteration and update the assignments once every t iterations. We observe that the modified approach works well even for values of t as high as 500. This is crucial in limiting the training time of the method.

Performing a single K-means for the whole d dimensional parameters requires a huge codebook since the different parameters of the Gaussian are not necessarily correlated. Hence, we group similar types of parameters, e.g., all rotation matrices, together and cluster them independently to learn a separate codebook for each. This requires storing multiple indices for each Gaussian. In our main method, we quantize DC component of color, spherical harmonics, scale, and rotation parameters separately, resulting in 4 codebooks. We do not quantize opacity parameter since it is a single scalar and do not quantize the position of the Gaussians since sharing them results in overlapping Gaussians.

Since the indices are integer values, we use fewer number of bits compared to the original parameters to store each. Moreover, 3DGS models the scene as a set of order-less Gaussians. Hence, we sort the Gaussians based on one of the indices, e.g., rotation, so that Gaussians using the same code appear together in the list. Then, for that index, instead of storing n integers, we store how many times each code appears in the list, reducong the storage from n integers to k integers. This is similar to run-length-encoding for data compression.

Opacity Regularization: Some parameters like position of the Gaussians cannot be quantized easily, so as shown in Table 3.6 after quantization, they dominate the memory(more than 80% of memory). This means quantization cannot improve the compression any further. One way to compress 3DGS more is to reduce the number of Gaussians. Interestingly, this reduction comes with a bi-product that is increase in inference speed. We know that very small values of opacity (σ) correspond to transparent or nearly invisible Gaussians. Hence, inspired by training sparse models, we add ℓ_1 norm of the opacity to the loss as a regularizer to encourage zero values for opacity. Therefore, the final loss becomes: $L = L_{\text{3DGS}} + \lambda_{\text{reg}} \sum_i^N \sigma_i$, where L_{3DGS} is the original loss of 3DGS with or without quantization and λ_{reg} controls the sparsity of opacity. Finally, similar to the original 3DGS, we remove the Guassians with opacity smaller than a threshold, resulting in significant reduction in storage and inference time.

3.1.4 Experiments

Implementation details: For all our experiments, we use the publicly available official code repository [5] of 3DGS [151] provided by its authors. There are no changes in the hyperparameters used for training compared to 3DGS. The Gaussian parameters are trained without any vector quantization till $20K$ iterations and K-means quantization is used for the remaining $10K$ iterations. A standard K-means iteration involves distance calculation between all elements (Gaussian parameters) and all cluster centers followed by assignment to the closest center. The centers are then updated using new cluster assignments and the loop is repeated. We use just 1 such K-means iteration in our experiments once every 100 iterations till iteration $25K$ and keep the assignments constant thereafter till the last iteration, $30K$. The K-means cluster centers are updated using the non-quantized Gaussian parameters after each iteration of training. The covariance (scale and rotation) and color (DC and harmonics) components of each Gaussian is vector quantized while position (mean) and opacity parameters are not quantized. Additional results with different parameters being quantized are provided in Table 3.9. Unless mentioned differently, we use a codebook of size 4096 for the color and 16384 (CompGS $16K$) for the covariance parameters. The scale parameters of covariance are quantized before applying the exponential activation on them. Similarly, quaternion based rotation parameters are quantized before normalization. For opacity regularization, we use $\lambda_{\text{reg}} = 10^{-7}$ from iterations $15K$ to $20K$ along with opacity based pruning every 1000 iterations and remove regularization thereafter. All experiments were run on a single RTX-6000 GPU.

TABLE 3.1. **Comparison with SOTA methods for novel view synthesis.** 3DGS [151] performs comparably or outperforms the best of the NeRF based approaches while maintaining a high rendering speed during inference. Trained NeRF models are significantly smaller than 3DGS since NeRFs are parameterized using neural networks while 3DGS requires storage of parameters of millions of 3D Gaussians. CompGS is a vector quantized version of 3DGS that maintains the speed and performance advantages of 3DGS while being 40× to 50× **smaller**. CompGS 32K BitQ is the post-training bit quantized version of CompGS 32K, in which position parameters are 16-bits, opacity is 8 bits, and the rest are 32 bits. *Reproduced using official code. † Reported from 3DGS [151]. Our timings for 3DGS and CompGS are reported using a RTX6000 GPU while those with † used A6000 GPU. We boldface entries for emphasis. Please see the Appendix for results on Deep Blending dataset.

	Mip-NeRF360						Tanks&Temples					
Method	SSIM↑	PSNR↑	LPIPS↓	FPS	Mem (MB)	Train Time(m)	SSIM↑	PSNR↑	LPIPS↓	FPS	Mem (MB)	Train Time(m)
Plenoxels† [88]	0.626	23.08	0.463	6.79	2,100	25.5	0.719	21.08	0.379	13.0	2300	25.5
INGP-Base† [213]	0.671	25.30	0.371	11.7	13	5.37	0.723	21.72	0.330	17.1	13	5.26
INGP-Big† [213]	0.699	25.59	0.331	9.43	48	7.30	0.745	21.92	0.305	14.4	48	6.59
M-NeRF360† [19]	0.792	27.69	0.237	0.06	8.6	48h	0.759	22.22	0.257	0.14	8.6	48h
3DGS † [151]	0.815	27.21	0.214	134	734	41.3	0.841	23.14	0.183	154	411	26.5
3DGS * [151]	0.813	27.42	0.217	149	778	21.6	0.844	23.68	0.178	206	433	12.2
LigthGaussian [78]	0.805	27.28	0.243	209	42	-	0.817	23.11	0.231	209	22	-
CGR [?]	0.797	27.03	0.247	128	29.1	-	0.831	23.32	0.202	185	20.9	-
CGS [?]	0.801	26.98	0.238	-	28.8	-	0.832	23.32	0.194	-	17.28	-
CompGS 16K	0.804	27.03	0.243	346	18	22.8	0.836	23.39	0.200	479	12	15.6
CompGS 32K	0.806	27.12	0.240	344	19	29.4	0.838	23.44	0.198	475	13	20.6
CompGS 32K BitQ	0.797	26.97	0.245	344	12	29.4	0.832	23.35	0.202	475	8	20.6

Datasets: We primarily show results on three challenging real world datasets - Tanks&Temples [154], Deep Blending [123] and Mip-NeRF360 [19] containing two, two and nine scenes respectively. Also, we provide results on a subset of the recently released DL3DV-10K dataset [?] which contains 140 scenes. DL3DV-10K [?] is an annotated dataset with $10,510$ real-world scene-level videos. Out of these, 140 scenes have been used to create a novel-view synthesis (NVS) benchmark, making it an order of magnitude larger than the typical NVS benchmarks. We use this NVS benchmark in our experiments. Additionally, we provide results on a subset of the large scale ARKit [21] dataset, called ARKit-200, which contains 200 scenes. Details of this dataset is presented in the Appendix.

Baselines: As we propose a method (termed CompGS) for compacting 3DGS, we focus our comparisons with 3DGS and different baseline methods for compressing it. We consider bit quantization (denoted as Int-16/8/4 in results) and 3DGS without the harmonic components for color (denoted as 3DGS-No-SH) as alternative parameter compression methods. Bit-quantization is performed using the standard Absmax quantization [66] technique. Similarly, we consider several alternative approaches to reduce the number of Gaussians. Densification process in 3DGS increases the Gaussian count and is controlled by the gradient threshold (termed grad thresh) parameter and the frequency (freq) and iterations (iters) until densification is performed. The opacity threshold (min opacity) controls the pruning of transparent Gaussians. We modify these parameters in 3DGS to compress the model with as little drop in performance as possible. Additionally, Table 3.1 shows comparison with state-of-the-art NeRF approaches [19, 88, 213]. Mip-NeRF360 [19] achieves high performance comparable to 3DGS while Plenoxels [88] and InstantNGP [213] have high frame-rate for rendering and very low training time. InstantNGP and Mip-NeRF360 are also comparable in model size to our compressed model.

Evaluation: For a fair comparison, we use the same train-test split as Mip-NeRF360 [19] and 3DGS [151] and directly report the metrics for other methods from 3DGS [151]. We also report our reproduced metrics for 3DGS since we observe slightly better results compared to the ones in [151]. We report the standard evaluation metrics of SSIM, PSNR and LPIPS along with memory or compression ratio, rendering FPS and training time. The common practice is to report the average of PSNR across a set of images and scenes. However, this metric may be dominated by very accurate reconstructions (smaller errors) since it is based on the geometric average of the errors due to the log operation in PSNR calculation. Hence, for the larger ARKit dataset, we also report PSNR-AM for which we average the error across all images and scenes before calculating the PSNR. In comparing model sizes, we normalize all methods by dividing them by the size of our method to obtain compression ratio.

Results: Comparison of our results with SOTA novel view synthesis approaches is shown in Table 3.1. Our vector quantized method has a comparable performance to the non-quantized 3DGS with a small drop on MipNerf-360 and TandT datasets and a small improvement on the DB dataset. We additionally report results with post-training bit quantization of our model (CompGS BitQ) where the position and opacity parameters are quantized to 16 bits and 8 bits respectively. The model memory footprint drastically reduces for CompGS compared to 3DGS , making it comparable to NeRF approaches. Our models are 65× and 54× **smaller** than 3DGS models on MipNerf-360 and TandT datasets respectively. This reduces a big disadvantage of 3DGS models and makes them more practical. The compression achieved by CompGS is impressive considering that more than two-thirds of its memory is due to the non-quantized position and opacity parameters (refer table 3.6). Additionally CompGS maintains the other advantages of 3DGS such as low inference memory usage and training time. while also **increasing its already impressive rendering FPS by** 2× **to** 3×. A limitation of CompGS compared to 3DGS is the overhead in compute and training time introduced by the K-means clustering algorithm. This is compensated in part by the reduced compute and time due to the decrease in Gaussian count. CompGS 16K variant requires marginally more time than 3DGS while CompGS 32K needs 1.4× to 1.7× more training time. However, this is still orders of magnitude

Table 3.2. **Comparison of parameter compression methods for 3DGS.** We evaluate different baseline approaches for compressing the parameters of 3DGS without any reduction in the number of Gaussians. All memory values are reported as a ratio of the method with our smallest model. Our K-Means based vector quantization performs favorably compared to all methods both in terms of novel view synthesis performance and compression. Not quantizing the position values (Int-x no-pos) is crucial in bit quantization. Since harmonics constitute 76% of each Gaussian, 3DGS-no-SH achieves a high level of compression. But CompGS with only quantized harmonics achieves similar compression with nearly no loss in performance compared to 3DGS .

	Mip-NeRF360			Tanks&Temples			Deep Blending			
Method	SSIM	PSNR	LPIPS	SSIM	PSNR	LPIPS	SSIM	PSNR	LPIPS	Mem
3DGS	0.813	27.42	0.217	0.844	23.68	0.178	0.899	29.49	0.246	20.0
3DGS-No-SH	0.802	26.80	0.229	0.833	23.16	0.190	0.900	29.50	0.247	4.8
Post-train K-means 4K	0.768	25.46	0.266	0.803	22.12	0.226	0.887	28.61	0.268	1.7
K-means 4K Only-SH	0.811	27.25	0.223	0.842	23.57	0.183	0.902	29.60	0.246	4.8
K-means 4K	0.804	26.97	0.234	0.836	23.31	0.194	0.904	29.76	0.248	1.7
K-means 32K	0.808	27.16	0.228	0.840	23.47	0.188	0.903	29.75	0.247	1.8
Int16	0.804	27.25	0.223	0.836	23.56	0.185	0.900	29.49	0.247	10.0
Int8 no-pos	0.812	27.38	0.219	0.843	23.67	0.180	0.900	29.47	0.247	5.8
Int8	0.357	14.41	0.629	0.386	12.37	0.625	0.709	21.58	0.457	5.0
Int4 no-pos	0.489	17.42	0.525	0.488	12.94	0.575	0.746	19.90	0.446	3.4
3DGS-No-SH Int16	0.789	26.59	0.237	0.826	23.04	0.198	0.900	29.50	0.248	2.4
K-means 4K, Int16	0.796	26.83	0.239	0.830	23.21	0.199	0.904	29.76	0.248	1.0

smaller than the high-quality NeRF based approaches like MipNerf-360. Per-scene evaluation metrics are in Appendix. Note that there are large differences in reproduced results for 3DGS across various works in the literature. We observe a median standard deviation of 0.05dB for PSNR when the experiment is repeated 20 times with several scenes having differences more than 0.4dB across runs (refer Appendix). One must be careful when analyzing as these variations are often comparable to differences in performance between methods.

We decouple our compression method into parameter and Gaussian count compression components and perform ablations on each of them.

Comparison of parameter compression methods: In Table 3.2, we compare the proposed vector quantization based compression against other baseline approaches for compressing 3DGS. Since the spherical harmonic components used for modeling color make up nearly three-fourths of all the parameters of each Gaussian, a trivial compression baseline is to use a variant of 3DGS with only the DC component for color and no harmonics. This baseline (3DGS-No-SH) achieves a high compression with just 23.7% of the original model size but has a drop in performance. Our K-Means approach outperforms 3DGS-No-SH while using less than half its memory. We also consider a variant of CompGS with a single codebook for both SH

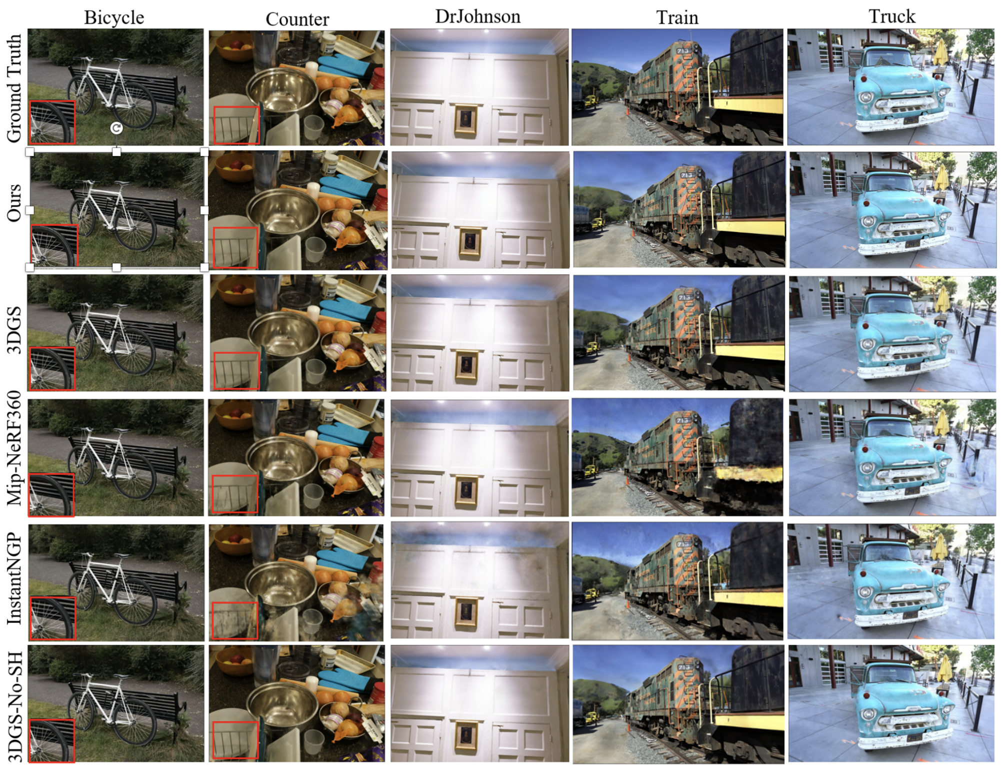


FIGURE 3.3. **Qualitative comparison of novel view synthesis approaches**. We visualize images from different scenes across datasets for SOTA NeRF, 3DGS, our CompGS and the No-SH variant of 3DGS . All methods based on 3DGS have better reconstruction of finer details like spokes of the bicycle wheel compared to NeRF approaches. Both compressed versions CompGS and 3DGS-No-SH are similar in appearance to 3DGS with no additional visually apparent errors.

and DC parameters (termed SH+DC) with a larger codebook of size of 4096. This has a marginal decrease in both memory and performance compared to default CompGS suggesting that correlated parameters can be combined to reduce the number of indices to be stored.

Fig. 3.3 shows qualitative comparison of CompGS across multiple datasets with both SOTA approaches and compression methods for 3DGS . Both CompGS and 3DGS-No-SH are visually similar to 3DGS, preserving finer details such as the spokes of the bike and bars of dish-rack. Among NeRF approaches, Mip-NeRF360 is closest in terms of quality to 3DGS while InstantNGP trades-off quality for inference and training speed.

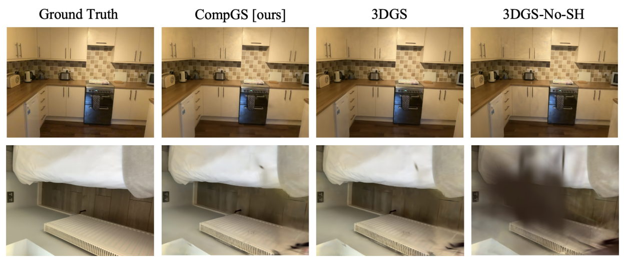
All the above approaches are trained using 32-bit precision for all Gaussian parameters. Post-training bit quantization of 3DGS to 16-bits reduces the memory by half with very little drop in performance. However, reducing the precision to 8-bits results in a huge degradation of the model. This drop is due to the quantization of the position parameters of the Gaussians. Excluding them from quantization (denoted as Int8 no-pos) results in a model comparable to the 32-bit variant. However, further reduction to 4-bits degrades the model even when the position parameters are not quantized. Note that bit quantization approaches offer

TABLE 3.3. **Reducing number of Gaussians in 3DGS.** We evaluate different baseline approaches for compressing 3DGS by reducing the number of Gaussians. Gaussian count is proportional to model size. CompGS performs favorably compared to all methods both in terms of novel view synthesis performance and compression.

	Mip-NeRF360				Tanks&Temples				Deep Blending			
Method	SSIM	PSNR	LPIPS	#Gauss	SSIM	PSNR	LPIPS	#Gauss	SSIM	PSNR	LPIPS	#Gauss
3DGS	0.813	27.42	0.217	3.30M	0.844	23.68	0.178	1.83M	0.899	29.49	0.246	2.80M
Min Opacity	0.802	27.12	0.244	1.46M	0.833	23.44	0.204	780K	0.902	29.50	0.255	1.01M
Densify Freq	0.794	26.98	0.255	1.07M	0.832	23.36	0.206	709K	0.902	29.76	0.258	844K
Densify Iters	0.780	27.02	0.267	1.12M	0.835	23.55	0.194	810K	0.896	29.42	0.264	795K
Grad Thresh	0.769	26.57	0.292	809K	0.825	23.31	0.217	578K	0.900	29.49	0.260	1.01M
Opacity Reg	0.813	27.42	0.227	845K	0.844	23.71	0.188	520K	0.905	29.73	0.249	554K

TABLE 3.4. **Comparison on ARKit-200 dataset.** It contains 200 scenes from the ARKit [20] indoor scene understanding dataset (see the Appendix for details.). We report results for just the vector quantized version of CompGS. (left) CompGS achieves a high level of compression with nearly identical metrics for view synthesis. (right) 3DGS-No-SH fails to reconstruct well in several images while CompGS is nearly identical to 3DGS with a large reduction in model size.

Method	SSIM	PSNR	PSNR-AM	LPIPS	Mem
3DGS	0.909	25.76	20.73	0.226	20.0
3DGS-No-SH	0.905	25.31	20.11	0.234	4.8
CompGS	0.909	25.70	20.73	0.229	1.7



	DL3DV-140					
Method	SSIM$^\uparrow$	PSNR$^\uparrow$	PSNR-AM$^\uparrow$	LPIPS$^\downarrow$	FPS	Mem(MB)
3DGS *	0.905	29.06	27.37	0.134	282	291
CompGS 32K	0.895	28.42	26.97	0.149	566	10

TABLE 3.5. Results on the 140 scenes NVS benchmark of DL3DV-10K [**?**] dataset. Similar to the results on the smaller benchmarks, CompGS compresses 3DGS by nearly 30 times with a small drop in reconstruction quality. * is our reproduced results.

significantly lower compression compared to CompGS and they are a subset of the possible solutions for our vector quantization method. Similar to 3DGS, CompGS has a small drop in performance when 16-bit quantization is used.

Comparison of Gaussian count compression methods: In Table 3.3, we compare the proposed opacity regularization method for reducing the Gaussian count with baselines. In these baselines, we modify the 3DGS parameters to decrease densification and increase pruning and thus reduce the number of Gaussians. We report the best metrics for each baseline here (refer Appendix for ablation). Our opacity regularization results in 3.5× to 5× reduction in Gaussian count with nearly identical performance as the larger models.

Similar level of compression is achieved only by the gradient threshold baseline that reduces densification. However, it results in a large drop in performance.

Results on ARKit and DL3DV datasets: Table 3.4 shows the quantitative and qualitative results on our large-scale ARKit benchmark. Our compressed model achieves nearly the same performance as 3DGS with ten times smaller memory. Unlike CompGS , the 3DGS-No-SH method suffers a significant drop in quality. We also report PSNR-AM as the PSNR calculated using arithmetic mean of MSE over all the scenes in the dataset to prevent the domination of high-PSNR scenes. Similarly, Table 3.5 shows the performance of CompGS with both KMeans quantization and opacity regularization. CompGS achieves nearly 30× compression compared to 3DGS with a small drop in performance.

3.1.4.1 Ablations

We analyze our design choices and the effect of various hyperparameters on reconstruction performance and model size.

Memory break-down of CompGS: In Table 3.6, we show the contribution of various components to the final memory usage of CompGS . Out of 59 parameters of each Gaussian, we quantize 55 parameters of color and covariance while the 3 position and 1 opacity parameters are used as is. However, the bulk of the stored memory (68% and 81% for 16- and 32-bits) is due to the non-quantized parameters. For the quantized parameters, nearly all the memory is used to store the cluster assignment indices with less than 2% used for the codebook.

Trade-off between performance, compression, and training time: Compressing the Gaussian parameters comes with a trade-off, particularly between performance and training time. In our method, the size of

TABLE 3.6. **Breakdown of memory usage in CompGS.** We observe that just 4 non-quantized values of the total 59 values per Gaussian contribute to 68% and 81% of the total memory in our 16-bit and 32-bit variants respectively. For the quantized parameters, nearly the entire memory is used to store the indices.

	Non Quant	Quant
Num Params	4	55
Mem (16bit)	68%	32%
Mem (32bit)	81%	19%

k-Means Quantization	
Index	Codebook
99%	1%
98%	2%

TABLE 3.7. **Compression-performance tradeoff.** Gaussian count decreases drastically with heavy regularization but also results in some drop in performance on Mip-NeRF360 dataset. We choose $\lambda = 10^{-7}$ as default.

$\lambda_{\text{reg}}(\times 10^{-7})$	SSIM	PSNR	LPIPS	#Gauss
0.5	0.808	27.17	0.234	1.21M
1.0	0.806	27.12	0.240	845K
2.0	0.801	26.98	0.253	536K
3.0	0.794	26.83	0.266	390K

TABLE 3.8. **Performance and training time trade-off.** Depending on user's needs, it is possible to obtain models with fast training or high performance. The hyperparameters of vector quantization - number of K-Means iterations (iters), K-Means frequency (freq) and codebook size (#codes) can be varied to obtain the desired point on the curve. They offer a good trade-off, with huge decrease in training time with minor changes in performance. Results are shown on MipNerf-360.

Iters	Freq	#Codes	SSIM	PSNR	Time
1	100	8K	0.802	26.94	19.3
3	100	8K	0.802	26.94	20.9
5	100	8K	0.802	26.95	22.5
10	100	8K	0.802	26.95	26.5
5	50	8K	0.803	27.00	28.7
5	200	8K	0.799	26.76	19.4
5	500	8K	0.783	26.15	18.1
5	100	4K	0.800	26.84	19.6
5	100	16K	0.804	27.05	28.9
5	100	32K	0.806	27.12	42.4

TABLE 3.9. **Effect of quantization on different Gaussian parameters.** Each Gaussian in 3DGS is parameterized using position (pos), scale, rotation (rot) and color (DC and harmonics SH). We analyze the effect of quantizing combinations of these parameters on the view synthesis performance. SH+DC denotes that a single codebook is used for both SH and DC. Position values cannot be quantized without greatly affecting model performance. The rest of the parameters can be simultaneously combined to obtain a high degree of compression without much loss in quality.

Quantized	Train		Truck		
Params	SSIM↑	PSNR↑	SSIM↑	PSNR↑	Mem
3DGS	0.811	21.99	0.878	25.38	20.0
3DGS-No-SH	0.798	21.40	0.871	24.92	4.8
Variants of CompGS					
Pos	0.673	19.81	0.730	21.65	19.0
SH	0.809	21.88	0.876	25.27	4.8
SH, DC	0.806	21.68	0.875	25.24	3.8
Rot(R)	0.808	21.83	0.876	25.32	18.7
Scale(Sc)	0.809	21.79	0.877	25.30	19.0
SH,R	0.805	21.67	0.874	25.20	3.5
SH,Sc	0.806	21.63	0.875	25.18	3.8
SH,Sc,R	0.801	21.64	0.872	25.02	2.6
SH+DC,Sc,R	0.797	21.41	0.868	24.89	1.6
SH,DC,Sc,R	0.801	21.64	0.871	24.97	1.7
SH,DC,Sc,R Int16	0.790	21.49	0.869	24.93	1.0

codebook, frequency of code assignment and number of iterations in code computation control this trade-off. Similarly, regularization strength can be modified in Gaussian count reduction to obtain a trade-off between performance and compression. We show ablations on these hyperparameters in Tables 3.7 and 3.8. CompGS offers great flexibility, with different levels of compression and training time without sacrificing much on performance.

Parameter selection for quantization: Table 3.9 shows the effect of quantizing different subsets of the Gaussian parameters on the Tanks&Temples dataset. Quantizing the position parameters significantly reduces the performance on both the scenes. We thus do not quantize position in any of our other experiments. Quantizing only the harmonics (SH) of color parameter is nearly identical in size to the no-harmonics (3DGS-No-SH) of 3DGS . Our SH has very little drop in metrics compared to 3DGS while 3DGS-No-SH is much worse off without the harmonics. As more parameters are quantized, the performance of CompGS slowly reduces. The combination of all color and covariance parameters still results in a model with good qualitative and quantitative results.

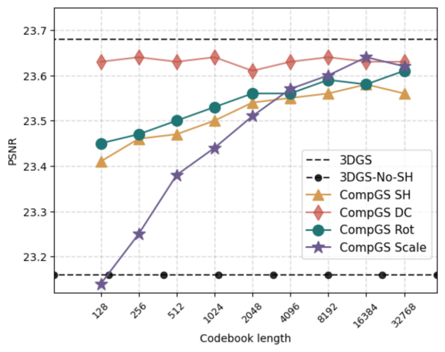


FIGURE 3.4. **Effect of codebook length.** We vary codebook length independently for the four (SH, DC, Rotation, Scale) parameters by quantizing just one of them.

Dataset	Mip-NeRF360		
Method	SSIM↑	PSNR↑	LPIPS↓
3DGS	0.815	27.21	0.214
3DGS *	0.813	27.42	0.217
CompGS 4k	0.804	26.97	0.234
CompGS Shared Codebook	0.797	26.64	0.242

FIGURE 3.5. **Effect of shared codebook.** A frozen codebook trained on one scene ('Counter' scene) generalizes well to all other scenes in MipNerf-360 dataset. Only code assignments are learnt during training.

Effect of codebook size: Fig. 3.4 shows the effect of codebook size for quantization of different Gaussian parameters on the Tanks&Temples dataset. The DC component of color has the smallest drop in performance upon quantization and achieves results similar to the non-quantized version with as few as 128 cluster centers. The harmonics (SH) components of color lead to a much bigger drop at lower number of clusters and improve as more clusters are added. Note that CompGS with only SH components is nearly the same size as 3DGS-No-SH but has better performance (23.43 for ours vs. 23.14 for 3DGS-No-SH). The covariance parameters (rotation and scale) have a drop in performance at a codebook size of 1024 but improve as the codebook size is increased.

Generalization of codebook across scenes We train our train our method on a single scene ('Counter') of the Mip-NeRF360 dataset. We then freeze the codebook and calculate only assignments for the rest of the eight scenes in the dataset and report the averaged performance metrics over all scenes (Fig. B.3). Interestingly, we observe that the shared codebook generalizes well across all scenes with a small drop in performance compared to learning a codebook for each scene. Sharing learnt codebook can further reduce the memory requirement and can help speed up the training of CompGS. The quality of the codebook can be improved by learning it over multiple scenes.

3.1.5 Conclusion

3D Gaussian Splatting efficiently models 3D radiance fields, outperforming NeRF in learning and rendering efficiency at the cost of increased storage. To reduce storage demands, we apply opacity regularization and K-means-based vector quantization, compressing indices and employing a compact codebook. Our method cuts the storage cost of 3DGS by almost 45×, increases rendering FPS by 2.5× while maintaining image quality across benchmarks.

3.2 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

3.2.1 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 [33] across a variety of tasks, including translation, question-answering, cloze tasks, and reasoning. Similarly, in DINOv2, [228] showcase how a large pre-trained ViT model [70] 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 [45, 67, 135, 281, 335].

Inspired by the low intrinsic dimensionality of over-parameterized networks' optimal parameters [10, 181], [135] 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 [67, 335]. 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 [223], 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 [135], 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 3.6 illustrates the fundamental concept of NOLA.

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

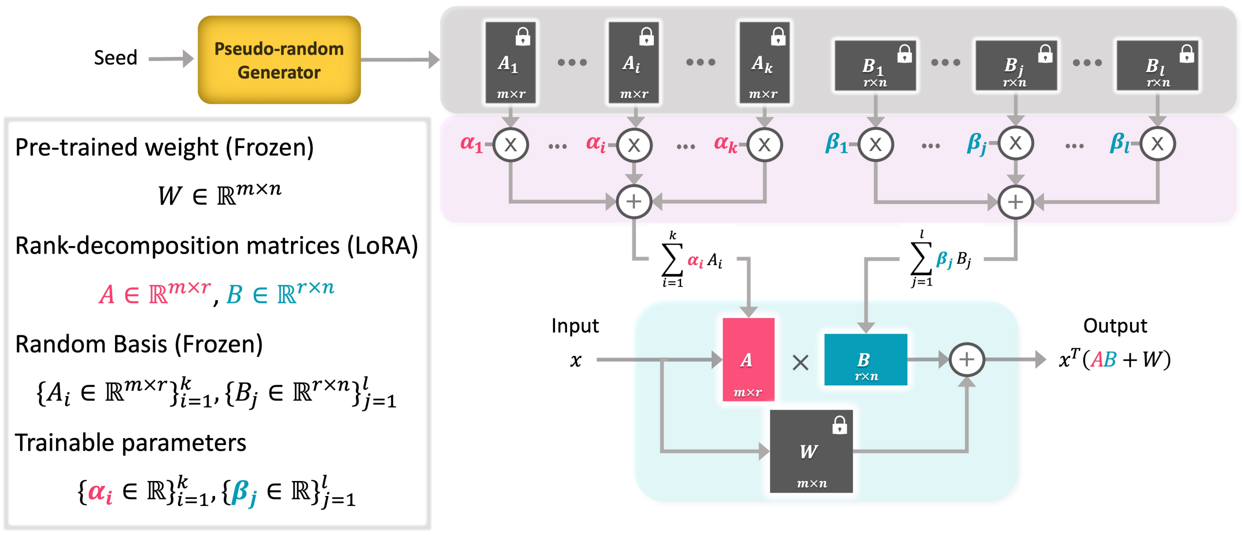

Seed
Pseudo-random Generator
Pre-trained weight (Frozen)
$W \in \mathbb{R}^{m \times n}$
Rank-decomposition matrices (LoRA)
$A \in \mathbb{R}^{m \times r}, B \in \mathbb{R}^{r \times n}$
Random Basis (Frozen)
$\{A_i \in \mathbb{R}^{m \times r}\}_{i=1}^{k}, \{B_j \in \mathbb{R}^{r \times n}\}_{j=1}^{l}$
Trainable parameters
$\{\alpha_i \in \mathbb{R}\}_{i=1}^{k}, \{\beta_j \in \mathbb{R}\}_{j=1}^{l}$
A_1 $m \times r$
A_i $m \times r$
A_k $m \times r$
B_1 $r \times n$
B_j $r \times n$
B_l $r \times n$
α_1 α_i α_k
β_1 β_j β_l
$\sum_{i=1}^{k} \alpha_i A_i$
$\sum_{j=1}^{l} \beta_j B_j$
Input
x
A $m \times r$
B $r \times n$
W $m \times n$
Output
$x^T(AB + W)$

FIGURE 3.6. **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.

versatile toolbox to tackle diverse tasks [266]. 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.

3.2.2 Related Works

Vision and Language Transformer Models: Transformer networks, introduced by [306], emerged as a sequence-to-sequence model in Natural Language Processing (NLP). Their success soon extended to the computer vision community, with subsequent works [71,299] 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 [68] and GPT-2 [245] in NLP, and MAE [117] and DINO [39] 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, [33] show that GPT-3 with 175B parameters is a good few-shot learner. Lastly, the scaling law presented by [148] 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 [68, 245] 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 [133, 191, 201, 251] 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 [347] only adapt bias of the network. Ladder tuning [280] reduce memory footprint in training by avoiding back-propagation through the main backbone. IA3 [192] 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 [111, 179, 186, 194].

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.

[135] 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 [67, 102, 173, 335], optimizing the design choice of LoRA through neural architecture search [353], or dynamically allocating parameters based on the required rank for each weight matrix [351]. 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 [116, 153, 187, 190, 271, 297, 313] and quantization [175, 249] stand as the principal methods for compressing neural networks. Techniques such as those in [145, 172] can achieve a high pruning rate, leading to significant compression.

3.2.3 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 [223] 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 [146, 248] 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 [24] 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 [223] 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 [223], 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.

3.2.4 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.

3.2.4.1 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 [135, 186] for our adaptation setup.

Datasets: We utilize the following datasets for our Natural Language Generation (NLG) task: E2E NLG Challenge [227] 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 [215] 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 [94] 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 [133, 191, 240, 261], and Prefix-layer tuning (PreLayer) [186].

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 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 3.10 and Table B.13 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

TABLE 3.10. **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 [135]. 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	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**

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 3.10 and B.13. 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.

TABLE 3.11. **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

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 3.11. 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 3.12. 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 [135], understanding the effect of rank needs more rigorous study as future work.

3.2.4.2 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 3.13 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.

Table 3.12. **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 3.13. **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

3.2.4.3 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

TABLE 3.14. **Instruction finetuning for quantized LLaMA-2:** NOLA fine-tunes LLaMA-2 70B (8-bit) with 0.57*M* 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

model, so for LLaMA-2 [300] 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 [287], reporting both training and validation loss metrics specific to the Alpaca dataset. Additionally, we employ the MMLU (Massively Multitask Language Understanding) benchmark [126] to assess performance across a diverse set of language understanding tasks. This benchmark comprises 57 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, [67]. 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 3.14. Remarkably, NOLA is capable of fine-tuning LLaMA-2 70B with fewer than 0.6*M* parameters, representing an average reduction of parameters by 95% compared to LoRA with rank one.

3.2.4.4 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 [323]. 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 [169], CIFAR100 [168], CUB-200-2011 [321], Caltech-101 [83], Aircraft [203], Food101 [28], Pets [233] and SUN397 [327] datasets for finetuning.

Baselines: We compare NOLA with three baseline approaches: Linear, Full-FT (full fine-tuning) and LoRA [135]. 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 3.15. 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

TABLE 3.15. **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		CIFAR-100		CUB-200-2011		Caltech-101	
			5	10	5	10	5	10	5	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 3.16. **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		Food101		Pets		SUN397	
		5 Shot	10 Shot	5 Shot	10 Shot	5 Shot	10 Shot	5 Shot	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)

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. We provide results on four additional datasets used for benchmarking transfer learning in

Table 3.16. Aircraft, Food101 and Pets are finegrained 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 3.15, suggesting that it is harder to adapt to these datasets. In line with our prior observations in Table 3.15, NOLA with just one-third the number of parameters performs comparably to LoRA.

3.2.4.5 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 3.7 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.

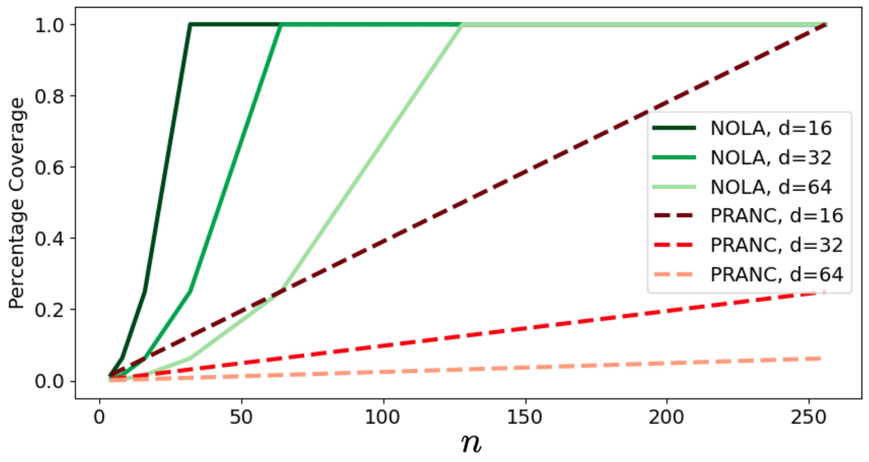


FIGURE 3.7. Comparing the rank of samples in the solution subspace for PRANC and NOLA, given the same number of parameters, n. "Percentage Coverage" is the subspace rank divided by the max possible rank (d^2), so 1.0 denotes full rank. As expected, the coverage for PRANC increases linearly while it saturates very fast for NOLA.

3.2.4.6 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 [223] for our experiments on image classification.

Datasets and Architectures: We consider two architectures in our experiments: ResNet20 with $270K$ parameters, and ResNet18 [122] with $11M$ parameters. We train ResNet20 on CIFAR10 [169], and ResNet18 on ImageNet100 [260].

Results: We report result of ImageNet100 in Table B.11, and CIFAR10 in Table B.10. 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 3.17. **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 3.18. **Training On ImageNet100:** Result of our method on ImageNet-100 dataset and ResNet18

Method	# Params	Acc.
trained model	11,227,812	**82.1%**
HashedNet [50]	129,200	52.96%
PRANC	119,200	61.08%
LoRA	150,000	63.50%
NOLA	109,600	64.66%

3.2.5 Conclusion

We develop NOLA, a parameter efficient fine-tuning method based on LoRA. NOLA uses a linear combination of low-rank random matrices as weights for the fine-tuning task. The combination coefficients are optimized while the random matrices are frozen. Thus, NOLA provides a way to decouple the number of fine-tuning parameters both from the pretrained model architecture and the rank of the low-rank matrices. It performs as well as or even better than LoRA with 95% fewer parameters on LLaMA-2. It is also on par with LoRA but with half to one-third number of parameters using smaller LLMs like GPT2 and on image classification tasks with vision transformers.

CHAPTER 4

Robustness of Efficient Models in Vision Transformers

In the previous chapters, we looked at how to improve the efficiency of models in terms of compute, memory, data and resources. Here, we *attack* models to make them less efficient! We analyse the robustness of a specific type of efficient models - adaptive inference models for image classification - to adversarial attacks on their efficiency. As the size of models is growing rapidly and efficient solutions are introduced to keep pace, more such studies on adversarial attack and defense on efficiency are the need of the hour.

4.1 SlowFormer: Adversarial Attack on Compute and Energy Consumption 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

4.1.1 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,

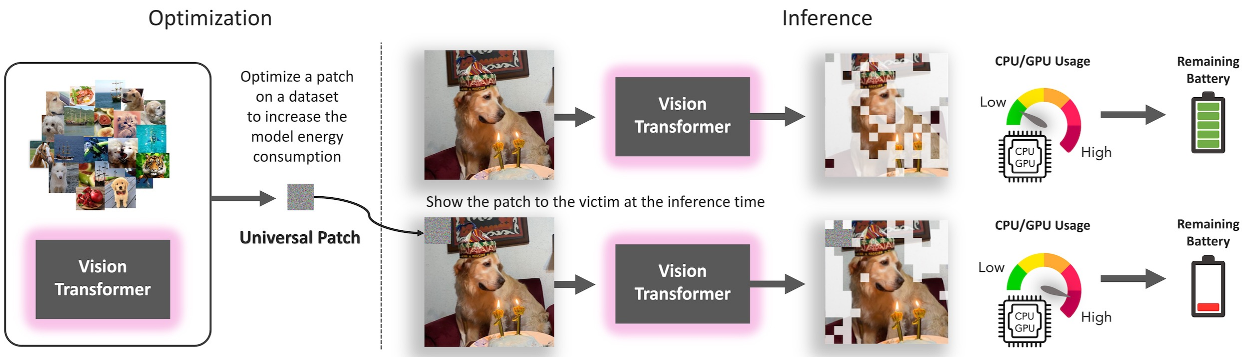

Optimization
Inference
Optimize a patch on a dataset to increase the model energy consumption
Universal Patch
Vision Transformer
Show the patch to the victim at the inference time
CPU/GPU Usage
Low
High
CPU GPU
Remaining Battery

FIGURE 4.1. **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.

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 [2] (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 [8], 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 [264].

We design our attack, SlowFormer, for three different methods (A-ViT [338], ATS [81], and Ada-VIT [207]) 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 4.1 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 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.

4.1.2 Related Work

Vision Transformers: The popularity of transformers [306] in vision has grown rapidly since the introduction of the first vision transformer [70, 298]. Recent works demonstrate the strength of vision transformers on a variety of computer vision tasks [56,70,195,247,298,342,355,356,358]. Moreover, transformers are the backbone of recent Self-Supervised Learning (SSL) models [39,117], and vision-language models [244]. 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 [31, 150, 341]. To accomplish this, some lines of work study token pruning with the goal of removing uninformative tokens in each layer [81, 206, 207, 246, 338]. ToMe [27] merges similar tokens in each layer to decrease the computation. Some works address quadratic computation of self-attention module by introducing linear attention [12, 149, 158, 198, 270]. Efficient architectures [131, 196] 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 [81, 207, 338].

Dynamic Computation: There are different approaches to reducing the computation of vision models, including knowledge distillation to lighter network [130, 199], model quantization [193, 248] and model pruning [182]. 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 [26, 74, 86, 104, 109, 139, 289, 307, 316] and selectively activate neurons, channels or branches for dynamic width [23, 35, 42, 82, 92, 129, 137, 343] 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 [357], encouraging a new line of work to make CNNs more efficient through spatially dynamic computation. Pixel-Wise dynamic architectures [36, 43, 77, 156, 254, 308, 332] 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 [89, 188]. 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 [134]. Correspondingly, [336, 337] 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 [80, 207, 338]. Fayyaz et al. [80] 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 [338] reduces the number of tokens in vision transformers by discarding redundant spatial tokens. Meng et al. [207] 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 [103, 171, 283]. 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 [85, 185, 230, 329]. Patch-Fool [90] considers adversarial patch-based attacks on transformers. Some recent works [202, 319, 350] 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 [114], Haque et al. attack two CNN-based input-dynamic methods: SkipNet [316] and SACT [86] using image specific perturbation. DeepSloth [132] attack focuses on slowing down early-exit methods, reducing their energy efficiency by 90-100%. GradAuto [229] successfully attacks methods that are both dynamic width and dynamic depth. NICGSlowDown and TransSlowDown [46, 47] 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 [44] and NodeAttack [115] 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 [114], DeepSloth [132] and GradAuto [229] 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 [90].

4.1.3 Computation and Energy Attack

4.1.3.1 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 [32, 262] 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.

4.1.3.2 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 [338] 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:

$$L_{\text{ponder}} = \frac{1}{K}\sum_{k=1}^{K}(N_k + r_k), \qquad r_k = 1 - \sum_{l=1}^{N_{k-1}} h_k^l \tag{4.1}$$

Additionally, A-ViT enforces a Gaussian prior on the expected halting scores of all tokens via KL-divergence based distribution loss, $L_{distr.}$. These loss terms are minimized along with the task-specific loss L_{task}. Thus, the overall training objective is $L = L_{task} + \alpha_d L_{distr.} + \alpha_p L_{ponder}$ 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 $-(\alpha_d L_{distr.} + \alpha_p L_{ponder})$ as our loss. It is possible to preserve (or hurt) the task performance by additionally using $+L_{task}$ (or $-L_{task}$) in the loss formulation.

Attacking AdaViT:

Background: To improve the inference efficiency of vision transformers, AdaViT [207] 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.

$$(m_l^p, m_l^h, m_l^b) = (W_l^p, W_l^h, W_l^b)Z_l \tag{4.2}$$

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 [200] 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:

$$L_{usage} = \left(\frac{1}{D_p}\sum_{d=1}^{D_p} M_d^p - \gamma_p\right)^2 + \left(\frac{1}{D_h}\sum_{d=1}^{D_h} M_d^h - \gamma_h\right)^2 + \left(\frac{1}{D_b}\sum_{d=1}^{D_b} M_d^b - \gamma_b\right)^2 \tag{4.3}$$

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. 4.3. 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 [81] assigns importance score S_j for each token j by measuring how much the classification token attends to it:

$$S_j = \frac{A_{1,j} \times ||V_j||}{\sum_{i=2} A_{1,i} \times ||V_i||} \tag{4.4}$$

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:

$$L = \sum_{i=2}^{N} ||A_{1,i} - \frac{1}{N}||_2^2 \tag{4.5}$$

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.

4.1.4 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.

4.1.5 Experiments

4.1.5.1 Attack on Efficient Vision Transformers

Dataset: We evaluate the effectiveness of our attack on two datasets: ImageNet-1K [65] and CIFAR-10 [166]. 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. $AttackSuccess = \frac{(FLOPs_{attack} - FLOPs_{min})}{(FLOPs_{max} - FLOPs_{min})}$ where $FLOPs_{min}$ is the compute of the efficient model and $FLOPs_{max}$ 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

TABLE 4.1. **Computation and Energy Attack on Efficient ViTs:** 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%.

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 4.2. **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.

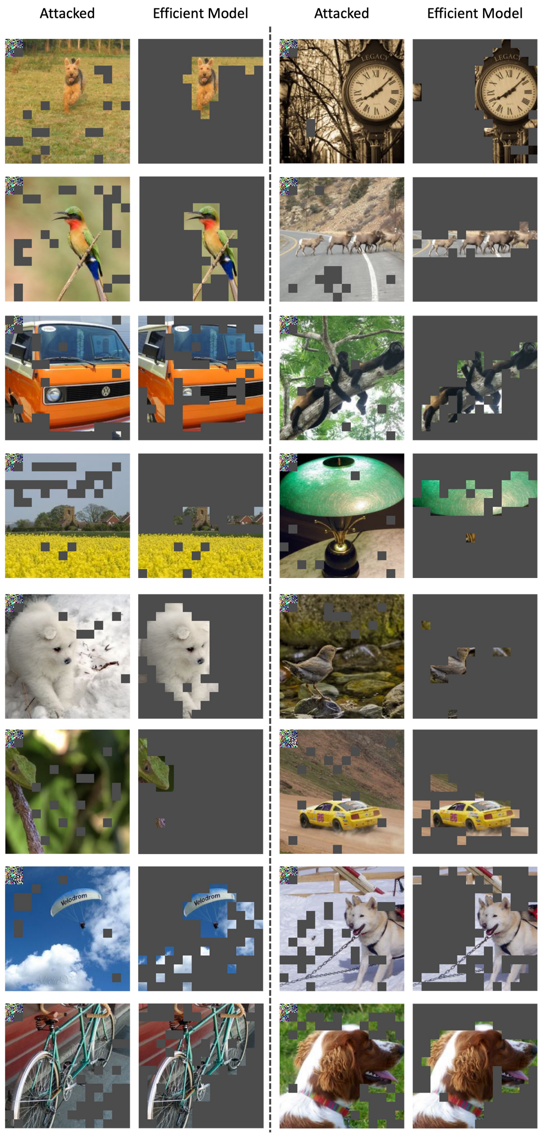


TABLE 4.2. **Results on CIFAR10 dataset.** We report results on CIFAR10 dataset to show that our attack is not specific to ImageNet alone. CIFAR-10 is a small dataset compared to ImageNet and thus results in an extremely efficient A-ViT model. Our attack increases the FLOPs from 0.11 to 0.58 which restores 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 4.3. **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%

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 [236] 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 [197] 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.

TABLE 4.4. **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%

ATS Details: As in ATS [81], 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 [338], the patches are optimized for one epoch with a learning rate of 0.2 and a batch size of 512 (128 × 4GPUs) using AdamW [197] 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 [207], 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 [197] 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 4.5. **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**	-	-
A-ViT	-	No attack	0.87	71.4%	-
	16	SlowFormer	1.15	6.1%	73%
	32	SlowFormer	1.25	0.5%	98.4%
ATS	-	No attack	0.84	70.3%	-
	16	SlowFormer	0.98	15.6%	30.4%
	32	SlowFormer	1.04	0.8%	43.5%
		ViT-Small	**4.6**	-	-
A-ViT	-	No attack	3.7	78.8%	-
	16	SlowFormer	4.48	20%	86.4%
	32	SlowFormer	4.59	1%	98.1%
ATS	-	No attack	3.1	79.2%	-
	16	SlowFormer	3.6	31.0%	33.3%
	32	SlowFormer	3.8	3.6%	46.7%
AdaVit	-	No attack	2.25	77.3%	-
	16	SlowFormer	3.0	26.1%	31.9%
	32	SlowFormer	3.2	2.8%	40.4%

Results. The results of our attack, SlowFormer , on various methods on ImageNet dataset are shown in table 4.1. 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 B.10. 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 C.1. 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. 4.3, we visualize the optimized patches for each of the three efficient methods.

4.1.5.2 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 4.1, 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 either preserving or destroying the task performance by additionally employing a task loss in the patch optimization. Table 4.3 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 4.4. 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 4.5). In perturbation attacks, all pixels in the image can be modified, but with an upper bound on the ℓ_∞ norm of the perturbation.

4.1.5.3 Adversarial training based defense

Our simple defense that is adopted from standard adversarial training is explained in Section 4.1.4. The results for defending against attacking A-ViT are shown in Table 4.6. The original A-ViT reduces the

TABLE 4.6. **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	1.01	65.8%	34%

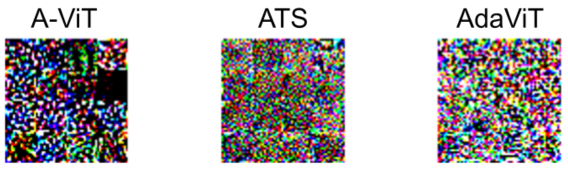


FIGURE 4.3. **Visualization of optimized patch:** We show the learned universal patches for each of the three efficient methods.

GFLOPs from 1.26 to 0.87, our attack increases it back to 1.26 with 100% attack success. The proposed 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.

4.1.6 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 5

Discussion

In this work, I have tried to develop compute and memory efficient solutions to diverse computer vision tasks. The developed solutions lead to reduced training and inference times, storage memory and hardware requirements, making them easier and more accessible to use. In self-supervised representation learning (SSL) from images, we proposed a novel training algorithm that requires lesser computation per training iteration while also converging faster. Knowledge from such SSL models can also be distilled to smaller models using our simple regression based technique proposed in SimReg. On a more specific setting of Gaussian splatting for novel view synthesis, we show that it is possible to vastly compress the models without greatly affecting their performance. This significantly reduces the bottleneck in widespread adoption of Gaussian splat models. Keeping pace with the advances in foundation models, our work on parameter efficient fine-tuning (PEFT) also reduces the memory requirement for task-specific parameters. This work will potentially have a bigger impact as the market for custom solutions using large vision and language models rapidly increases in the near future. Seemingly contrary to these directions, our work analysing the robustness of these models aims to attack their efficiency. We find that adaptive inference methods, a class of compute efficient approaches, are susceptible to adversarial attacks, highlighting the need for more such studies and development of defenses against such attacks.

Several of the above mentioned works can be extended or adapted to novel settings and tasks. Here are some potential avenues for future research along these directions.

Distillation for 3DGS model compression: We looked at compression of 3DGS models in our work CompGS. CompGS focused on training a compressed version of 3DGS from scratch. It is possible to pretrain a non-compressed version first and incorporate its knowledge in training the compressed one. This is aligned with the ideas of knowledge distillation from a stronger teacher model to a weaker student model. The novel view synthesis task is especially suited for such distillation since the scene remains the same between the train and test settings. Novel view points in addition to the training data can easily be generated by sampling the view points and rendering the teacher model from the corresponding views.

Continual learning for vision models: Continual learning (CL) involves iteratively improving a model as and when new data is obtained. It could also be seen as a variant of fine-tuning, albeit with the assumption that the initially observed data is no longer available. Thus, methods and ideas from the rich fine-tuning literature could be borrowed and adapted to work effectively in the continual learning setting. Specifically, similar to our work on fine-tuning, it is interesting to look at parameter efficient solutions for CL. Traditional solutions have relied on a single common network as more data and/or tasks are added. However, PEFT can be a simple and clever alternative to limit the growth in model size but improve performance. With the use of task-specific parameters, it could also be trivially easy to overcome the negative backward effects in CL where training on newer data / tasks results in catastrophic forgetting on the earlier tasks.

Compute adversarial attack on LLMs and VLMs: In SlowFormer, we adversarially attacked the efficiency of vision transformer models. It is possible that a similar attack can be used to render large language models (LLMs) and vision language models (VLMs) inefficient. In vision language models, the input image could be attacked similar to SlowFormer, with the additional aim of slowing down the language component. While adaptive input methods are not as prevalent in language models as compared to vision, there are several potential ways to decrease the model's inference efficiency. In language generation tasks, the attack desideratum could be to increase the length of the generated text. Recent models like OpenAI-o1 include an additional step of thinking about the initial answer and modifying it to provide the final output. Such models could be forced to think longer.

APPENDIX A

Compute Efficiency in Computer Vision in Training and Inference

A.1 Constrained Mean Shift for Representation Learning

Here, we provide additional results and analysis on self-supervised (section A.1.1), cross modal constraint (section A.1.2), semi-supervised (section A.1.3) and supervised (section A.1.4) settings. Additional results include analysis of retrieved far neighbors (Figs. A.1 and A.2) and ablations to justify various design choices (tables A.5, A.6, A.7, A.8, A.10, A.12, A.13). More details on implementation (section A.1.5) and compute calculation (Eq. A.1, table A.11) are provided. We also provide the code as part of the supplementary materials.

A.1.1 Results on Self-supervised Setting

A.1.1.1 Using far neighbors in unconstrained MSF:

In 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 [162] method. Here we train MSF with $k = 500$ to compare with our CMSF$_{\text{self}}$. We train all models for 80 epochs. Settings are similar to our self-supervised settings in the main text. For fair comparison, we use memory-bank of size $256k$ for MSF while we use $128K$ for CMSF$_{\text{self}}$. Results are in Table **??**. 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. 5 of the main submission). This experiment shows that it is not trivial to to sample far NNs with good purity.

TABLE A.1. **Using far neighbors in unconstrained MSF [162]:** 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, CMSF$_{\text{self}}$ achieves high accuracy while using distant NNs.

	MSF [162] Top-k = 500	MSF [162] Top-k = 5	CMSF$_{\text{self}}$ Top-$k = k' = 5$
NN	35.8	49.7	51.4
20-NN	40.2	54.0	55.5

A.1.1.2 Effect of number of neighbors k

CMSF$_{\text{self}}$ uses top-k NNs as part of loss calculation. Here we study the effect of k in CMSF$_{\text{self}}$ performance. We set $k' = k$ in all models. Note that as described in Line 350 of the main submission, 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 CMSF$_{\text{self}}$ in main text. Results are shown in Table ??. Higher values of k and k' degrade model performance. We thus use $k' = k = 5$ in all our main experiments.

TABLE A.2. **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

we experimented with ResNet-18 and ResNet-101 networks. ResNet-101 is trained for 100 epochs. CMSF outperforms MSF by 1.9 points (51.7% vs 49.8%) with ResNet-18 and 0.8% points (71.9% vs 71.1%) with ResNet-101.

A.1.1.3 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.3. The proposed CMSF$_{\text{self}}$ improves over MSF across different architectures. Note that the networks are trained only for 100 epochs on ResNet-101.

A.1.2 Cross-modal Constraint

Fig. 6 of main submission showed that proposed CMSF$_{\text{sup}}$ 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

TABLE A.3. **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 [162]	49.8	72.2	71.1
CMSF$_{\text{self}}$	51.7	73.0	71.9

two SSL models on the RGB and Flow modalities separately using InfoNCE method [113, 303]. 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 [113], we use split-1 of UCF-101 [277] (13k videos) as the unlabeled dataset. We use similar augmentation and pre-processing as [113] and calculate optical-flow using unsupervised TV-L1 [345] algorithm. For cross-modal experiments, we use S3D [331] architecture with the input size of 128×128 pixels. We initialize from the pretrained weights of InfoNCE (400-epoch) released by [113]. 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 [113] for linear evaluation.

Results: The results are shown in Table A.4. 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 [113] baseline and MSF with 2 stages.

A.1.3 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.1.3.1 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.1, 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.1 shows that the accuracy of the classifier on the confident samples remains high throughout training, limiting the number of incorrect

TABLE A.4. **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 [113] also uses another modality as a constraint in the form of contrastive learning. We continue training InfoNCE SSL model for 200 epochs using MSF [162] for a fair comparison. We use S3D [331] 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
$\text{InforNCE}_{Flow}*$	Flow	400	45.3	66.1
$\text{InforNCE}_{Flow} \rightarrow \text{MSF}_{Flow}$	Flow	400+200	47.3	64.7
$\text{InforNCE}_{Flow}*$	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
$\text{InforNCE}_{Flow}*$	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

pseudo-labels. Results for threshold value (t) selection are shown in table A.5. 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.

Table A.5. **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**

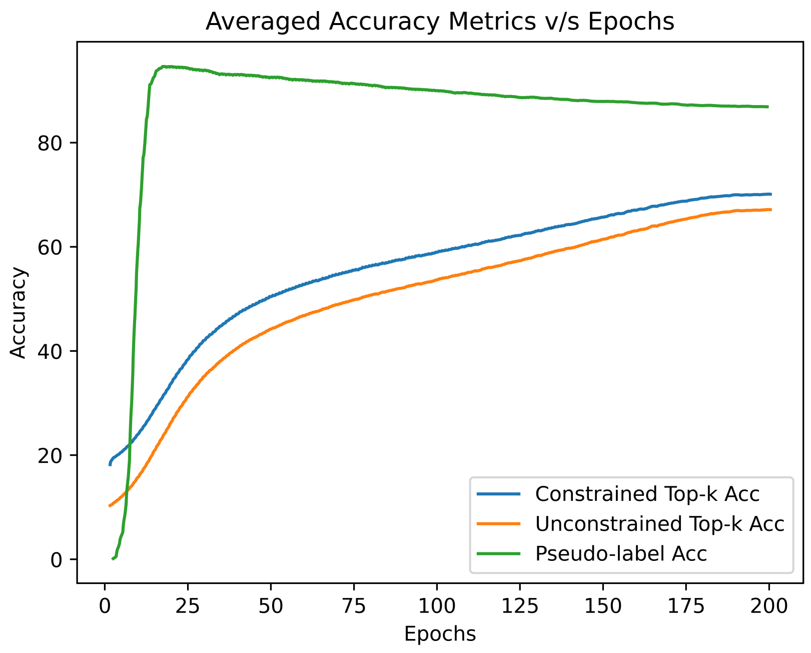


Figure A.1. **Unconstrained NN accuracy in semi-supervised:** For analysis, we track the pseudo-labeling accuracy and accuracy of top-k neighbors chosen with and w/o applying the pseudo-label based constraint during training. The more accurate constrained NNs provide a better training signal. Pseudo-label accuracy on confident samples remains high throughout training, decreasing slightly as more confident samples are added.

A.1.3.2 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

TABLE A.6. **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

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 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.6). 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.1.3.3 Pseudo-label Classifier Selection

The classifier used to generate pseudo-labels plays a crucial role in obtaining effective constraint sets for 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.7. 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.7, 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.7. **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.8. **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 the 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.1.3.4 Fine-tuning without Pseudo-labels

Since we do not explicitly optimizer 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.8, 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.1.4 Results on Supervised Setting

A.1.4.1 Coarse-grained ImageNet

CMSF$_{\text{sup}}$ 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

TABLE A.9. **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$_{\text{sup}}$ top-10 outperforms all other variants and baselines.

Train Dataset	ImageNet-1k Validation Set				
	Xent	SupCon	CMSF$_{\text{sup}}$ top-*all*	CMSF$_{\text{sup}}$ top-1000	CMSF$_{\text{sup}}$ top-10
ImageNet-1k	77.2	**77.5**	75.7	-	76.4
ImageNet-coarse	61.4	58.7	67.0	71.0	**74.2**

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 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$_{\text{sup}}$ in Sec.3.2 of main submission.

In Table A.9 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$_{\text{sup}}$ 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$_{\text{sup}}$ top-*all* and top-1000, cross-entropy and supervised contrastive - see a huge drop in accuracy.

A.1.4.2 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.10. Training and evaluation details are the same as in the main submission.

A.1.5 Implementation Details

A.1.5.1 Transfer Learning

We use the LBFGS optimizer (max_iter=20, and history_size=10) along with the Optuna library [11] in the Ray hyperparameter tuning framework [189]. 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) 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 [107] and SimCLR [48] with our framework. The features are

Table A.10. **Ablations of baselines and CMSF$_{\mathbf{sup}}$:** 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 [9] (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

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.1.5.2 Supervised Setting

Implementation Details of Baselines

TABLE A.11. **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-	Iters	Epochs	Total	FLOPs
	Fwd	Bwd	BS	Fwd	Bwd	BS	Batch	per epoch		Pass ($\times 10^8$)	($\times 10^{18}$)
Mean Shift [162]	2	1	256				768	5004	200	7.7	4
BYOL [107]	4	2	4096				24576	312	1000	76.7	40
SwAV [38]	3.1	3.1	4096				25395	312	800	63.4	37
SimCLRv2 [49]	2	2	4096				16384	312	800	40.9	16
UDA† [330]	2	1	15360	1	1	512	47104	40000		18.8	10
FixMatch† [276]	2	1	5120	1	1	1024	17408	250	300	13.1	70
MPL† [241]	3	2	2048	2	2	128	10752	500000		53.8	30
PAWS (sup=6720) [14]	3.1	3.1	4096	1	1	6720	38835	312	300	36.6	21
PAWS (sup=1680) [14]	3.1	3.1	256	1	1	1680	4947	5004	100	24.8	15
PAWS (sup=400) [14]	3.1	3.1	256	1	1	400	2387	5004	100	12.0	7
CMSF$_{\text{semi}}$-basic	2	1	256				768	5004	200	7.7	4
CMSF$_{\text{semi}}$	2	1	256				768	5004	200	7.7	4
CMSF$_{\text{semi}}$-mix prec.	2	1	768				2304	1668	200	7.7	4

TABLE A.12. **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$_{\text{sup}}$ 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$_{\text{sup}}$ 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$_{\text{sup}}$ 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$_{\text{sup}}$ 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$_{\text{sup}}$ 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$_{\text{sup}}$ 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$_{\text{sup}}$ 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$_{\text{sup}}$ 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$_{\text{sup}}$ 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$_{\text{sup}}$ 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

SupCon: The MLP architecture for SupCon baseline is: linear (2048x2048), batch norm, ReLU, and linear (2048x128). To optimize the SupCon baseline, following [152], we use the first 10 epochs for learning-rate warmup. For both SupCon and ProtoNW, the temperature is 0.1.

TABLE A.13. **Transfer dataset details:** Train, val, and test splits of the transfer datasets are listed in this table. **Test split:** We follow the details in [162]. 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 [107], 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 [28]	101	68175	7575	25250	Top-1	-
CIFAR-10 [166]	10	49500	500	10000	Top-1	-
CIFAR-100 [166]	100	45000	5000	10000	Top-1	-
Sun397 (split 1) [327]	397	15880	3970	19850	Top-1	-
Cars [163]	196	6509	1635	8041	Top-1	-
Aircraft [203]	100	5367	1300	3333	Mean	Yes
DTD (split 1) [59]	47	1880	1880	1880	Top-1	Yes
Pets [233]	37	2940	740	3669	Mean	-
Caltech-101 [83]	101	2550	510	6084	Mean	-
Flowers [221]	102	1020	1020	6149	Mean	Yes

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 [274], 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.

A.1.5.3 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. 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.11) 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 [121] 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 [136]. 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.

$$\begin{aligned}
\text{Fwd mini-batch} &= (\text{Unlabeled fwd crops} * \text{Unlabeled batch-size}) \\
&\quad + (\text{Labeled fwd crops} * \text{Labeled batch-size}) \\
\text{Bwd mini-batch} &= (\text{Unlabeled bwd crops} * \text{Unlabeled batch-size}) \\
&\quad + (\text{Labeled bwd crops} * \text{Labeled batch-size}) \\
\text{Fwd passes} &= \text{Fwd mini-batch} * \text{Iterations per epoch} * \text{Epochs} \\
\text{Bwd passes} &= \text{Bwd mini-batch} * \text{Iterations per epoch} * \text{Epochs} \\
\text{Total FLOPs} &= (\text{Fwd passes} + 2 * \text{Bwd passes}) * (3.9 \times 10^9)
\end{aligned} \tag{A.1}$$

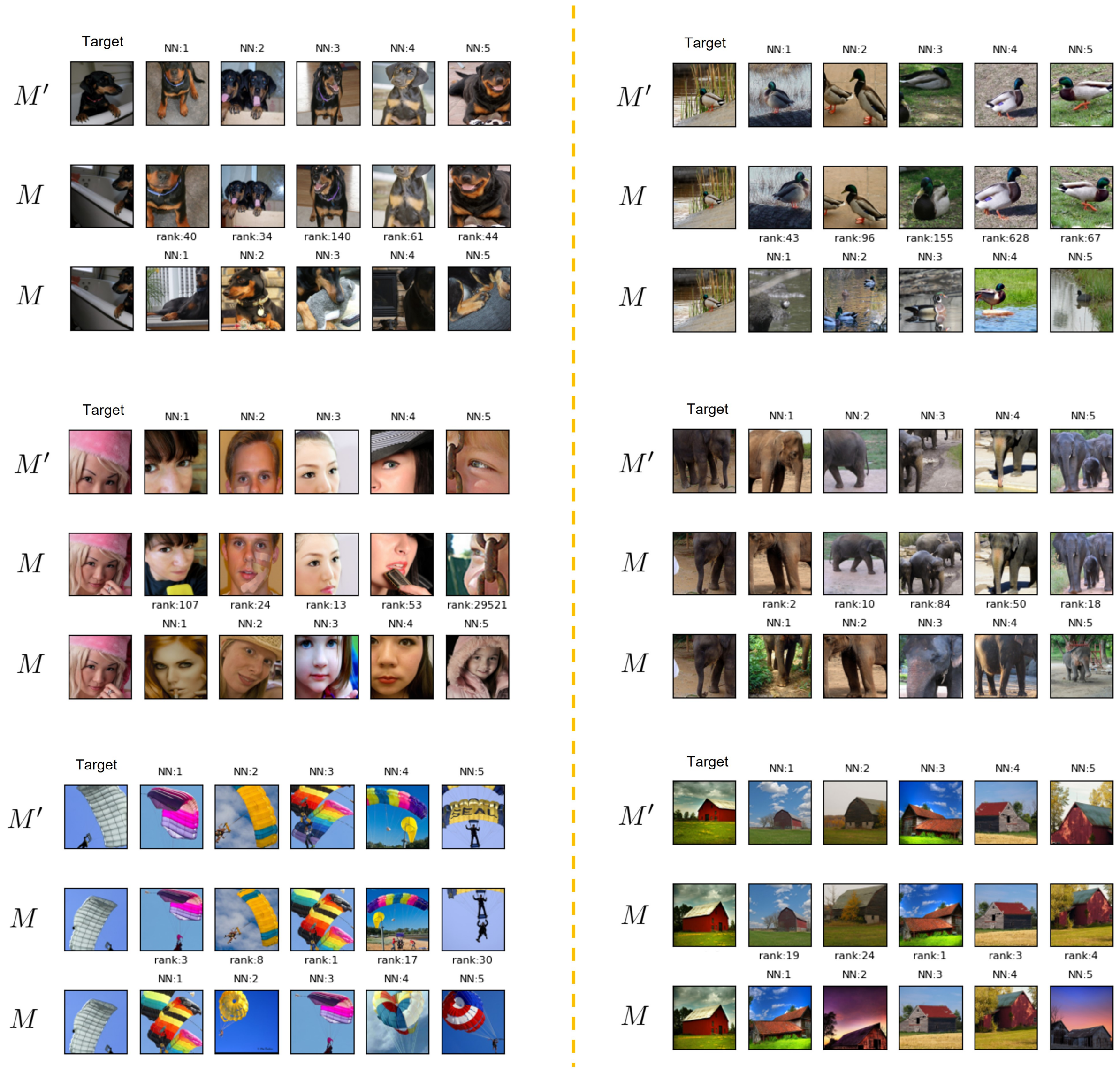


FIGURE A.2. **CMSF$_{\text{self}}$ nearest neighbor selection:** We use epoch 100 of CMSF$_{\text{self}}$ to visualize Top-5 NN from primary (M) and auxiliary (M') memory banks. M stores features for the current epoch while M' contains representations from a different augmentation of the same image instance 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.2 SimReg: Regression as a Simple Yet Effective Tool for Self-supervised Knowledge Distillation

In this supplementary material, we present additional experimental results (Sec. A.2.1) and details on experiment settings and implementation (Sec. A.2.2). Additional results include those on the role of MLP head during training (Sec. A.2.1.1) and self-distillation (Sec. A.2.1.2). We publicly release the code[*].

A.2.1 Additional Experimental Results

A.2.1.1 Role of MLP Head

In tables 1 and 2 of main we analyze how the depth of MLP head during training and inference affects classification performance. We present additional results here in table A.14 with different teacher and student network settings. The student networks are trained with different prediction head configurations. The evaluation is always performed using features from backbone network for a fair comparison. In addition to the self-supervised (SSL) teacher models used in the main paper, we consider a supervised teacher network. The teacher is trained with cross-entropy loss using ground truth labels on the ImageNet dataset. As in SSL teachers, we use only the backbone network for distillation from a supervised teacher. Note that the supervised labels are absent during student training. In both the supervised and self-supervised settings, the student with 4 layer MLP head consistently outperforms others on all metrics. **Compared to Linear head, 4L-MLP achieves 5 (MoCo-v2, ResNet-18), 11.2 (MoCo-v2, MobileNet-v2) and 6 (Supervised, MobileNet-v2) percentage points improvement on linear evaluation**.

A.2.1.2 Self-distillation

In all the previous experiments, a larger teacher network is distilled to a shallower student. In self-distillation, we consider the same backbone architecture for both teacher and student. Similar to other experiments, we use a prediction head (linear or MLP) atop student backbone during distillation and remove it during evaluation. As we observe in table A.15, the student with a 4 layer MLP head outperforms the teacher in both ImageNet classification and transfer tasks. The improvement in transfer performance is particularly significant (+4 percentage points) and might be attributed to the use of prediction head and weaker augmentations during distillation.

[*]Code is available at `https://github.com/UCDvision/simreg`

Teacher	Student Arch (Inference)	Prediction Head (Train)	1-NN	20-NN	Linear
MoCo-v2 ResNet-50	ResNet-18 Backbone	4L-MLP	**54.8**	**59.9**	**65.1**
		2L-MLP	52.7	58.5	63.6
		Linear	48.8	54.3	60.1
MoCo-v2 ResNet-50	MobileNet-v2 Backbone	4L-MLP	**55.46**	**59.73**	**69.1**
		2L-MLP	54.4	59.6	68.5
		Linear	48.7	54.2	57.9
Supervised ResNet-50	MobileNet-v2 Backbone	4L-MLP	63.77	67.87	**73.5**
		2L-MLP	**64.7**	**69.3**	**73.5**
		Linear	55.4	62.0	67.5

TABLE A.14. **Effect of MLP Heads on ImageNet classification performance.** As in table 1 of main paper, we analyze the role of the prediction head used during training by varying the number of MLP layers. However, the evaluation here is performed using the features from the backbone network and the prediction head plays no role during inference. A linear prediction head corresponds to the architecture used in earlier works [161]. We observe that a deeper prediction module during training results in substantial boosts in performance. This observation is consistent across different teacher networks (both SSl and supervised) and student architectures. **Compared to Linear head, 4L-MLP achieves 5(MoCo-v2, ResNet-18), 11.2(MoCo-v2, MobileNet-v2) and 6(Supervised, MobileNet-v2) percentage points improvement on linear evaluation**.

Student Arch	Prediction Head	ImageNet			Transfer
(Inference)	(Train)	1-NN	20-NN	Linear	Linear
MoCo-v2 Teacher	-	57.3	60.9	70.8	74.3
ResNet-50	4L-MLP	**58.2**	**62.2**	**72.0**	**78.3**
ResNet-50	Linear	56.4	60.6	69.7	71.8

TABLE A.15. **ImageNet Classification and transfer results for self-distillation with prediction head.** In self-distillation, the teacher and student backbone architectures are the same (ResNet-50). We use a MoCo-v2 pretrained teacher and train student networks with linear and 4 layer MLP heads. All evaluations are performed using backbone features. The student with MLP prediction head outperforms the teacher on both ImageNet classification and transfer tasks. A boost of 4 percentage points on average transfer accuracy suggests that the use of prediction head and weaker augmentations during distillation are beneficial in learning a good generalizable model.

A.2.1.3 Comparison with CompRess without MLP

In table 1 of the main paper, we observed that the use of MLP head during distillation benefits both the CompRess variants on the ImageNet classification task. Here, we show that similar boost in CompRess performance can be achieved on transfer tasks when distilled with MLP head. We use the officially provided pretrained models for vanilla CompRess-2q ResNet-18 and MobileNet-v2 architectures for our comparison and perform transfer analysis similar to that in table 5 of the main paper. Results in table A.16 demonstrate

Arch	MobileNet-v2			ResNet-18		
Method	CompRess-2q plain	CompRess-2q -4L-MLP	SimReg -4L-MLP	CompRess-2q plain	CompRess-2q -4L-MLP	SimReg -4L-MLP
Food	61.4	71.4	**73.1**	57.6	61.7	**65.4**
CIFAR10	85.3	90.3	**91.2**	82.5	87.3	**88.6**
CIFAR100	65.1	73.9	**76.1**	62.5	68.4	**70.2**
SUN	53.9	58.0	**59.4**	52.2	54.3	**57.1**
Cars	35.0	60.3	**62.4**	30.0	37.2	**42.3**
Aircraft	42.1	57.7	**58.7**	38.0	42.3	**45.8**
DTD	70.4	71.7	**74.5**	67.4	69.3	**70.9**
Pets	82.9	**86.7**	85.6	81.6	**84.0**	83.9
Caltech	85.6	91.1	**91.7**	85.3	87.3	**89.2**
Flowers	87.6	94.3	**95.1**	83.0	86.4	**90.9**

TABLE A.16. **Transfer learning performance of CompRess with and without MLP.** Since the teacher networks are self-supervised, generalization of learnt features to other datasets is important. Similar to ImageNet classification, CompRess with MLP significantly outperforms vanilla CompRess (CompRess-2q plain) on all datasets and metrics. MLP heads, if present, are only used during distillation and are not part of inference networks.

that performance of vanilla CompRess models are significantly worse compared to both CompRess with MLP and proposed regression based distillation. Note that the MLP heads are not used during inference for fair comparison.

A.2.1.4 Results with Intermediate Layers of CNN

In our results in table 2 of main paper, we analyze how the classification performance changes as we consider features from the earlier layers of the prediction head. Here, we analyze results from various intermediate layers including those from the CNN backbone. We train a single ResNet-18 student from a MoCo-v2 ResNet-50 teacher and perform k-NN evaluation using features from different layers. In table A.17, Conv-1 refers to the output of the first convolutional layer while ResBlk-j refers to the output from the j^{th} residual block. The CNN features for evaluation are obtained by reducing their spatial dimension and then vectorizing. The spatial dimensions are reduced so that the feature lengths are roughly the same throughout the backbone for fair comparison. We observe that the performance increases as we go deeper into the backbone. The best performance is achieved at the intermediate layer of prediction head and there is a small drop in accuracy at the final prediction layer.

Eval Layer	Conv-1		ResBlk-1		ResBlk-2		ResBlk-3		ResBlk-4		2L-MLP		4L-MLP	
Student	1-NN	20-NN	1-NN	20-NN	1-NN	20-NN	1-NN	20-NN	1-NN	20-NN	1-NN	20-NN	1-NN	20-NN
ResNet-18	6.2	7.2	16.0	18.0	21.8	24.3	33.4	37.4	55.3	60.2	56.0	60.9	53.4	57.6

TABLE A.17. **ImageNet classification using intermediate features**. We consider a single student network with 4 layer MLP head and perform k-NN evaluation using features from various intermediate layer features from the network. For fair comparison, we match the dimensions from the intermediate convolutional features (Conv-1 and residual block features) to that of the final backbone feature (ResBlk-4) by reducing their spatial dimensions. As expected, performance improves as we use features from deeper layers of the CNN. This changes in the MLP head where a drop in accuracy at the very last layer of the prediction head is observed.

A.2.2 Implementation Details

A.2.2.1 Teacher Networks

We use teacher networks trained using four different self-supervised representation learning approaches - MoCo-v2 [52], BYOL [107], SwAV [37] and SimCLR [48]. We use the official and publicly available pre-trained weights for these networks with ResNet-50x4 architecture pretrained model for SimCLR teacher and ResNet-50 models for the remaining methods. MoCo-v2 and SwAV have been trained for 800 epochs and BYOL and SimCLR for 1000 epochs. For distillation with BYOL, SwAV and SimCLR teachers we use cached features from the teacher. The cached features are obtained by passing the entire training data through the teacher network once and storing the features. Random image augmentation as would be used in non-cached version is employed to generate the inputs for caching.

A.2.2.2 Image Augmentations

We use two strategies for augmenting the input image during distillation - ‘weak’ and ‘strong’. ‘Strong’ augmentation refers to the setting used in MoCo-v2 [52]. In both augmentation settings, we apply a series of stochastic transformations on the input image. A random resized crop (scale is in range [0.2, 1.]), random horizontal flip with probability 0.5 and normalization to channel-wise zero mean and unit variance are common for both augmentation methods. In addition to these transformations, ‘strong’ augmentations use random color jittering (strength of 0.4 for brightness, contrast and saturation and 0.1 for hue) with probability 0.8, random grayscaling with probability 0.2 and Gaussian blur (standard deviation chosen uniformly from [0, 1]).

A.2.2.3 Optimizer

In all our distillation experiments, we use SGD optimizer with cosine scheduling of learning rate, momentum of 0.9 and weight decay of 0.0001. Initial learning rate is set to 0.05. The networks are trained for 130 epochs with a batch size of 256 using PyTorch [235] framework.

A.2.2.4 Evaluation Metrics

We utilize k-NN and linear evaluation to evaluate classification performance on ImageNet and linear evaluation to evaluate transfer performance. For ImageNet linear evaluation, the inputs to the linear layer are normalized to unit l_2 norm and then each dimension is shifted and scaled to have unit mean and zero variance [161]. The layer is trained for 40 epochs using SGD with initial learning rate of 0.01 and momentum of 0.9. The learning rate is scaled by 0.1 at epochs 15 and 30. For evaluation of transfer performance, we use the optimizer settings from [162]. The shorter side of the input image is resized to 256 and centre crop with length 224 is used. The input is channel-wise normalized using the statistics from ImageNet dataset. We use LBFGS optimizer with parameters max_iter=20 and history_size=10. Learning rate and weight decay are optimized by performing a grid search using validation set. The best model is obtained by retraining with optimal parameters on the combined train and validation set. 10 different log spaced values in [-3, 0] are used for learning rate while 9 log values in [-10, -2] are used for weight decay.

A.2.2.5 MLP Architecture

For the proposed prediction head, we experiment with linear, 2 and 4 layer MLPs. Each MLP layer is composed of a linear projection followed by 1D batch normalization and ReLU activation. Let the dimension of the student backbone output be m and that of teacher d. For linear evaluation, a single layer with input and output dimensions of (m,d) is used. For a 2 layer MLP, following [107], we use the dimensions $(m,2m,d)$. We extend this to a 4 layer MLP with the following intermediate feature dimensions: $(m,2m,m,2m,d)$. Batch normalization and ReLU activation are not employed at the end of layer 2 for 4 layer MLP head (equivalent to stacking two 2-layer MLP heads). For our ablation on the role of MLP head during inference (table 2 in main paper), we compare the performance of our method at different layers of the MLP head from a single trained network. For fair comparison, we require all the intermediate dimensions to be same as that of the output. Thus, for this experiment alone, we use an MLP such that the feature dimensions are (m,d,d,d,d). The output dimension (m) for ResNet-18, ResNet-50 and MobileNet-v2 are 512, 2048 and 1280 respectively. The teacher output dimensions are 2048 and 8192 respectively for ResNet-50 and

ResNet-50x4 architectures. From table 2 (network with MLP feature dimensions (512, 2048, 2048, 2048, 2048)) and table 4 (network with MLP feature dimensions (512, 1024, 512, 1024, 2048)) results, we observe that higher MLP feature dimensions might help further boost performance (65.7 vs 65.1 on ImageNet linear). More ablations on this are necessary to optimize the MLP architecture.

APPENDIX B

Memory Efficiency of Models in 3D, Vision and NLP Applications

B.1 CompGS: Smaller and Faster Gaussian Splatting with Vector Quantization

In this supplementary pdf, we compare the performance of our CompGS with state-of-the-art approaches on the NeRF-Synthetic dataset (Section B.2). Section B.3 shows exploratory results on generalization of the learnt vector codebook across scenes and Section B.5 provides insights on the learnt codebook assignments. We also provide scene-wise results (section B.6), ablations on baselines (section B.7) and additional visualizations and qualitative comparisons on the ARKit dataset (section B.9). The anonymized code for our approach is provided along with this pdf in the zipped supplementary material file.

B.2 Results on NeRF-Synthetic and DeepBlending Datasets

The results (PSNR) for the NeRF-Synthetic dataset [208] are presented in Table B.1. Our CompGS approach achieves an impressive average improvement of 1.13 points in PSNR compared to the 3DGS-No-SH baseline while using less than half its memory. As reported in the main submission, we report metrics for 3DGS both from the original paper and using our own runs. We observe an improvement for 3DGS [151] over their official reported numbers by 0.5 points. The comparison between our CompGS and SOTA methods for novel view synthesis on Deep Blending dataset is shown in Table B.2. CompGS maintains the speed and performance advantages of 3DGS while being 40× to 50× smaller. CompGS-32K BitQ is the post-training bit quantized version of CompGS-32K, in which position parameters are 16-bits, opacity is 8 bits, and the rest are 32 bits.

B.3 Generalization of codebook across scenes

We train our vector quantization approach including the codebook and the code assignments on a single scene (‘Counter’) of the Mip-NeRF360 dataset. We then freeze the codebook and learn only assignments for the rest of the eight scenes in the dataset and report the averaged performance metrics over all scenes.

TABLE B.1. **Results on NeRF-Synthetic dataset.** Here, we present the PSNR values for the synthesized novel views on the NeRF-Synthetic dataset [208]. Our CompGS approach achieves an impressive average improvement of 1.13 points in PSNR compared to the 3DGS-No-SH baseline while using less than half its memory. As reported in the main submission, we report metrics for 3DGS both from the original paper and using our own runs. We observe an improvement of 3DGS over the reported numbers by 0.5points. * indicates our own run.

	Mic	Chair	Ship	Materials	Lego	Drums	Ficus	Hotdog	Avg.
Plenoxels	33.26	33.98	29.62	29.14	34.10	25.35	31.83	36.81	31.76
INGP-Base	36.22	35.00	31.10	29.78	36.39	26.02	33.51	37.40	33.18
Mip-Nerf	36.51	35.14	30.41	30.71	35.70	25.48	33.29	37.48	33.09
Point-NeRF	35.95	35.40	30.97	29.61	35.04	26.06	36.13	37.30	33.30
3DGS	35.36	35.83	30.80	30.00	35.78	26.15	34.87	37.72	33.32
3DGS*	36.80	35.51	31.69	30.48	36.06	26.28	35.49	38.06	33.80
3DGS-No-SH	34.37	34.09	29.86	28.42	34.84	25.48	32.30	36.43	31.97
CompGS 4k	35.99	34.92	31.05	29.74	35.09	25.93	35.04	37.04	33.10

TABLE B.2. **Comparison with SOTA methods for novel view synthesis on Deep Blending dataset.** CompGS is a vector quantized version of 3DGS that maintains the speed and performance advantages of 3DGS while being 40× to 50× **smaller**. CompGS 32K BitQ is the post-training bit quantization version of CompGS 32K, in which position parameters are 16-bits, opacity is 8 bits, and rest are 32 bits. *Reproduced using official code. † Reported from 3DGS [151]. Our timings for 3DGS and CompGS are reported using a RTX6000 GPU while those with † used A6000 GPU. We boldface entries for emphasis.

	Deep Blending					
Method	SSIM↑	PSNR↑	LPIPS↓	FPS	Mem (MB)	Train Time(m)
Plenoxels† [88]	0.795	23.06	0.510	11.2	2700	27.5
INGP-Base† [213]	0.797	23.62	0.423	3.26	13	6.31
INGP-Big † [213]	0.817	24.96	0.390	2.79	48	8.00
M-NeRF360† [19]	0.901	29.40	0.245	0.09	8.6	48h
3DGS † [151]	0.903	29.41	0.243	137	676	36.2
3DGS * [151]	0.899	29.49	0.246	151	662	19.3
LigthGaussian [78]	-	-	-	-	-	-
CGR [?]	0.900	29.73	0.258	181	23.8	-
CGS [?]	0.898	29.38	0.253	-	25.30	-
CompGS 16K	0.906	29.90	0.252	485	12	20.3
CompGS 32K	0.907	29.90	0.251	484	13	26.2
CompGS 32K BitQ	0.907	29.89	0.251	484	8	26.2

In addition to the results in Fig.5 of the main submission, we provide results with our 32K variant here in table B.3. Interestingly, we observe that the shared codebook generalizes well across all scenes with a small drop in performance compared to learning a codebook for each scene. Sharing learnt codebook can further reduce the memory requirement and can help speed up the training of CompGS. The quality of the codebook

TABLE B.3. **Effect of shared codebook.** We train our vector quantization approach including the codebook and the code assignments on a single scene ('Counter') of the Mip-NeRF360 dataset. We then freeze the codebook and learn only assignments for the rest of the eight scenes in the dataset and report the averaged performance metrics over all scenes. Interestingly, we observe that the shared codebook generalizes well across all scenes with a small drop in performance compared to learning a codebook for each scene. Sharing learnt codebook can further reduce the memory requirement and can help speed up the training of CompGS. The quality of the codebook can be improved by learning it over multiple scenes.

Dataset	Mip-NeRF360		
Method	SSIM$^{\uparrow}$	PSNR$^{\uparrow}$	LPIPS$^{\downarrow}$
3DGS	0.815	27.21	0.214
3DGS *	0.813	27.42	0.217
CompGS 4K	0.804	26.97	0.234
CompGS 32K	0.806	27.12	0.240
CompGS Shared Codebook 4K	0.797	26.64	0.242
CompGS Shared Codebook 32K	0.800	26.780	0.247

Scene	1	2	3	4	5	6	7	8	9	10	11
Min	25.140	31.774	28.854	21.414	27.084	30.864	31.134	26.522	22.418	21.733	25.250
Max	25.271	32.183	29.068	21.562	27.293	31.425	31.545	26.624	22.585	22.193	25.415
Mean	25.221	32.038	28.997	21.491	27.227	31.228	31.427	26.583	22.495	21.940	25.333
Diff	0.131	0.409	0.215	0.148	0.210	0.561	0.410	0.102	0.167	0.460	0.165
Std	0.032	0.109	0.051	0.037	0.048	0.155	0.105	0.024	0.045	0.123	0.046

TABLE B.4. **Variance in 3DGS performance:** We run the 3DGS baseline method 20 times on all the scenes of Mip-NeRF360 and Tanks&Temples datasets with different seeds and report the statistics of the performance. We only report the PSNRs for brevity. Some of the scenes (Bonsai, Kitchen, Room and Train) have a huge difference in performance between their best and worst runs (denoted as 'Diff') with a PSNR difference more than 0.4. The median of the standard deviation of these runs across all scenes is 0.048.

can be improved by learning it over multiple scenes. Fig. B.3 shows qualitative comparison of the same. There are no apparent differences between CompGS and CompGS-Shared-Codebook approaches.

B.4 Variance in 3DGS Performance

We observe huge differences in the reported performance of 3DGS across different published works. To analyze this, we run the 3DGS baseline method 20 times on all the scenes of Mip-NeRF360 and Tanks&Temples datasets with different seeds and report the statistics of the performance in table B.4. We only report the PSNRs for brevity. Some of the scenes (Bonsai, Kitchen, Room and Train) have a huge difference in performance between their best and worst runs with a PSNR difference more than 0.4. Thus, one must be careful when comparing performances of methods with small differences. The median of the standard deviation of these runs across all scenes is 0.048.

B.5 Analysis of learnt code assignments

In Fig. B.1, we plot the sorted histogram of the code assignments (cluster to which each Gaussian belongs to) for each parameter on the 'Train' scene of Tanks&Temples dataset. We observe that just a single code out of the 512 in total is assigned to nearly 5% of the Gaussians for both the SH and DC parameters. Similarly, a few clusters dominate even in the case of rotation and scale parameters, albeit to a lower extent. Such a non-uniform distribution of cluster sizes suggest that further compression can be achieved by using Huffman coding to store the assignment indices.

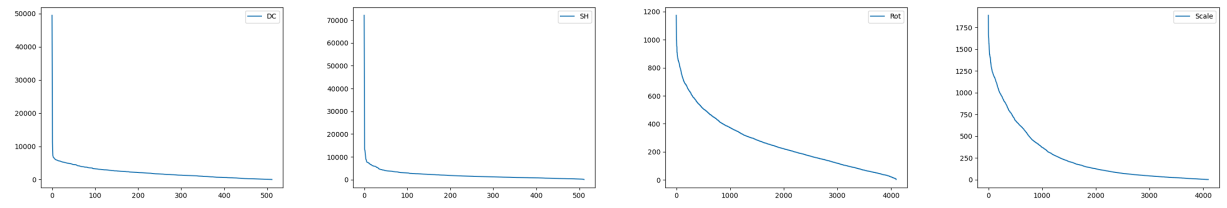


FIGURE B.1. **Histogram of code assignments.** We plot the sorted histogram of the code assignments (cluster to which each Gaussian belongs to) for each parameter on the 'Train' scene of Tanks&Temples dataset. We observe that just a single code out of the 512 in total is assigned to nearly 5% of the Gaussians for both the SH and DC parameters. Similarly, a few clusters dominate even in the case of rotation and scale parameters, albeit to a lower extent. Such a non-uniform distribution of cluster sizes suggest that further compression can be achieved by using Huffman coding to store the assignment indices.

B.6 Scene-wise Metrics

For brevity, we reported the averaged metrics over all scenes in a dataset in our main submission. Here in table B.5, we provide the detailed scene-wise metrics for MipNerf-360, Tanks and Temples and DeepBlending datasets.

B.7 Ablations for Gaussian count reduction

We perform ablations to choose the right hyperparameters for the baseline approaches to reduce the Gaussian count. The metrics for the chosen settings are reported in table 3 of the main submission. The ablations for minimum opacity, densification interval and end iteration and gradient threshold are shown in tables B.6, B.7, B.8 and B.9 respectively. Among these baselines, modify gradient threshold provides the best trade-off between model size and performance.

Table B.5. **Scene-wise metrics.** We report the scene-wise metrics on all the scenes for both 3DGS and CompGS 32K. As observed in the averaged metrics in the main submission, CompGS achieves a high level of compression and fast rendering without losing much on rendering quality.

Scene	Method	SSIM$^{\uparrow}$	PSNR$^{\uparrow}$	LPIPS$^{\downarrow}$	Train Time(s)	FPS	Mem(MB)	#Gauss
Bicycle	3DGS	0.763	25.169	0.212	1845	68	1422	6026079
	CompGS 32K	0.755	25.068	0.244	2123	242	29	1314018
Bonsai	3DGS	0.940	31.918	0.206	876	276	293	1241520
	CompGS 32K	0.932	31.195	0.223	1405	492	10	377227
Counter	3DGS	0.906	29.018	0.202	968	208	283	1200091
	CompGS 32K	0.895	28.467	0.222	1571	356	10	385515
Flowers	3DGS	0.602	21.456	0.339	1192	133	851	3605827
	CompGS 32K	0.588	21.262	0.367	1744	343	23	1037132
Garden	3DGS	0.863	27.241	0.108	1870	76	1347	5709543
	CompGS 32K	0.848	26.822	0.140	2177	258	30	1370624
Kitchen	3DGS	0.926	31.510	0.127	1206	158	425	1801403
	CompGS 32K	0.918	30.774	0.142	1729	317	13	564382
Room	3DGS	0.917	31.346	0.221	1020	190	364	1541909
	CompGS 32K	0.912	31.131	0.235	1439	444	9	327191
Stump	3DGS	0.772	26.651	0.215	1445	104	1136	4815087
	CompGS 32K	0.770	26.605	0.236	1936	303	27	1226044
Treehill	3DGS	0.633	22.504	0.327	1213	122	879	3723675
	CompGS 32K	0.634	22.747	0.355	1744	336	23	1002290
Train	3DGS	0.811	21.991	0.209	563	253	254	1077461
	CompGS 32K	0.804	21.789	0.231	1169	456	12	500811
Truck	3DGS	0.878	25.385	0.148	897	159	611	2588966
	CompGS 32K	0.872	25.092	0.165	1306	494	13	540081
DrJohnson	3DGS	0.898	29.089	0.247	1312	121	772	3270679
	CompGS 32K	0.906	29.445	0.249	1717	379	17	714902
Playroom	3DGS	0.901	29.903	0.246	1003	181	551	2335846
	CompGS 32K	0.908	30.347	0.253	1422	589	10	393414

Table B.6. **Ablation study on opacity threshold**. We changed the default value of 0.005 for minimum opacity as a baseline approach for compressing 3DGS by reducing the number of Gaussians. Gaussians with opacity values below the threshold are pruned, resulting in smaller models for lower thresholds. Table 3 in main paper shows the results of this experiment when min opacity is 0.1.

	Mip-NeRF360				Tanks&Temples				Deep Blending			
	SSIM	PSNR	LPIPS	#Gauss	SSIM	PSNR	LPIPS	#Gauss	SSIM	PSNR	LPIPS	#Gauss
0.05	0.810	27.308	0.230	1.93M	0.839	23.508	0.194	1.04M	0.902	29.542	0.251	1.47M
0.1	0.802	27.120	0.244	1.46M	0.833	23.439	0.204	780K	0.902	29.504	0.255	1.01M

TABLE B.7. **Ablation study on densification interval**. We modify the densification interval in 3DGS as a baseline approach for compressing 3DGS by reducing the number of Gaussians. Higher intervals results in less frequent densification and thus smaller number of Gaussians. Table 3 in main paper shows the results of this experiment when interval is 500.

	Mip-NeRF360				Tanks&Temples				Deep Blending			
	SSIM	PSNR	LPIPS	#Gauss	SSIM	PSNR	LPIPS	#Gauss	SSIM	PSNR	LPIPS	#Gauss
300	0.803	27.201	0.241	1.70M	0.837	23.517	0.195	1.00M	0.902	29.705	0.253	1.30M
500	0.794	26.98	0.255	1.07M	0.832	23.36	0.206	709K	0.902	29.76	0.258	844K

TABLE B.8. **Ablation study on densification end iteration**. Early or late stopping of densification process impacts the number of Gaussians and the model performance of 3DGS. We report the results with the value set to 3000 in our main submission (table 3).

	Mip-NeRF360				Tanks&Temples				Deep Blending			
	SSIM	PSNR	LPIPS	#Gauss	SSIM	PSNR	LPIPS	#Gauss	SSIM	PSNR	LPIPS	#Gauss
5000	0.797	27.199	0.241	1.92M	0.838	23.599	0.188	1.12M	0.897	29.486	0.256	1.34M
3000	0.780	27.02	0.267	1.12M	0.835	23.55	0.194	810K	0.896	29.42	0.264	795K

TABLE B.9. **Ablation study on gradient threshold**. We modify the gradient threshold, used as a criterion in the densification process of 3DGS. A higher threshold results in a smaller 3DGS model. Among the baselines considered for reducing Gaussian count, modify gradient threshold provides the best trade-off between model size and performance. Table 3 in main paper shows the results of this experiment for gradient threshold of 0.00045.

	Mip-NeRF360				Tanks&Temples				Deep Blending			
	SSIM	PSNR	LPIPS	#Gauss	SSIM	PSNR	LPIPS	#Gauss	SSIM	PSNR	LPIPS	#Gauss
0.00035	0.786	26.934	0.265	1.27M	0.833	23.579	0.203	833K	0.901	29.574	0.254	1.41M
0.00045	0.769	26.57	0.292	809K	0.825	23.31	0.217	578K	0.900	29.49	0.260	1.01M
0.00055	0.754	26.290	0.312	552K	0.818	23.041	0.228	433K	0.899	29.544	0.265	764K

B.8 Details of ARKit dataset.

Unlike visual recognition community that uses large scale benchmarks, interestingly, radiance field modeling community traditionally has used small datasets with a handful of 3D scenes: maximum of 13 total real world scenes in several papers (e.g., 3DGS [151]). We believe this is due to the computational cost of NeRF-based methods (several hours of training for each scene). Hence, with the recent advancements in this field including 3DGS that takes only a few minutes to learn a scene, it may be time to move beyond small benchmarks and evaluate methods on larger scale benchmarks. Therefore, we use an existing scene-understanding dataset [21] as a benchmark for radiance field modeling that is an order of magnitude larger than the traditional datasets (200 vs 13 scenes). We believe the community will benefit from using this larger benchmark to evaluate future radiance field methods with a reasonable computational demand. Note that

DL3DV-140 is another large dataset that is concurrent work to ours, so we include that one too in the results of our main paper.

ARKit [21] is an indoor scene understanding dataset comprising $5,048$ scans encompassing $1,661$ distinct scenes. The videos are recorded using the 2020 iPad Pro and have a resolution of 1920×1440. We exclusively utilize the RGB frames from each video. To construct the dataset subset for view synthesis, we randomly select 200 raw videos from the ARKit dataset, extracting a uniform sample of 300 frames from each. Subsequently, we employ the code provided by 3DGS [151] to extract undistorted images and Structure-from-Motion (SfM) information from the input images. The dataset can easily be extended in the future by including more of the remaining scenes from the ARKit dataset.

B.9 Qualitative comparison on ARKit dataset.

Figures B.4 and B.5 provide qualitative results on the ARKit dataset.

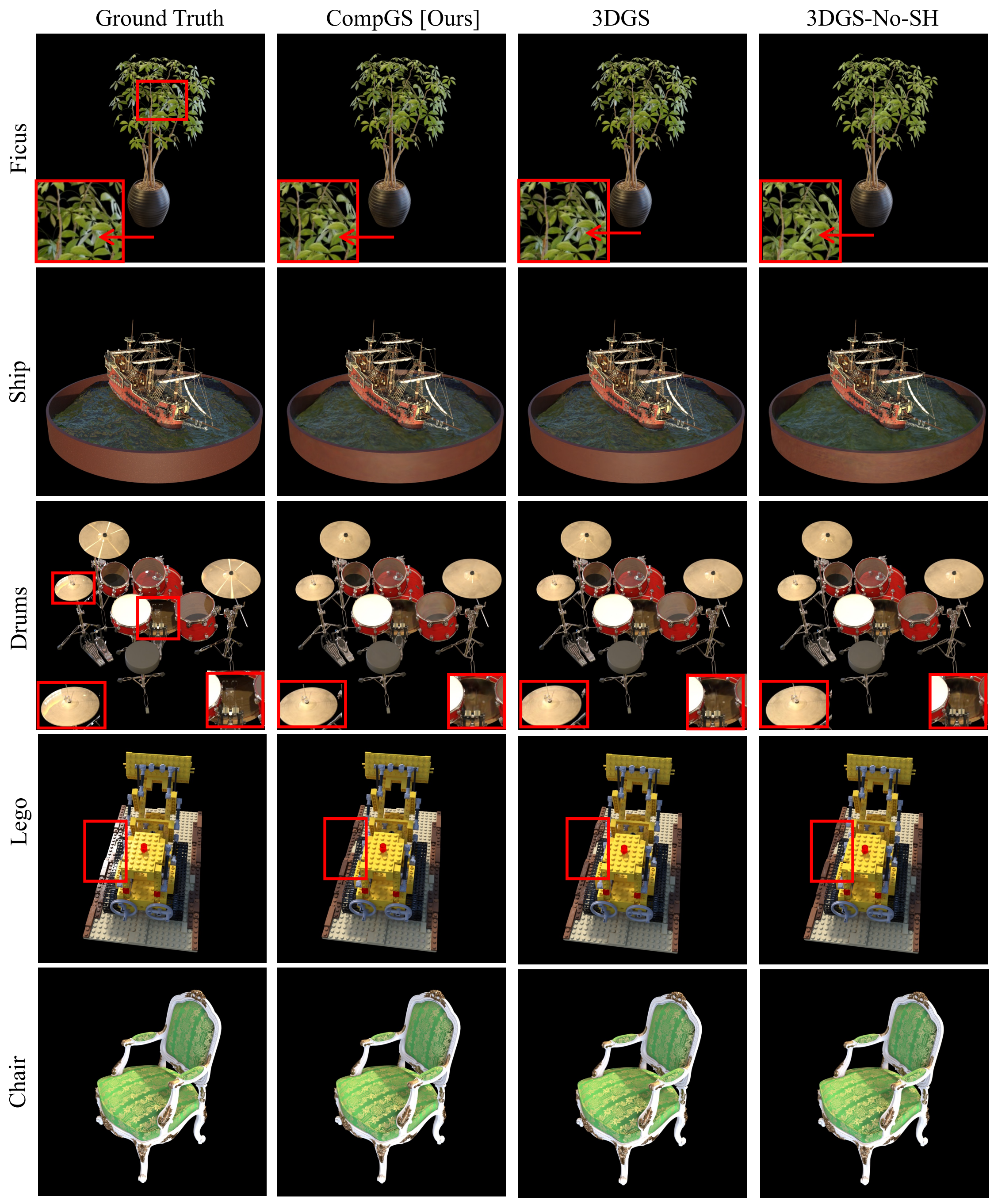


FIGURE B.2. **Visualization of results on Sythetic-NeRF dataset.** We compare the performance of our compressed CompGS with the original 3DGS and 3DGS-No-SH approaches on different scenes of the NeRF-Synthetic dataset. The difference between CompGS and 3DGS-No-SH is apparent in some of these scenes. E.g., 3DGS-No-SH fails to effectively model the brown color of branches and shadows and bright light on the leaves of the 'Ficus' scene. All approaches including 3DGS have imperfect reconstruction in some of the scenes like 'Drums' and 'Lego'. The scenes and views used for visualization were chosen at random.

B.9. QUALITATIVE COMPARISON ON ARKIT DATASET.

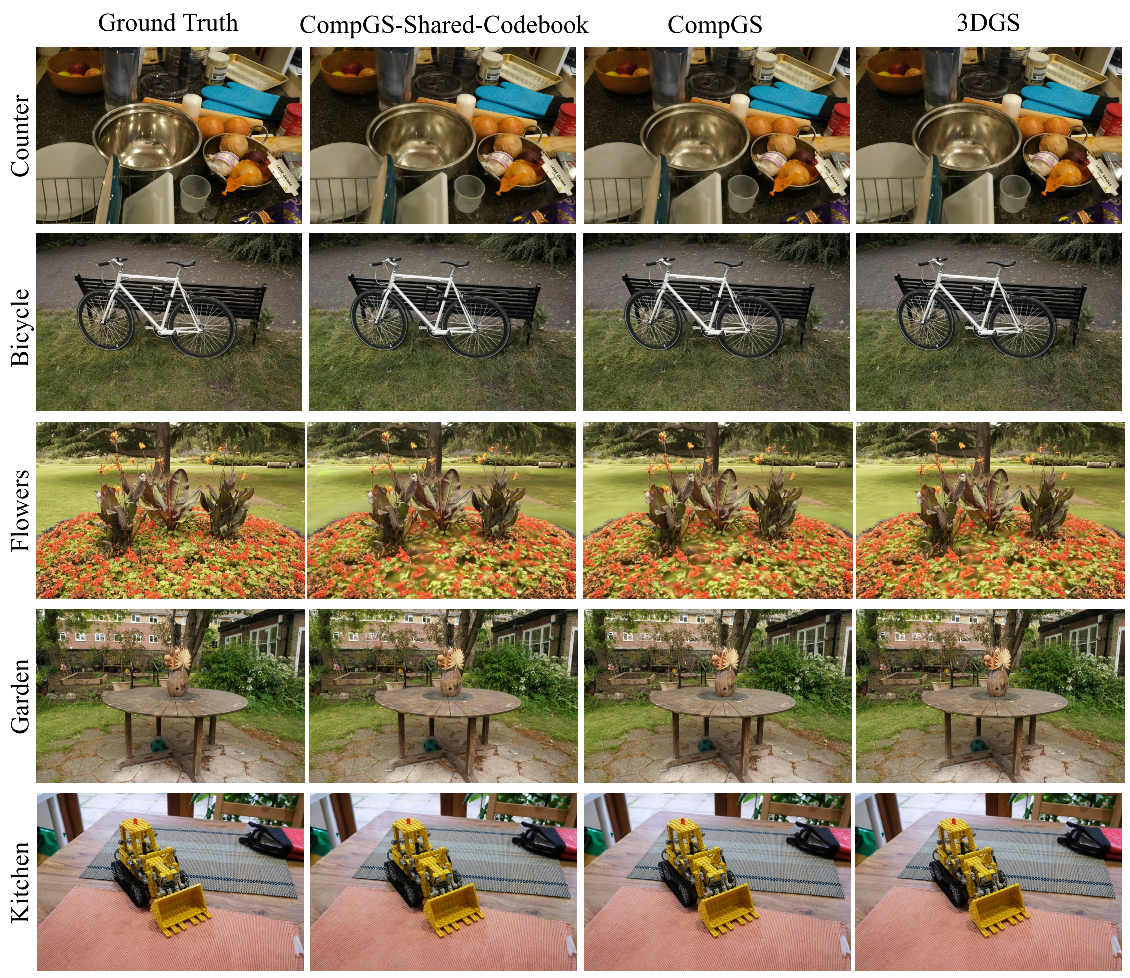


FIGURE B.3. **Qualitative analysis of shared codebook.** We show the generalization of codebook learned using a single scene on various scenes of the Mip-NeRF360 dataset. The codebook was trained on the 'Counter' scene (row-1) and frozen for the remaining scenes. The codebooks for all four parameters (DC, SH, Scale, Rot) are shared across scenes. Both CompGS and CompGS-Shared-Codebook are visually similar to the uncompressed 3DGS with no conspicuous differences between them. 3DGS-No-SH requires twice more memory than CompGS while 3DGS is ten times bigger than CompGS . The scenes and views used for visualization were chosen at random.

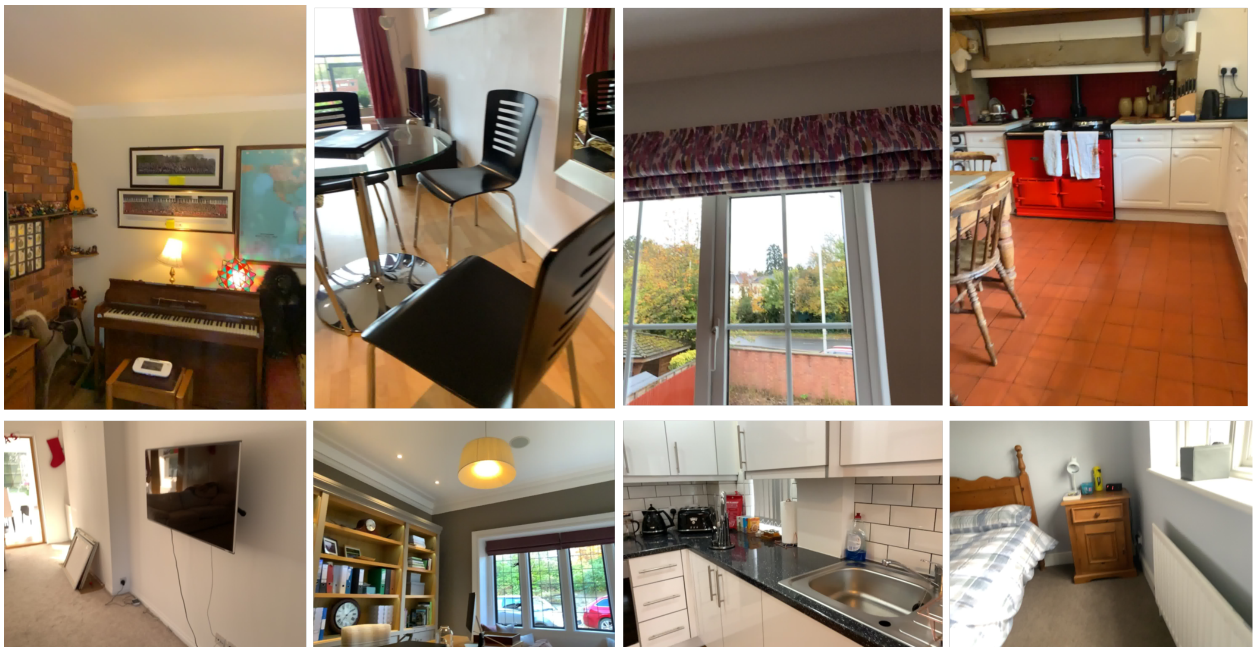


FIGURE B.4. **Visualization of ARKit dataset.** ARKit is a 3D indoor scene dataset captured using a iPads/i-Phones. The dataset consists of videos of indoor environments like houses and office space from multiple view-points. We uniform sample images from each video to form our benchmark dataset for novel view synthesis. Some sample images from different scenes are shown in this figure. The dataset presents unique challenges such as the presence of motion blur due to the use of videos.

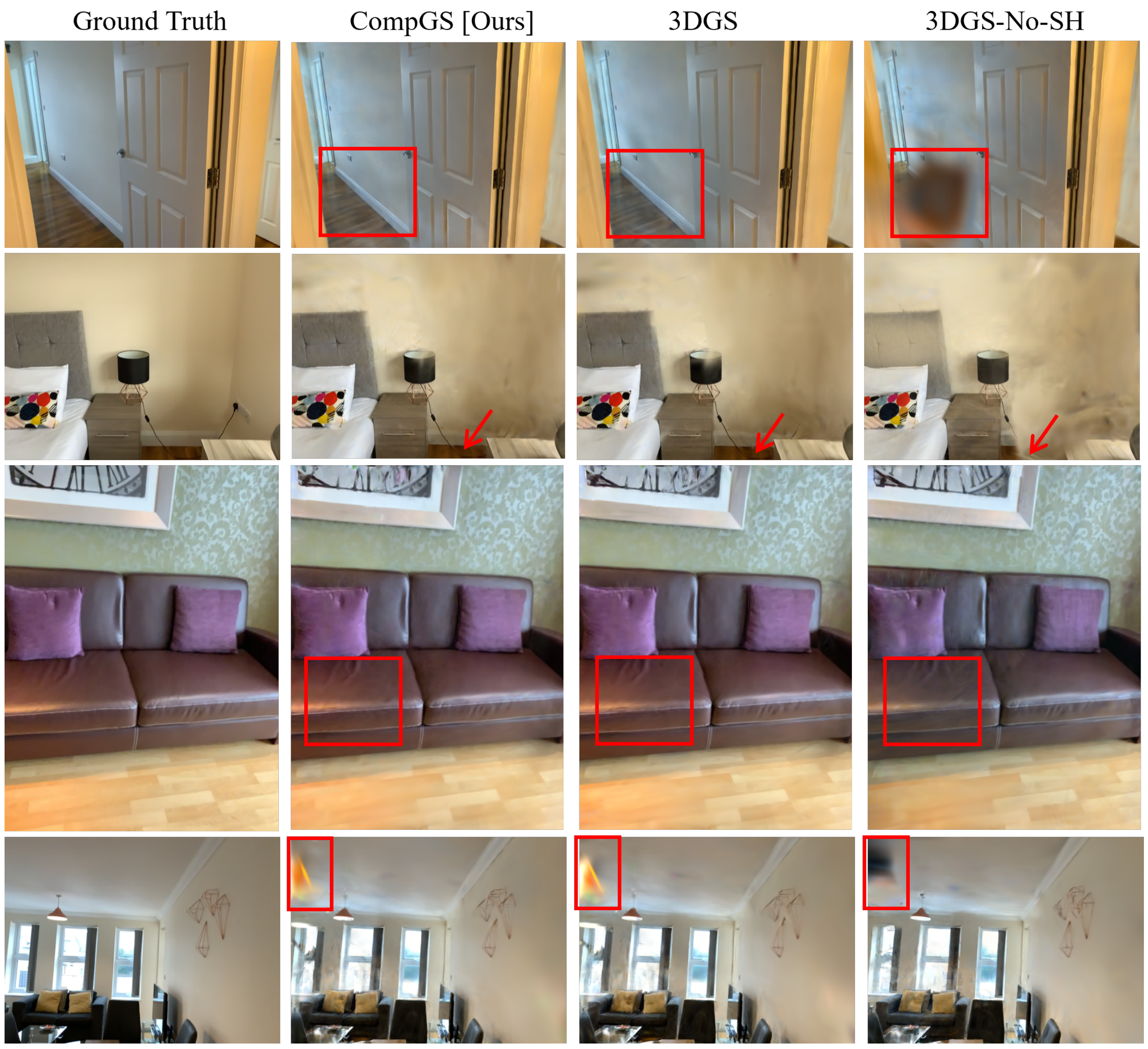


FIGURE B.5. **Qualitative analysis on ARKit dataset.** We visualize the results of CompGS along with the uncomressed 3DGS and its variant 3DGS-No-SH . Presence of large noisy blobs is a common error mode for 3DGS-No-SH on this dataset. It also fails to faithfully reproduce the colors and lighting in several scenes. The visual quality of the synthesized images for all methods is lower on this dataset compared to the scenes present in standard benchmarks like Mip-NeRF360 , indicating its utility as a novel benchmark. Further comparison with various NeRF based approaches and more analysis can help improve the results on this dataset.

B.10 NOLA: Compressing LoRA using Linear Combination of Random Basis

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 nooralinejad2023pranc for our experiments on image classification.

Datasets and Architectures: We consider two architectures in our experiments: ResNet20 with $270K$ parameters, and ResNet18 resnet with $11M$ parameters. We train ResNet20 on CIFAR10 cifar10, and ResNet18 on ImageNet100 imagenet.

Results: We report result of ImageNet100 in Table B.11, and CIFAR10 in Table B.10. 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 B.10. **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 B.11. **Training On ImageNet100:** Result of our method on ImageNet-100 dataset and ResNet18

Method	# Params	Acc.
trained model	11,227,812	**82.1%**
HashedNet chen2015compressing	129,200	52.96%
PRANC	119,200	61.08%
LoRA	150,000	63.50%
NOLA	109,600	64.66%

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 B.12). 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 B.12. **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)

B.10.1 Results of NLG task on DART and WebNLG datasets:

In Table B.13, we report more results similar to Table 3.10 using GPT-2 M and GPT-2 L on DART nan2020dart and WebNLG gardent2017webnlg datasets.

TABLE B.13. **DART and WebNLG Dataset**: Similar to Table 3.10 we compare `NOLA` to other methods. `NOLA` is on par or better with other methods with the same number of parameters.

GPT-2 M									
Method	Adapted Layers	Adapter Rank	# Trainable Parameters	DART			WebNLG		
				BLEU↑	MET↑	TER↓	BLEU↑	MET↑	TER↓
Finetune	All Layers	-	354.000M	46.2	0.39	0.46	46.5	0.38	0.53
Adapter^{L}	Extra Layers	-	0.370M	42.4	0.36	0.48	50.2	0.38	0.43
Adapter^{L}	Extra Layers	-	11.000M	45.2	0.38	0.46	54.9	0.41	0.39
Finetune^{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
Adapter^{L}	Extra Layers	-	0.880M	45.7	0.38	0.46	56.0	0.41	0.39
Adapter^{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

APPENDIX C

Robustness of Efficient Models in Vision Transformers

C.1 SlowFormer: Adversarial Attack on Compute and Energy Consumption of Efficient Vision Transformers

In sections C.2 and C.3, we provide visualizations of our learnt patches and token dropping respectively. In Sec. C.4, we provide additional details on our train and test settings.

C.2 Patch Visualization

In Fig. C.1, 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.

We optimize patches for A-ViT using different initializations and visualize them in Fig. C.2. All patches achieve Attack Success close to 100%. Presence of multiple universal adversarial patches highlights the vulnerability of the current efficient methods.

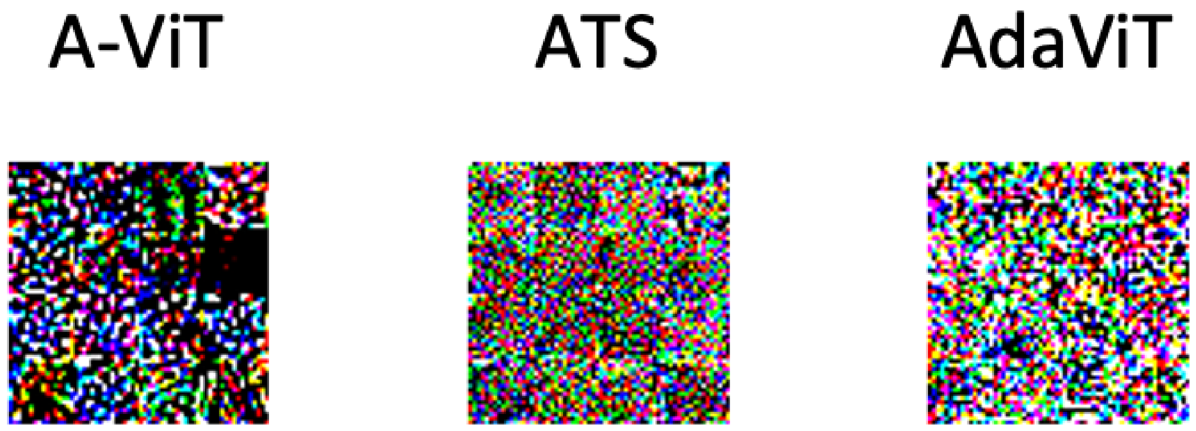


FIGURE C.1. **Visualization of optimized patch:** We show the learnt universal patches for each of the three efficient methods that we attack.

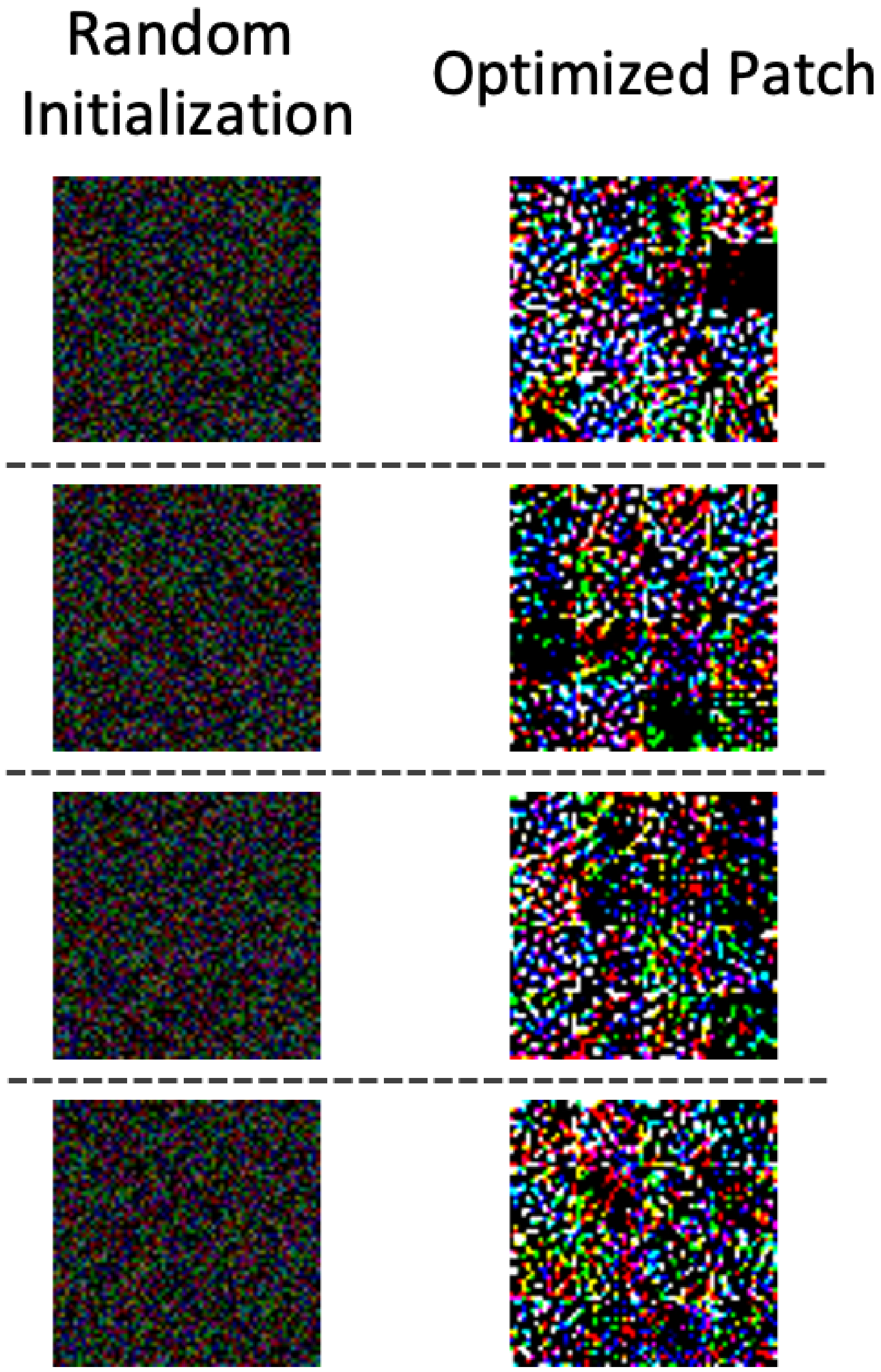


FIGURE C.2. **Optimized patches With different initializations:** Here, we show the optimized patches for A-ViT . A different initialization is used to train each of these patches. All patches achieve Attack Success close to 100%. Presence of multiple universal adversarial patches highlights the vulnerability of the current efficient methods.

We show the evolution of the patch as training progresses in Fig. C.4. The patch is trained to attack A-ViT approach. We observe that the patch converges quickly, requiring less than an epoch for 100% Attack

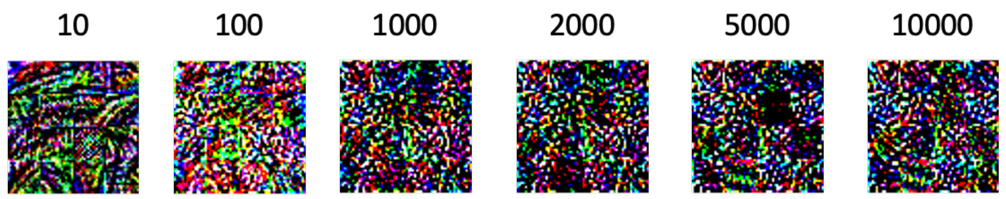


Figure C.3. **Visualization of patch optimization:** We train our patch to attack A-ViT and display the patch at various stages of optimization. We observe that the patch converges quickly. The patch at 1000 iterations (0.1 epoch) is similar to that at 10000 iterations (1 epoch) in terms of both appearance and attack performance.

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.

C.3 Visualization of Token Dropping

In Fig. C.3, 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.

C.4 Implementation Details

ATS Details: As in ATS [81], 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 $[2.0, 0.1, 0.02, 0.01, 0.005, 0.005, 0.00$ The weights are vastly different at initial and final layers to account for the difference in loss magnitudes across layers.

A-ViT Details: The patches are optimized for one epoch with a learning rate of 0.2 and a batch size of 512 ($128 \times$ 4GPUs) using AdamW [197] 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

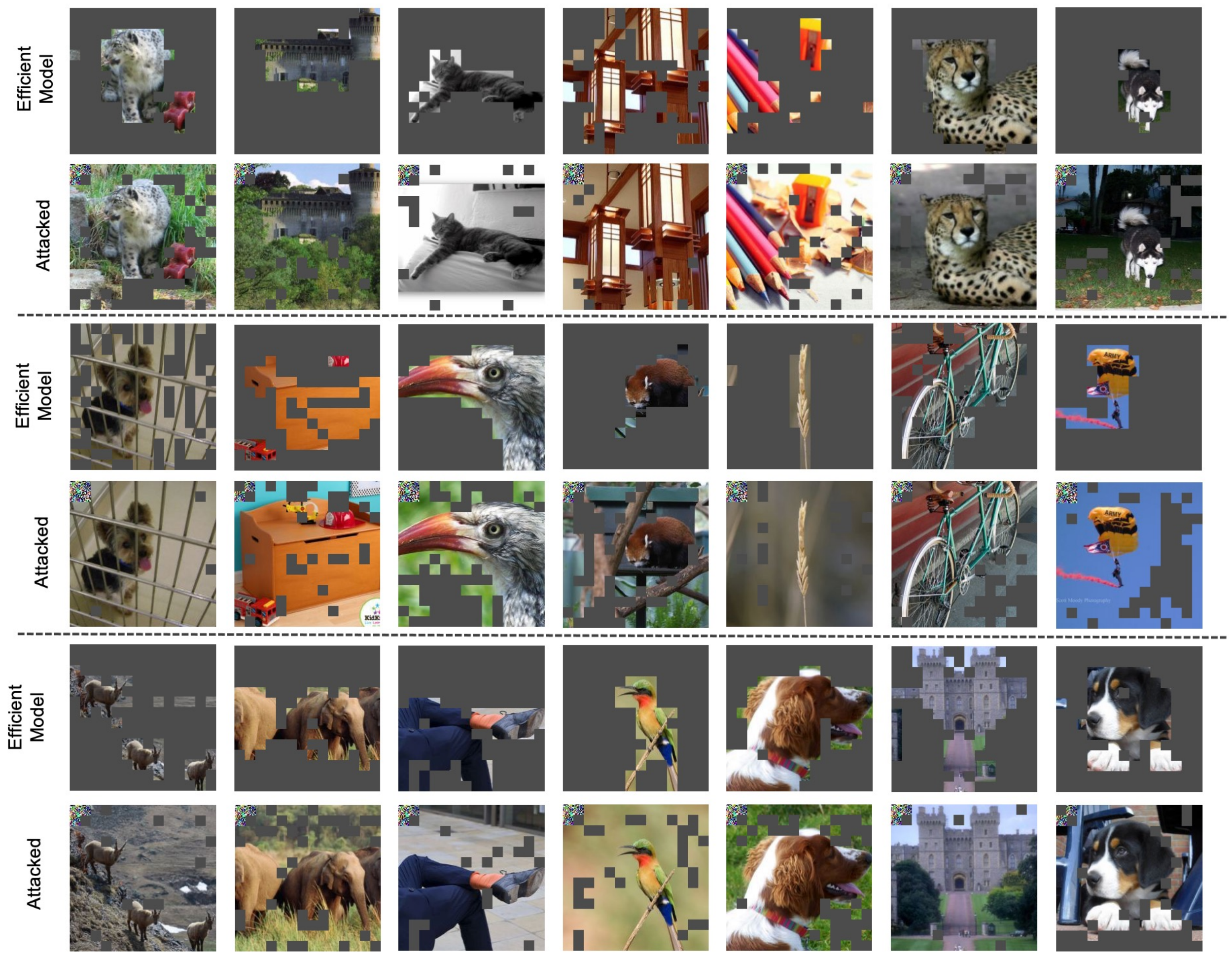


FIGURE C.4. **Visualization of our Energy Attack on Vision Transformers:** Similar to Figure 2 of the main submission, we visualize the A-ViT-Small with and without our attack. We use patch size of 32 for the attack (on the top-left corner). We show pruned tokens at layer 8 of A-ViT-Small. Our attack can recover most of the pruned tokens, resulting in increased computation and power consumption.

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 [197] 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.

Bibliography

[1] *Common crawl*. https://commoncrawl.org/.

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

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

[4] *Meta llama 3*. https://ai.meta.com/blog/meta-llama-3/.

[5] *Official code repository of 3d gaussian splatting for real-time radiance field rendering*. https://github.com/graphdeco-inria/gaussian-splatting.

[6] *Official pytorch code base for moco*. https://github.com/facebookresearch/moco.

[7] *Official pytorch resnet-50 pretrained model*. https://pytorch.org/vision/stable/models.html.

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

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

[10] 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.

[11] 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.

[12] 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).

[13] 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).

[14] 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).

[15] 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).

[16] L. J. Ba and R. Caruana, *Do deep nets really need to be deep?*, arXiv preprint arXiv:1312.6184, (2013).

[17] 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.

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

[19] J. T. Barron, B. Mildenhall, D. Verbin, P. P. Srinivasan, and P. Hedman, *Mip-nerf 360: Unbounded anti-aliased neural radiance fields*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 5470–5479.

[20] G. Baruch, Z. Chen, A. Dehghan, T. Dimry, Y. Feigin, P. Fu, T. Gebauer, B. Joffe, D. Kurz, A. Schwartz, and E. Shulman, *Arkitscenes - a diverse real-world dataset for 3d indoor scene understanding using mobile rgb-d data*, in NeurIPS, 2021.

[21] G. Baruch, Z. Chen, A. Dehghan, Y. Feigin, P. Fu, T. Gebauer, D. Kurz, T. Dimry, B. Joffe, A. Schwartz, and E. Shulman, *ARKitscenes: A diverse real-world dataset for 3d indoor scene understanding using mobile RGB-d data*, in Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1), 2021.

[22] 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.

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

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

[25] L. Beyer, X. Zhai, A. Royer, L. Markeeva, R. Anil, and A. Kolesnikov, *Knowledge distillation: A good teacher is patient and consistent*, arXiv preprint arXiv:2106.05237, (2021).

[26] 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.

[27] 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).

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

[29] ———, *Food-101–mining discriminative components with random forests*, in European conference on computer vision, Springer, 2014, pp. 446–461.

[30] 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.

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

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

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

[34] C. Buciluă, 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.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

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

[40] A. Chen, Z. Xu, A. Geiger, J. Yu, and H. Su, *Tensorf: Tensorial radiance fields*, in European Conference on Computer Vision, Springer, 2022, pp. 333–350.

[41] G. Chen, W. Choi, X. Yu, T. Han, and M. Chandraker, *Learning efficient object detection models with knowledge distillation*, in Proceedings of the 31st International Conference on Neural Information Processing Systems, 2017, pp. 742–751.

[42] 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.

[43] 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.

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

[45] 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.

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

[47] 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.

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

[49] 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.

[50] 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.

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

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

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

[54] X. Chen and K. He, *Exploring simple siamese representation learning*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 15750–15758.

[55] Z. Chen, T. Funkhouser, P. Hedman, and A. Tagliasacchi, *Mobilenerf: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2023, pp. 16569–16578.

[56] B. Cheng, A. G. Schwing, and A. Kirillov, *Per-pixel classification is not all you need for semantic segmentation*, in Advances in Neural Information Processing Systems (NeurIPS), 2021.

[57] M. Cho, K. A. Vahid, Q. Fu, S. Adya, C. C. Del Mundo, M. Rastegari, D. Naik, and P. Zatloukal, *edkm: An efficient and accurate train-time weight clustering for large language models*, arXiv preprint arXiv:2309.00964, (2023).

[58] 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.

[59] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi, *Describing textures in the wild*, in Computer Vision and Pattern Recognition, 2014.

[60] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi, *Describing textures in the wild*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2014, pp. 3606–3613.

[61] P. C. Cosman, K. L. Oehler, E. A. Riskin, and R. M. Gray, *Using vector quantization for image processing*, Proceedings of the IEEE, 81 (1993), pp. 1326–1341.

[62] 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.

[63] S. Dash, I. Lyngaas, J. Yin, X. Wang, R. Egele, G. Cong, F. Wang, and P. Balaprakash, *Optimizing distributed training on frontier for large language models*, arXiv preprint arXiv:2312.12705, (2023).

[64] C. L. Deng and E. Tartaglione, *Compressing explicit voxel grid representations: fast nerfs become also small*, in Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, 2023, pp. 1236–1245.

[65] 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.

[66] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, *Llm. int8 (): 8-bit matrix multiplication for transformers at scale*, arXiv preprint arXiv:2208.07339, (2022).

[67] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, *Qlora: Efficient finetuning of quantized llms*, 2023.

[68] 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.

[69] 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.

[70] 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).

[71] 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.

[72] 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.

[73] 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.

[74] M. Elbayad, J. Gu, E. Grave, and M. Auli, *Depth-adaptive transformer*, arXiv preprint arXiv:1910.10073, (2019).

[75] W. H. Equitz, *A new vector quantization clustering algorithm*, IEEE transactions on acoustics, speech, and signal processing, 37 (1989), pp. 1568–1575.

[76] 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.

[77] 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.

[78] Z. Fan, K. Wang, K. Wen, Z. Zhu, D. Xu, and Z. Wang, *Lightgaussian: Unbounded 3d gaussian compression with 15x reduction and 200+ fps*, arXiv preprint arXiv:2311.17245, (2023).

[79] 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.

[80] 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.

[81] 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.

[82] 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.

[83] 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).

[84] 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*, in 2004 conference on computer vision and pattern recognition workshop, IEEE, 2004, pp. 178–178.

[85] R. Feinman, R. R. Curtin, S. Shintre, and A. B. Gardner, *Detecting adversarial samples from artifacts*, arXiv preprint arXiv:1703.00410, (2017).

[86] 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.

[87] J. Flynn, I. Neulander, J. Philbin, and N. Snavely, *Deepstereo: Learning to predict new views from the world's imagery*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 5515–5524.

[88] S. Fridovich-Keil, A. Yu, M. Tancik, Q. Chen, B. Recht, and A. Kanazawa, *Plenoxels: Radiance fields without neural networks*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 5501–5510.

[89] 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.

[90] 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).

[91] T. Furlanello, Z. C. Lipton, M. Tschannen, L. Itti, and A. Anandkumar, *Born again neural networks*, 2018.

[92] 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).

[93] S. J. Garbin, M. Kowalski, M. Johnson, J. Shotton, and J. Valentin, *Fastnerf: High-fidelity neural rendering at 200fps*, in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021, pp. 14346–14355.

[94] 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.

[95] A. Gersho and R. M. Gray, *Vector quantization and signal compression*, vol. 159, Springer Science & Business Media, 2012.

[96] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, *A survey of quantization methods for efficient neural network inference*, arXiv preprint arXiv:2103.13630, (2021).

[97] 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.

[98] S. Gidaris, P. Singh, and N. Komodakis, *Unsupervised representation learning by predicting image rotations*, in International Conference on Learning Representations, 2018.

[99] S. Girish, K. Gupta, and A. Shrivastava, *Eagles: Efficient accelerated 3d gaussians with lightweight encodings*, arXiv preprint arXiv:2312.04564, (2023).

[100] 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.

[101] Y. Gong, L. Liu, M. Yang, and L. Bourdev, *Compressing deep convolutional networks using vector quantization*, in arXiv preprint arXiv:1412.6115, 2014.

[102] 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.

[103] I. J. Goodfellow, J. Shlens, and C. Szegedy, *Explaining and harnessing adversarial examples*, in arXiv preprint arXiv:1412.6572, 2014.

[104] A. Graves, *Adaptive computation time for recurrent neural networks*, arXiv preprint arXiv:1603.08983, (2016).

[105] R. Gray, *Vector quantization*, IEEE Assp Magazine, 1 (1984), pp. 4–29.

[106] 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.

[107] 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).

[108] S. Gu, D. Chen, J. Bao, F. Wen, B. Zhang, D. Chen, L. Yuan, and B. Guo, *Vector quantized diffusion model for text-to-image synthesis*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 10696–10706.

[109] J. Guan, Y. Liu, Q. Liu, and J. Peng, *Energy-efficient amortized inference with cascaded deep classifiers*, arXiv preprint arXiv:1710.03368, (2017).

[110] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, *Deep learning with limited numerical precision*, in International conference on machine learning, PMLR, 2015, pp. 1737–1746.

[111] K. Hambardzumyan, H. Khachatrian, and J. May, *WARP: Word-level Adversarial ReProgramming*, arXiv:2101.00121 [cs], (2020). arXiv: 2101.00121.

[112] S. Han, H. Mao, and W. J. Dally, *Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding*, arXiv preprint arXiv:1510.00149, (2015).

[113] T. Han, W. Xie, and A. Zisserman, *Self-supervised co-training for video representation learning*, 2021.

[114] 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.

[115] M. Haque, S. Chen, W. A. Haque, C. Liu, and W. Yang, *Nodeattack: Adversarial attack on the energy consumption of neural odes*, (2021).

[116] S. Hayou, J.-F. Ton, A. Doucet, and Y. W. Teh, *Robust pruning at initialization*, arXiv preprint arXiv:2002.08797, (2020).

[117] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick, *Masked autoencoders are scalable vision learners*, arXiv preprint arXiv:2111.06377, (2021).

[118] 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.

[119] 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 (CVPR), 2020.

[120] ———, *Momentum contrast for unsupervised visual representation learning*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 9729–9738.

[121] 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.

[122] ———, *Deep residual learning for image recognition*, in CVPR, 2016, pp. 770–778.

[123] P. Hedman, J. Philip, T. Price, J.-M. Frahm, G. Drettakis, and G. Brostow, *Deep blending for free-viewpoint image-based rendering*, ACM Transactions on Graphics (ToG), 37 (2018), pp. 1–15.

[124] P. Hedman, P. P. Srinivasan, B. Mildenhall, J. T. Barron, and P. Debevec, *Baking neural radiance fields for real-time view synthesis*, in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021, pp. 5875–5884.

[125] 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).

[126] 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).

[127] P. Henzler, N. J. Mitra, and T. Ritschel, *Escaping plato's cave: 3d shape from adversarial rendering*, in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 9984–9993.

[128] B. Heo, M. Lee, S. Yun, and J. Y. Choi, *Knowledge transfer via distillation of activation boundaries formed by hidden neurons*, in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, 2019, pp. 3779–3787.

[129] 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.

[130] G. Hinton, O. Vinyals, and J. Dean, *Distilling the knowledge in a neural network*, in NIPS Deep Learning and Representation Learning Workshop, 2015.

[131] J. Ho, N. Kalchbrenner, D. Weissenborn, and T. Salimans, *Axial attention in multidimensional transformers*, arXiv preprint arXiv:1912.12180, (2019).

[132] 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).

[133] 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).

[134] 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).

[135] 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.

[136] J. Hu, L. Shen, and G. Sun, *Squeeze-and-excitation networks*, in CVPR, 2018, pp. 7132–7141.

[137] 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).

[138] B. Huang, X. Yan, A. Chen, S. Gao, and J. Yu, *Pref: Phasorial embedding fields for compact neural representations*, arXiv preprint arXiv:2205.13524, (2022).

[139] 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).

[140] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, *Densely connected convolutional networks*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 4700–4708.

[141] J. Huang, Q. Dong, S. Gong, and X. Zhu, *Unsupervised deep learning via affinity diffusion*, in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, 2020, pp. 11029–11036.

[142] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu, et al., *Gpipe: Efficient training of giant neural networks using pipeline parallelism*, Advances in neural information processing systems, 32 (2019), pp. 103–112.

[143] T. Huynh, S. Kornblith, M. R. Walter, M. Maire, and M. Khademi, *Boosting contrastive self-supervised learning with false negative cancellation*, 2020.

[144] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, *Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size*, arXiv preprint arXiv:1602.07360, (2016).

[145] 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.

[146] 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.

[147] 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).

[148] 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.

[149] 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.

[150] F. D. Keles, P. M. Wijewardena, and C. Hegde, *On the computational complexity of self-attention*, arXiv preprint arXiv:2209.04881, (2022).

[151] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, *3d gaussian splatting for real-time radiance field rendering*, ACM Transactions on Graphics (ToG), 42 (2023), pp. 1–14.

[152] 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).

[153] 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.

[154] A. Knapitsch, J. Park, Q.-Y. Zhou, and V. Koltun, *Tanks and temples: Benchmarking large-scale scene reconstruction*, ACM Transactions on Graphics (ToG), 36 (2017), pp. 1–13.

[155] N. Komodakis and S. Zagoruyko, *Paying more attention to attention: improving the performance of convolutional neural networks via attention transfer*, in ICLR, 2017.

[156] 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.

[157] S. A. Koohpayegani, K. Navaneet, P. Nooralinejad, S. Kolouri, and H. Pirsiavash, *Nola: Networks as linear combination of low rank random basis*, arXiv preprint arXiv:2310.02556, (2023).

[158] S. A. Koohpayegani and H. Pirsiavash, *Sima: Simple softmax-free attention for vision transformers*, arXiv preprint arXiv:2206.08898, (2022).

[159] S. A. Koohpayegani, A. Singh, K. L. Navaneet, H. Jamali-Rad, and H. Pirsiavash, *Genie: Generative hard negative images through diffusion*, 2024.

[160] S. A. Koohpayegani, A. Tejankar, and H. Pirsiavash, *Compress: Self-supervised learning by compressing representations*, Advances in neural information processing systems, (2020).

[161] ———, *Compress: Self-supervised learning by compressing representations*, NeurIPS, (2020).

[162] S. A. Koohpayegani, A. Tejankar, and H. Pirsiavash, *Mean shift for self-supervised learning*, 2021.

[163] 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.

[164] 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.

[165] R. Krishnamoorthi, *Quantizing deep convolutional networks for efficient inference: A whitepaper*, arXiv preprint arXiv:1806.08342, (2018).

[166] A. Krizhevsky, *Learning multiple layers of features from tiny images*, tech. rep., University of Toronto, 2009.

[167] A. Krizhevsky, G. Hinton, et al., *Learning multiple layers of features from tiny images*, (2009).

[168] ———, *Learning multiple layers of features from tiny images*, (2009).

[169] A. Krizhevsky, V. Nair, and G. Hinton, *The cifar-10 dataset*, online: http://www. cs. toronto. edu/kriz/cifar. html, 55 (2014).

[170] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *Imagenet classification with deep convolutional neural networks*, in NeurIPS, vol. 25, 2012, pp. 1097–1105.

[171] 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.

[172] 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.

[173] 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).

[174] 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.

[175] 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.

[176] J. C. Lee, D. Rho, X. Sun, J. H. Ko, and E. Park, *Compact 3d gaussian representation for radiance field*, arXiv preprint arXiv:2311.13681, (2023).

[177] 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.

[178] Y. Y. Lee and J. W. Woods, *Motion vector quantization for video coding*, IEEE Transactions on Image Processing, 4 (1995), pp. 378–382.

[179] 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.

[180] E. Levin and M. Fleisher, *Accelerated learning in layered neural networks*, Complex systems, 2 (1988), p. 3.

[181] C. Li, H. Farkhoor, R. Liu, and J. Yosinski, *Measuring the intrinsic dimension of objective landscapes*, in International Conference on Learning Representations, 2018.

[182] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, *Pruning filters for efficient convnets*, arXiv preprint arXiv:1608.08710, (2016).

[183] L. Li, Z. Shen, Z. Wang, L. Shen, and L. Bo, *Compressing volumetric radiance fields to 1 mb*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2023, pp. 4222–4231.

[184] L. Li, Z. Shen, Z. Wang, L. Shen, and P. Tan, *Streaming radiance fields for 3d video synthesis*, Advances in Neural Information Processing Systems, 35 (2022), pp. 13485–13498.

[185] 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.

[186] X. L. Li and P. Liang, *Prefix-Tuning: Optimizing Continuous Prompts for Generation*, arXiv:2101.00190 [cs], (2021).

[187] 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.

[188] 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.

[189] 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).

[190] T. Lin, S. U. Stich, L. Barba, D. Dmitriev, and M. Jaggi, *Dynamic model pruning with feedback*, arXiv preprint arXiv:2006.07253, (2020).

[191] 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.

[192] 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.

[193] J. Liu, Z. Pan, H. He, J. Cai, and B. Zhuang, *Ecoformer: Energy-saving attention with linear complexity*, arXiv preprint arXiv:2209.09004, (2022).

[194] 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.

[195] 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.

[196] ———, *Swin transformer: Hierarchical vision transformer using shifted windows*, in Proceedings of the IEEE/CVF international conference on computer vision, 2021, pp. 10012–10022.

[197] I. Loshchilov and F. Hutter, *Decoupled weight decay regularization*, arXiv preprint arXiv:1711.05101, (2017).

[198] 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).

[199] W. Lu, J. Jiao, and R. Zhang, *Twinbert: Distilling knowledge to twin-structured bert models for efficient retrieval*, arXiv preprint arXiv:2002.06275, (2020).

[200] 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).

[201] R. K. Mahabadi, J. Henderson, and S. Ruder, *Compacter: Efficient low-rank hypercomplex adapter layers*, 2021.

[202] 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.

[203] S. Maji, E. Rahtu, J. Kannala, M. Blaschko, and A. Vedaldi, *Fine-grained visual classification of aircraft*, in arXiv preprint arXiv:1306.5151, 2013.

[204] ———, *Fine-grained visual classification of aircraft*, arXiv preprint arXiv:1306.5151, (2013).

[205] J. Makhoul, S. Roucos, and H. Gish, *Vector quantization in speech coding*, Proceedings of the IEEE, 73 (1985), pp. 1551–1588.

[206] 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).

[207] 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.

[208] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, *Nerf: Representing scenes as neural radiance fields for view synthesis*, in Proceedings of the European Conference on Computer Vision (ECCV), 2020.

[209] I. Misra and L. v. d. Maaten, *Self-supervised learning of pretext-invariant representations*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 6707–6717.

[210] I. Misra and L. van der Maaten, *Self-supervised learning of pretext-invariant representations*, arXiv preprint arXiv:1912.01991, (2019).

[211] S. Mohamadi, G. Mujtaba, N. Le, G. Doretto, and D. A. Adjeroh, *Chatgpt in the age of generative ai and large language models: a concise survey*, arXiv preprint arXiv:2307.04251, (2023).

[212] W. Morgenstern, F. Barthel, A. Hilsmann, and P. Eisert, *Compact 3d scene representation via self-organizing gaussian grids*, arXiv preprint arXiv:2312.13299, (2023).

[213] T. Müller, A. Evans, C. Schied, and A. Keller, *Instant neural graphics primitives with a multiresolution hash encoding*, ACM Transactions on Graphics (ToG), 41 (2022), pp. 1–15.

[214] R. Müller, S. Kornblith, and G. Hinton, *When does label smoothing help?*, 2020.

[215] 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).

[216] 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).

[217] 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).

[218] 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 British Machine Vision Conference (BMVC), 2021.

[219] 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.

[220] S. Niedermayr, J. Stumpfegger, and R. Westermann, *Compressed 3d gaussian splatting for accelerated novel view synthesis*, arXiv preprint arXiv:2401.02436, (2023).

[221] 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.

[222] ———, *Automated flower classification over a large number of classes*, in 2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing, IEEE, 2008, pp. 722–729.

[223] 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.

[224] 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.

[225] 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.

[226] 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.

[227] J. Novikova, O. Dušek, and V. Rieser, *The e2e dataset: New challenges for end-to-end generation*, arXiv preprint arXiv:1706.09254, (2017).

[228] 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.

[229] 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.

[230] 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.

[231] 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.

[232] O. M. Parkhi, A. Vedaldi, A. Zisserman, and C. Jawahar, *Cats and dogs*, in 2012 IEEE conference on computer vision and pattern recognition, IEEE, 2012, pp. 3498–3505.

[233] O. M. Parkhi, A. Vedaldi, A. Zisserman, and C. V. Jawahar, *Cats and dogs*, in Computer Vision and Pattern Recognition, 2012.

[234] 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.

[235] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, *Pytorch: An imperative style, high-performance deep learning library*, in Advances in Neural Information Processing Systems 32, 2019, pp. 8024–8035.

[236] 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.

[237] 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.

[238] S. Peng, C. Jiang, Y. Liao, M. Niemeyer, M. Pollefeys, and A. Geiger, *Shape as points: A differentiable poisson solver*, Advances in Neural Information Processing Systems, 34 (2021), pp. 13032–13044.

[239] E. Penner and L. Zhang, *Soft 3d reconstruction for view synthesis*, ACM Transactions on Graphics (TOG), 36 (2017), pp. 1–11.

[240] J. Pfeiffer, A. Kamath, A. Rücklé, K. Cho, and I. Gurevych, *Adapterfusion: Non-destructive task composition for transfer learning*, 2021.

[241] 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.

[242] A. Polino, R. Pascanu, and D. Alistarh, *Model compression via distillation and quantization*, arXiv preprint arXiv:1802.05668, (2018).

[243] S. Purushwalkam Shiva Prakash and A. Gupta, *Demystifying contrastive self-supervised learning: Invariances, augmentations and dataset biases*, Advances in Neural Information Processing Systems, 33 (2020).

[244] 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.

[245] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, et al., *Improving language understanding by generative pre-training*, (2018).

[246] Y. Rao, W. Zhao, B. Liu, J. Lu, J. Zhou, and C.-J. Hsieh, *Dynamicvit: Efficient vision transformers with dynamic token sparsification*, Advances in neural information processing systems, 34 (2021).

[247] Y. Rao, W. Zhao, Z. Zhu, J. Lu, and J. Zhou, *Global filter networks for image classification*, in Advances in Neural Information Processing Systems (NeurIPS), 2021.

[248] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, *Xnor-net: Imagenet classification using binary convolutional neural networks*, 2016.

[249] 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.

[250] A. Razavi, A. Van den Oord, and O. Vinyals, *Generating diverse high-fidelity images with vq-vae-2*, Advances in neural information processing systems, 32 (2019).

[251] S.-A. Rebuffi, H. Bilen, and A. Vedaldi, *Learning multiple visual domains with residual adapters*, arXiv:1705.08045 [cs, stat], (2017). arXiv: 1705.08045.

[252] 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.

[253] C. Reiser, S. Peng, Y. Liao, and A. Geiger, *Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps*, in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021, pp. 14335–14345.

[254] 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.

[255] 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.

[256] G. Riegler and V. Koltun, *Free view synthesis*, in Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIX 16, Springer, 2020, pp. 623–640.

[257] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, *Fitnets: Hints for thin deep nets*, ICLR, (2015).

[258] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning representations by back-propagating errors*, nature, 323 (1986), pp. 533–536.

[259] 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*, IJCV, (2015).

[260] 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.

[261] A. Rücklé, G. Geigle, M. Glockner, T. Beck, J. Pfeiffer, N. Reimers, and I. Gurevych, *Adapterdrop: On the efficiency of adapters in transformers*, 2020.

[262] 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.

[263] R. Salakhutdinov and G. Hinton, *Learning a nonlinear embedding by preserving class neighbourhood structure*, in Artificial Intelligence and Statistics, PMLR, 2007, pp. 412–419.

[264] 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.

[265] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, *Mobilenetv2: Inverted residuals and linear bottlenecks*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 4510–4520.

[266] 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).

[267] F. Schroff, D. Kalenichenko, and J. Philbin, *Facenet: A unified embedding for face recognition and clustering*, in CVPR, 2015, pp. 815–823.

[268] C. Schuhmann, R. Beaumont, R. Vencu, C. Gordon, R. Wightman, M. Cherti, T. Coombes, A. Katta, C. Mullis, M. Wortsman, et al., *Laion-5b: An open large-scale dataset for training next generation image-text models*, Advances in Neural Information Processing Systems, 35 (2022), pp. 25278–25294.

[269] K. Schwarz, A. Sauer, M. Niemeyer, Y. Liao, and A. Geiger, *Voxgraf: Fast 3d-aware image synthesis with sparse voxel grids*, Advances in Neural Information Processing Systems, 35 (2022), pp. 33999–34011.

[270] 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.

[271] 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.

[272] A. Singh, R. Hu, V. Goswami, G. Couairon, W. Galuba, M. Rohrbach, and D. Kiela, *Flava: A foundational language and vision alignment model*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 15638–15650.

[273] V. Sitzmann, J. Thies, F. Heide, M. Niessner, G. Wetzstein, and M. Zollhofer, *Deepvoxels: Learning persistent 3d feature embeddings*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 2437–2446.

[274] J. Snell, K. Swersky, and R. S. Zemel, *Prototypical networks for few-shot learning*, arXiv preprint arXiv:1703.05175, (2017).

[275] 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.

[276] 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).

[277] K. Soomro, A. R. Zamir, and M. Shah, *Ucf101: A dataset of 101 human actions classes from videos in the wild*, 2012.

[278] S. Sukhbaatar, J. Bruna, M. Paluri, L. Bourdev, and R. Fergus, *Training convolutional networks with noisy labels*, 2015.

[279] C. Sun, M. Sun, and H.-T. Chen, *Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 5459–5469.

[280] 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.

[281] ———, *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.

[282] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, *Rethinking the inception architecture for computer vision*, 2015.

[283] 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).

[284] T. Takikawa, A. Evans, J. Tremblay, T. Müller, M. McGuire, A. Jacobson, and S. Fidler, *Variable bitrate neural fields*, in ACM SIGGRAPH 2022 Conference Proceedings, 2022, pp. 1–9.

[285] M. Tan and Q. Le, *Efficientnet: Rethinking model scaling for convolutional neural networks*, in International Conference on Machine Learning (ICML), 2019.

[286] J. Tang, X. Chen, J. Wang, and G. Zeng, *Compressible-composable nerf via rank-residual decomposition*, in Advances in Neural Information Processing Systems, 2022.

[287] 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.

[288] Technology Innovation Institute, *falcon-refinedweb (revision 184df75)*, 2023.

[289] 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.

[290] A. Tejankar, S. A. Koohpayegani, V. Pillai, P. Favaro, and H. Pirsiavash, *Isd: Self-supervised learning by iterative similarity distillation*, 2020.

[291] J. Thies, M. Zollhöfer, and M. Niessner, *Deferred neural rendering: Image synthesis using neural textures*, Acm Transactions on Graphics (TOG), 38 (2019), pp. 1–12.

[292] B. Thomee, D. A. Shamma, G. Friedland, B. Elizalde, K. Ni, D. Poland, D. Borth, and L.-J. Li, *Yfcc100m: The new data in multimedia research*, Communications of the ACM, 59 (2016), pp. 64–73.

[293] Y. Tian, D. Krishnan, and P. Isola, *Contrastive multiview coding*, arXiv preprint arXiv:1906.05849, (2019).

[294] Y. Tian, D. Krishnan, and P. Isola, *Contrastive multiview coding*, in ECCV, 2019.

[295] Y. Tian, D. Krishnan, and P. Isola, *Contrastive representation distillation*, ICLR, (2020).

[296] 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).

[297] R. Tiwari, U. Bamba, A. Chavan, and D. K. Gupta, *Chipnet: Budget-aware pruning with heaviside continuous approximations*, arXiv preprint arXiv:2102.07156, (2021).

[298] 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, PMLR, 2021, pp. 10347–10357.

[299] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, *Training data-efficient image transformers and distillation through attention*, 2021.

[300] 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.

[301] H. Touvron, A. Sablayrolles, M. Douze, M. Cord, and H. Jégou, *Grafit: Learning fine-grained image representations with coarse labels*, 2020.

[302] Y.-H. H. Tsai, T. Li, W. Liu, P. Liao, R. Salakhutdinov, and L.-P. Morency, *Integrating auxiliary information in self-supervised learning*, 2021.

[303] A. van den Oord, Y. Li, and O. Vinyals, *Representation learning with contrastive predictive coding*, 2018.

[304] A. Van Den Oord, O. Vinyals, et al., *Neural discrete representation learning*, Advances in neural information processing systems, 30 (2017).

[305] V. Vanhoucke, A. Senior, and M. Z. Mao, *Improving the speed of neural networks on cpus*, (2011).

[306] 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.

[307] 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.

[308] 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.

[309] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra, *Matching networks for one shot learning*, 2017.

[310] F. Wang, H. Liu, D. Guo, and S. Fuchun, *Unsupervised representation learning by invariance propagation*, in Advances in Neural Information Processing Systems, 2020.

[311] G. Wang, K. Wang, G. Wang, P. H. S. Torr, and L. Lin, *Solving inefficiency of self-supervised representation learning*, 2021.

[312] G.-H. Wang, Y. Ge, and J. Wu, *In defense of feature mimicking for knowledge distillation*, arXiv preprint arXiv:2011.01424, (2020).

[313] H. Wang, C. Qin, Y. Zhang, and Y. Fu, *Neural pruning via growing regularization*, arXiv preprint arXiv:2012.09243, (2020).

[314] L. Wang, J. Zhang, X. Liu, F. Zhao, Y. Zhang, Y. Zhang, M. Wu, J. Yu, and L. Xu, *Fourier plenoctrees for dynamic radiance field rendering in real-time*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 13524–13534.

[315] 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.

[316] 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.

[317] C. Wei, H. Wang, W. Shen, and A. Yuille, *Co2: Consistent contrast for unsupervised visual representation learning*, arXiv preprint arXiv:2010.02217, (2020).

[318] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, et al., *Emergent abilities of large language models*, arXiv preprint arXiv:2206.07682, (2022).

[319] 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.

[320] 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.

[321] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona, *Caltech-ucsd birds 200*, (2010).

[322] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, *Learning structured sparsity in deep neural networks*, arXiv preprint arXiv:1608.03665, (2016).

[323] R. Whitman, *Timm*. https://github.com/huggingface/pytorch-image-models/tree/main/timm. [Online; accessed 28-Sep-2023].

[324] 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.

[325] X. Wu, J. Xu, Z. Zhu, H. Bao, Q. Huang, J. Tompkin, and W. Xu, *Scalable neural indoor scene rendering*, ACM Transactions on Graphics (TOG), (2022).

[326] Z. Wu, A. A. Efros, and S. X. Yu, *Improving generalization via scalable neighborhood component analysis*, 2018.

[327] 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.

[328] J. Xiao, J. Hays, K. A. Ehinger, A. Oliva, and A. Torralba, *Sun database: Large-scale scene recognition from abbey to zoo*, in 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2010, pp. 3485–3492.

[329] 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.

[330] Q. Xie, Z. Dai, E. Hovy, M.-T. Luong, and Q. V. Le, *Unsupervised data augmentation for consistency training*, NeurIPS, (2020).

[331] S. Xie, C. Sun, J. Huang, Z. Tu, and K. Murphy, *Rethinking spatiotemporal feature learning: Speed-accuracy trade-offs in video classification*, 2018.

[332] 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.

[333] Q. Xu, Z. Xu, J. Philip, S. Bi, Z. Shu, K. Sunkavalli, and U. Neumann, *Point-nerf: Point-based neural radiance fields*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 5438–5448.

[334] Y. Xu, Q. Qian, H. Li, R. Jin, and J. Hu, *Weakly supervised representation learning with coarse labels*, 2021.

[335] 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.

[336] 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.

[337] 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.

[338] 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.

[339] A. YM., R. C., and V. A., *Self-labelling via simultaneous clustering and representation learning*, in International Conference on Learning Representations, 2020.

[340] A. Yu, R. Li, M. Tancik, H. Li, R. Ng, and A. Kanazawa, *Plenoctrees for real-time rendering of neural radiance fields*, in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021, pp. 5752–5761.

[341] 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.

[342] X. Yu, Y. Rao, Z. Wang, Z. Liu, J. Lu, and J. Zhou, *Pointr: Diverse point cloud completion with geometry-aware transformers*, in IEEE/CVF International Conference on Computer Vision (ICCV), 2021.

[343] 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.

[344] 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.

[345] C. Zach, T. Pock, and H. Bischof, *A duality based approach for realtime tv-l1 optical flow*, in DAGM-Symposium, 2007.

[346] S. Zagoruyko and N. Komodakis, *Wide residual networks*, in British Machine Vision Conference 2016, British Machine Vision Association, 2016.

[347] 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).

[348] 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.

[349] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, *mixup: Beyond empirical risk minimization*, 2018.

[350] 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.

[351] Q. Zhang, M. Chen, A. Bukharin, P. He, Y. Cheng, W. Chen, and T. Zhao, *Adaptive budget allocation for parameter-efficient fine-tuning*, 2023.

[352] R. Zhang, P. Isola, and A. A. Efros, *Colorful image colorization*, in European conference on computer vision, Springer, 2016, pp. 649–666.

[353] Y. Zhang, K. Zhou, and Z. Liu, *Neural prompt search*, 2022.

[354] Z. Zhang and M. R. Sabuncu, *Generalized cross entropy loss for training deep neural networks with noisy labels*, arXiv preprint arXiv:1805.07836, (2018).

[355] H. Zhao, L. Jiang, J. Jia, P. Torr, and V. Koltun, *Point transformer*, in IEEE/CVF International Conference on Computer Vision (ICCV), 2021.

[356] 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.

[357] 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.

[358] 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).

[359] T. Zhou, S. Tulsiani, W. Sun, J. Malik, and A. A. Efros, *View synthesis by appearance flow*, in Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14, Springer, 2016, pp. 286–301.

[360] 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.