UNIVERSITY OF CALIFORNIA,
IRVINE

Theoretical and Applied Deep Learning for the Physical Sciences

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Computer Science

by

Jordan Andrew Melchionne Ott

Dissertation Committee:
Professor Pierre Baldi, Chair
Professor Franklin Dollar
Professor Roy Fox

2022

DEDICATION

To my mother who taught me to work
My father who taught me to play
My brother who taught me to fight
And my fiancée who taught me to love

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I wish to acknowledge financial support from the National Science Foundation grant NRT 1633631 and the hardware contributions from NVIDIA.

I wish to thank all co-authors of those publications in enabling the research which forms the basis for this dissertation. I would also like to highlight the members of Chapman University's MLAT lab for igniting my academic journey.

Portions of Chapter 2 were previously published as "Learning in the Machine: To Share or Not to Share? Journal of Neural Networks, June 2020. Jordan Ott, Erik Linstead, Nicholas LaHaye, Pierre Baldi". Permissions to reuse the text were granted by Elsevier and co-authors. Portions of Chapter 3 were previously published as "A Fortran-Keras Deep Learning Bridge for Scientific Computing, Scientific Programming, August 2020. Jordan Ott, Mike Pritchard, Natalie Best, Erik Linstead, Milan Curcic, and Pierre Baldi". Permissions to reuse the text were granted by Hindawi and co-authors. Portions of Chapter 4 were previously published as "An end-to-end CNN with attentional mechanism applied to raw EEG in a BCI classification task, Journal of Neural Engineering, August 2021. Elnaz Lashgari, Jordan Ott, Akima Connelly, Pierre Baldi, and Uri Maoz". Permissions to reuse the text were granted by IOP and co-authors.

Finally, I would like to thank my adviser Pierre Baldi for providing this opportunity and his guidance and support.

VITA

Jordan Andrew Melchionne Ott

EDUCATION

Doctor of Philosophy in Computer Science **2022**
University of California, Irvine *Irvine, California*

Master of Science in Computational and Data Sciences **2018**
Chapman University *Orange, California*

Bachelor of Science in Computer Science **2017**
Chapman University *Orange, California*

RESEARCH EXPERIENCE

Graduate Research Assistant **2018–2022**
University of California, Irvine *Irvine, California*

TEACHING EXPERIENCE

Lecturer **2018–2020**
Chapman University *Orange, California*

Teaching Assistant **2018–2019**
University of California, Irvine *Irvine, California*

PROFESSIONAL EXPERIENCE

Senior Computer Vision Engineer **March 2022–Present**
Path Robotics *Columbus, Ohio*

Computer Vision Engineer **October 2020–March 2022**
Path Robotics *Columbus, Ohio*

REFEREED JOURNAL PUBLICATIONS

Real-time reconstruction of intense, ultrafast laser pulses using deep learning Nature Scientific Reports	**2022**
An end-to-end CNN with attentional mechanism applied to raw EEG in a BCI classification task Journal of Neural Engineering	**2021**
Assessing the Potential of Deep Learning for Emulating Cloud Superparameterization in Climate Models with Real-Geography Boundary Conditions JAMES	**2021**
Detecting Pulmonary Coccidioidomycosis with Deep Convolutional Neural Networks Machine Learning with Applications	**2021**
Enforcing analytic constraints in neural-networks emulating physical systems The Journal of Physical Review Letters	**2021**
A Fortran-Keras deep learning bridge for scientific computing Scientific Computing	**2020**
Sherpa: Robust hyperparameter optimization for machine learning SoftwareX	**2020**
The Pretense of Knowledge Cosmos+Taxis	**2020**
Learning in the machine: To share or not to share? Neural Networks	**2020**

REFEREED CONFERENCE PUBLICATIONS

Deep-Learning-Based Kinematic Reconstruction for DUNE NIPS	**2020**

PUBLICATIONS UNDER REVIEW / IN PREPARATION

Deducing Neutron Star Equation of State From Telescope Spectra with Machine Learning **2022**

Deep-Learning-Based Kinematic Reconstruction for DUNE **2022**

Polynomial Neural Networks: Review and Progress **2022**

SOFTWARE

Fortran-Keras Bridge (FKB) `https://github.com/scientific-computing/FKB`
A Fortran-Keras Deep Learning Bridge for Scientific Computing.

ABSTRACT OF THE DISSERTATION

Theoretical and Applied Deep Learning for the Physical Sciences

By

Jordan Andrew Melchionne Ott

Doctor of Philosophy in Computer Science

University of California, Irvine, 2022

Professor Pierre Baldi, Chair

Weight-sharing is one of the pillars behind Convolutional Neural Networks and their successes. However, in physical neural systems such as the brain, weight-sharing is implausible. Taking inspiration from neural circuitry, we explore the use of Free Convolutional Networks and neurons with variable connection patterns. Using Free Convolutional Networks, we show that while weight-sharing is a pragmatic optimization approach, it is not a necessity in computer vision applications.

Second, we turn to an application of artificial networks for scientific computing. Implementing artificial neural networks is commonly achieved via high-level programming languages such as Python and easy-to-use deep learning libraries such as Keras. As a result, a deep learning practitioner will favor training a neural network model in Python, where these tools are readily available. However, many large-scale scientific computation projects are written in Fortran, making it difficult to integrate with modern deep learning methods. To alleviate this problem, we introduce a software library, the Fortran-Keras Bridge in order to connect environments where deep learning resources are plentiful with those where they are scarce.

Finally, we examine an application of neural networks to activity classification. Using raw EEG signals we demonstrate superior performance of our novel attention based network to predict a subject's motor execution.

Chapter 1

Introduction

1.1 Contributions

1.1.1 Climate Modeling

In climate modeling we introduced a systematic way of enforcing nonlinear analytic constraints in neural networks via constraints in the architecture or the loss function [32]. This was applied to convective processes for climate modeling, architectural constraints enforce conservation laws to within machine precision without degrading performance. In the real geography scenario, we demonstrated the ability of neural networks to parameterize convection dynamics over continents [128]. This allowed for accurate emulation of climate models locally in space and time.

1.1.2 Software Libraries

Implementing artificial neural networks is commonly achieved via high-level programming languages such as Python and easy-to-use deep learning libraries such as Keras. These software libraries come preloaded with a variety of network architectures, provide autodifferentiation, and support GPUs for fast and efficient computation. As a result, a deep learning practitioner will favor training a neural network model in Python, where these tools are readily available. However, many large-scale scientific computation projects are written in Fortran, making it difficult to integrate with modern deep learning methods. To alleviate this problem, we introduce a software library, the Fortran-Keras Bridge (FKB). This two-way bridge connects environments where deep learning resources are plentiful with those where they are scarce [143].

When implementing neural networks there are a number of choices - hyperparameters - that must be set prior to training. Choosing hyperparameters arbitrarily can lead to suboptimal results. We introduced SHERPA [77], a Python library for hyperparameter tuning.

1.1.3 Physics

In high energy particle physics, the Deep Underground Neutrino Experiment (DUNE) is a next-generation long-baseline neutrino oscillation experiment based on liquid argon TPC (LArTPC) technology. While LArTPC technology provides excellent spatial resolution, high neutrino detection efficiency, and superb background rejection, it poses significant reconstruction challenges. We design, train, and test two CNN-based methods, using 2-D and 3-D LArTPC pixelmaps, for the reconstruction of final state particle direction and energy, as well as neutrino energy, from both raw event pixelmaps and clustered shower and tracks. This proves that deep neural networks can automatically obtain individual particle information from a global pixelmap without human engineered clustering. Combining our models with particle masses yields a fully AI-based reconstruction chain producing the four-vector momenta of the final state particles. Improvements in energy and direction reconstruction for electrons, muons, ν_μ CC, and ν_e CC events demonstrate that quality and robustness of the approach across multiple reconstruction tasks [115].

In optics, we introduced a method for the phase reconstruction of an ultrashort laser pulse based on the parameterization of the nonlinear spectral changes induce by self-phase modulation [181].

1.1.4 Biomedical Applications

In the domain of brain computer interfaces (BCI), electroencephalography (EEG) is a relatively inexpensive and non-invasive monitoring technique. We demonstrated the use of attention based neural networks, to predict patients motor actions from raw EEG signals [105].

Coccidioidomycosis is the most common systemic mycosis in dogs in the southwestern United States. We demonstrated the effectiveness of automatically detecting the disease in canine subjects, paving the way to help doctors and veterinarians more easily identify and diagnose positive cases [141].

1.1.5 Theoretical Deep Learning

Weight-sharing is one of the pillars behind Convolutional Neural Networks and their successes. However, in physical neural systems such as the brain, weight-sharing is implausible. This discrepancy raises the fundamental question of whether weight-sharing is necessary. Taking inspiration from neural circuitry we assessed the necessity of weight sharing in computer vision applications [142].

Polynomials have a long history in mathematics going back hundreds of years. Their mathematical formulation is accompanied by a vast literature that offers interpretability, potential theorems, and alternative hardware embodiments for computation. This begs the question as to why polynomials have been noticeably absent from the development of deep neural networks. In this paper we investigate the interplay between polynomials and deep networks. Through a series of computer vision simulations we formalize the idea of polynomial neural networks and their various forms. We present insights into the trade off of polynomial degree, training paradigms, as well as polynomial extensions to the standard artificial neuron model.

Chapter 2

Learning in the Machine: To Share or Not to Share?

2.1 Introduction

Digital simulations of neural networks are successful in many applications but rely on a fantasy where neurons and synaptic weights are objects stored in digital computer memories. This fantasy often obfuscates some fundamental principles of computing in native neural systems. To remedy this obfuscation, learning in the machine refers to a general approach for studying neural computations. In this approach, the physical constraints of physical neural systems, such as brains or neuromorphic chips, are taken into consideration. When applied to single neurons, learning in the machine can lead, for instance, to the discovery of dropout [179, 19]. When applied to synapses, learning in the machine can lead, for instance, to the discovery of local learning [21] and random backpropagation [113, 18, 17]. Moreover, when applied to layers of neurons, as we do in this paper, learning in the machine leads one to question the fundamental assumption of weight-sharing behind convolutional neural networks (CNNs).

The technique of weight-sharing, whereby different synaptic connections share the same strength, is a widely used and successful technique in neural networks and deep learning. This paradigm is particularly true in computer vision where weight-sharing is one of the pillars behind convolutional neural networks and their successes. In any physical neural system, for instance, carbon- or silicon-based, exact sharing of connections strengths over spatial distances is difficult to realize, especially on a massive 3D scale. In physical systems, not only is it difficult to create identical weights at a given time point, but it is also challenging to maintain the identity over time. During phases of development and learning the weights may be changing rapidly. During more mature stages weights must retain their integrity against the microscopic, entropic forces surrounding any physical synapse. Furthermore, given the exquisitely complex geometry of neuronal dendritic trees and axon arborizations, it is implausible to form large arrays of neurons with identically translated connection patterns. In short, not only is it challenging to share weights exactly, but it is also difficult to exactly share the same connection patterns.

While weight-sharing has proven to be very useful in computer vision and other applications, it is extremely implausible in biological and other physical systems. This discrepancy raises the fundamental question of whether weight-sharing is a strict prerequisite for convolution-based deep learning, or if similar levels of learning are possible without it. In particular, we consider the following research questions:

1. Is weight-sharing necessary to prevent overfitting?
2. Is weight-sharing necessary to ensure translational invariant recognition?
3. Can acceptable classification performance be achieved without weight-sharing?
4. Does approximate or exact weight-sharing emerge in a natural way[1]?

[1]Without explicit constraints in the architecture or imposed constraints in the form of losses

In formulating the research questions above, we considered the most common reasons practitioners give for employing weight sharing in convolutional network architectures. The purpose is to challenge these common points based on the intuitive principle that weight sharing is implausible in biological systems. In total, answers to these questions provide new insight into whether weight sharing is a strict prerequisite for the effective training convolutional architectures, and what happens if the requirement for weight sharing is relaxed. The goal of this study is to investigate these research questions, primarily through simulations where the weight-sharing assumption is relaxed. Some of these questions have been previously considered in the literature in other contexts. We are not the first to utilize locally connected architectures or assess their performance on computer vision tasks [26], for example. However, in aggregate, the answer to these questions provide novel insight into whether weight-sharing is vital for convolutional architectures and whether it can emerge in other ways when connections are not shared.

2.2 Origins and Functions of Weight-Sharing

Before addressing the question of its necessity, it is useful to review the origins and functions of weight-sharing. Hubel and Wiesel's neurophysiological work [83] on the cat visual cortex was the inception of weight-sharing. These experiments suggested the existence of entire arrays of neurons dedicated to implementing simple operations, such as edge detection and other Gabor filters, at all possible image locations. Fukushima systematically used the ideas proposed by Hubel and Wiesel to create the neocognitron [61] in computer vision. Primarily a convolutional neural network architecture with Hebbian learning. However, Hebbian Learning alone applied to a feedforward CNN cannot solve vision tasks [21]. Solving vision tasks requires feedback channels and learning algorithms for transmitting target information to the deep synapses. A job that is precisely achieved by backpropagation, or stochastic gradient descent. Successful CNNs for vision problems, trained via backpropagation, were developed in the late 80s and 90s [49, 16, 170].

Substantial improvements in the size of the training sets and available computing power have led to a new wave of successful implementations in recent years, [100, 183, 180, 74], as well as applications to a variety of specific domains, ranging from biomedical images [45, 71, 199, 198, 57, 191] to particle physics [15, 12, 164] and video analysis [138, 188, 139, 190]. Older [218] as well as more recent work [41, 203] has also shown that not only do convolutional neural networks rival the object category recognition accuracy of the primate cortex but also seem to provide the best match to biological neural responses, at least at some coarse level of analysis.

It is worth pointing out that weight-sharing is sometimes used in other settings, for instance when Siamese Networks are used to process and compare objects [38, 16, 93], which also includes Siamese CNNs for images. Siamese Networks consist of two or more sub-networks that are used to train a contrastive loss function in order to learn the differences between pairs of inputs. In the case of Siamese CNN's, the typical weight sharing paradigm is used between each subnetwork architecture. In addition, the weights of each subnetwork are identical with each other, hence achieving another "level" of weight sharing. Finally, a different kind of weight-sharing that will not concern us here, when a recurrent network is unfolded in time. In this case, weight-sharing occurs over time and not over space [81, 42].

Biologically motivated architectures have become an import focal point of deep learning as of late [26, 24, 23, 168, 113, 22, 137]. Most recently [26] investigated how biologically motivated deep learning algorithms scale to more massive datasets. Locally connected networks were included in their simulations, where each weight kernel interacts with *local* features, not the entire input. However, the focus of the paper is on how learning algorithms such as feedback alignment [113] and target propagation [109] scale to more significant image recognition problems. Not addressed in the paper is the problem of learning translationally invariant representations, overfitting, variable connection patterns, and the emergence of approximate weight-sharing. Nor was there in-depth analysis comparing networks with weight-sharing to those without.

Two primary but different purposes are typically associated with weight-sharing. The first is to reduce the number of free parameters that need to be stored or updated during learning. This requirement can be important in applications where storage space is limited (i.e., cell phones), or where training data is limited, and overfitting is a danger. It is important to remember, however, that in convolutional architectures the local connectivity of neurons contributes at least as much to parameter reduction as weight-sharing. In other words, weight-sharing is not the only way to shrink the parameter space. The second function of weight-sharing is to apply the same operation at different locations of the input data, to process the data uniformly and provide a basis for invariance, typically translation invariant recognition in CNN architectures.

2.3 Free Convolutional Networks

The research questions proposed in Section 1 deal with the necessity of weight sharing and the result of its absence. As a result, it is natural to consider simulations where weights are not shared but instead are maintained for specific regions of the input space. Relaxing the weight-sharing assumption in CNNs yields a Free Convolutional Network (FCN). In FCNs, the weights of a filter at a specific location are not tied to the weights of the same filter at a different location (see Figure 2.1). This naturally means that FCNs have far more parameters than the corresponding CNN and are slower to train on a digital machine. However, this is not a concern here, as our primary goal is not to improve the efficiency of CNNs deployed on digital machines, but rather to understand the consequences of relaxing the weight-sharing assumption inside a native neural machine.

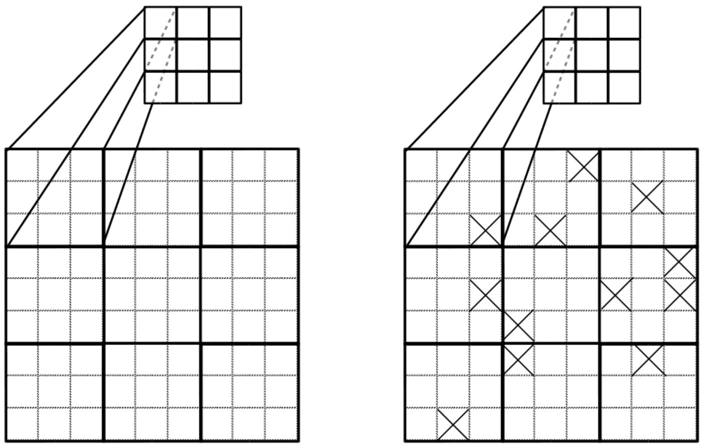


Figure 2.1: Free convolutional layers maintain a separate kernel at each location, unlike typical convolutional layers, that apply the same filter across all possible locations. The above figures are examples of FCN layers on a 9x9 input space. Each 3x3 subregion of the input is covered with a distinct kernel (weight matrix), as shown in the diagram on the left. The top square represents the output obtained from applying the filter to the corresponding input region. The diagram on the right depicts free convolutional layers with variable connection patterns, where the x's represent absent connections. In this example, 12 out of the 91 connections are missing, creating a variable connection probability of roughly 0.15.

Furthermore, it is highly implausible that a given neuron will share the same dendritic tree with a neighboring neuron [86], thus in addition to neighboring neurons of the same layer not having the same weights, we would like to consider also the possibility of them not having the same receptive field pattern. Thus, in addition to plain FCNs, FCNs with variable connection patterns are implemented in the simulations below. The implementation of variable connection patterns has many options. Here, for simplicity, a random percentage of connections are severed (Figure 2.1) [Note: this is very different from dropout where different sets of weights get randomly set to 0 at each presentation of a training example]. Similar to in methodology to DropConnect [197], however, the same weights remain 0 for all of training and testing. The x's in the right image of Figure 2.1 correspond to missing synaptic connections between neurons that are set to zero and never trained.

By running simulations comparing CNNs and FCNs (with and without variable connection patterns), we seek to answer the research questions laid out in the introduction. Appendix

A.3 contains complete details regarding simulations with variable connection patterns. To our knowledge, this is the first systematic study, through large-scale computational simulations, that explores the necessity of weight-sharing in deep convolutional architectures as it pertains to overfitting and performance in the broader context of biological plausibility. Further novelty is found in our assessment of the emergence of approximate weight-sharing in free convolutional architectures with and without variable connection patterns.

2.4 Data and Methods

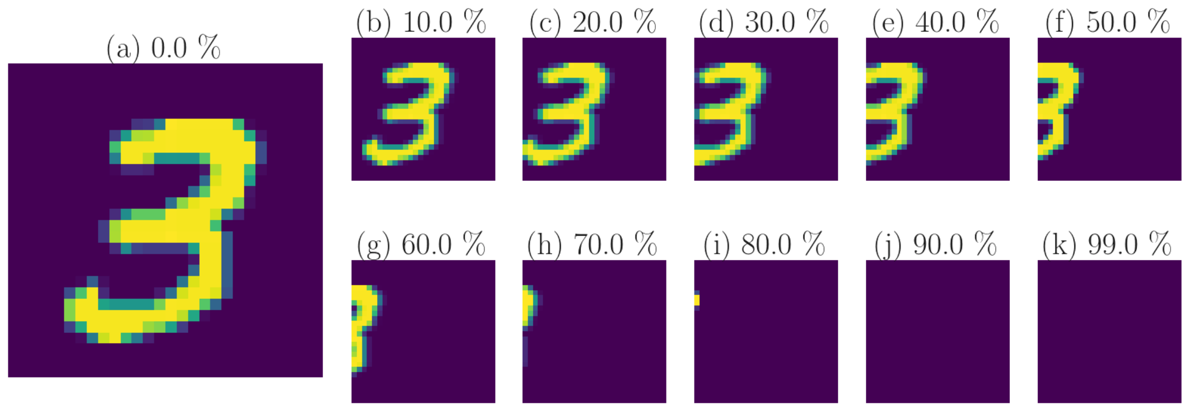


Figure 2.2: Examples of translational augmentation on MNIST. In training images are translated left-right as well as up-down. The above only display the result of continually translating an image leftwards, for a visual example. (a) 0% translation augmentation, equivalent to a un-altered MNIST image. (b-k) Gradually increasing the percentage of augmentation by 10% each time.

In the simulations, we focus exclusively on computer vision tasks. We evaluate various free and shared weight networks on two well-known benchmark datasets: the handwritten digit dataset, MNIST [108], as well as the CIFAR-10 object dataset [99].

In the case of free weights, we consider using data augmentation by translating images horizontally and vertically to potentially compensate for the lack of translation, inherent in the architecture. Due to the local receptivity of free weight networks, individual filters learn features solely within their receptive field. Translationally invariant data will allow filters to learn more or less the same features across the input space. Conversely, in CNNs, weights are shared across space. By applying the same operation across the input space, translational invariance is naturally embedded in the model. This property of CNNs gives them an inherent advantage in vision-based tasks. The purpose of these experiments is not to demonstrate the superiority of one network but to understand the consequences of weight-sharing and properties that arise around it.

Eleven settings of translation were tested in experiments (0% to 99% by increments of 10%). Translational augmentation involves shifting images horizontally and vertically by varying degrees of the width and height respectively. Points outside the boundaries of the input get filled according to the nearest pixels. Figure 2.2 shows examples of translational augmentation results on MNIST. During training, each image presented to the network is translated left-right as well as up-down by random amounts. 0% to the augmentation setting for that trial. e.g., at 90% augmentation, an image may be translated any amount 0-90% up-down as well as 0-90% left-right.

All simulations, each augmentation setting, were completed using five-fold cross-validation. Simulations were implemented in Keras [44] with a Tensorflow [2] backend using NVIDIA GeForce GTX Titan X GPUs with 12 GB memory. For purposes of reproducibility, the code has been made publicly available[2].

2.4.1 Networks

Conducting a grid search yielded appropriate hyperparameter configuration for all networks. See Appendix A.1.1 for complete details. The search produced network architectures com-

[2]`https://github.com/Learning-In-The-Machine/Weight-Sharing`

posed of three convolutional/free convolutional layers, followed by a single hidden layer, and one output layer for classification. A full description of all networks, including filter size, number of filters, stride, and learning rate can be found in Table A.2. MNIST and CIFAR require different architectures as the input sizes differ. For every setting of data augmentation (0 to 99%, increments of 10%) a CNN and FCN were trained via five-fold cross-validation.

For simplicity, the architecture of CNNs and FCNs are equivalent, except that free convolutional layers replace convolutional layers. The activation functions, softmax layer, as well as the number of filters per layer, remain the same across all networks. Table A.2 lists hyperparameter settings of the networks used in this paper. Additionally, it is essential to note that there is no architectural difference between the networks trained with data augmentation and those trained without.

2.4.2 Variable Connection Patterns

Also implemented in this study are neurons with variable connection patterns in FCNs. At the start of training, a chosen percentage of weights are randomly set to 0, representing the absence of a dendritic connection. These missing weights do not contribute to the output of the layer, and their values are never updated during backpropagation. The resulting connection patterns are maintained throughout training and testing. There are multiple options for implementing neurons with variable connection patterns. For computational simplicity, the implementation used in this paper is to turn off connections within each square filter given some probability (depicted in Figure 2.1). In simulations, we vary the drop probability from 0 to 99% by increments of 10%. The results from these simulations are reported in Figure A.8 and A.9 of the Appendix.

2.5 Results

We report accuracy metrics on translationally augmented training (0 - 99%, increments of 10%), un-augmented validation, and translationally augmented validation set (with 25% translation) throughout training. Results for MNIST and CIFAR are shown in Figure 2.3 and 2.4, respectively. The legend denotes the amount of translation used during training (0 - 99%, increments of 10%).

Tables 2.1 and 2.2 show the median accuracy of the five-fold cross validation experiments for the MNIST and CIFAR-10 datasets respectively. The tables display results for each corresponding setting of translation augmentation (0 - 99%, increments of 10%). Bold values indicate the highest performing model in that accuracy metric for CNN and FCN, respectively. The Appendix contains additional experiments regarding the use of other augmentation methods, additional training metrics, as well as simulations regarding neurons with variable connection patterns.

	Training Accuracy		Validation Accuracy		Translation Accuracy	
Aug %	CNN	FCN	CNN	FCN	CNN	FCN
0.00	**0.999964**	**0.999175**	0.990536	0.988071	0.366536	0.372287
0.10	0.997793	0.995665	**0.993071**	**0.991857**	0.837963	0.813368
0.20	0.996719	0.991730	0.993000	0.990357	0.986220	0.977033
0.30	0.992555	0.983998	0.992143	0.987643	**0.990741**	**0.985062**
0.40	0.976966	0.965521	0.991571	0.984500	0.989439	0.982422
0.50	0.935973	0.923703	0.990071	0.981571	0.987594	0.979456
0.60	0.863680	0.847367	0.988286	0.978500	0.986111	0.976273
0.70	0.752630	0.731917	0.987000	0.974286	0.984484	0.971933
0.80	0.626956	0.606541	0.985500	0.970786	0.982820	0.968171
0.90	0.520896	0.503312	0.984714	0.967821	0.982096	0.965133
0.99	0.448491	0.431218	0.984000	0.965071	0.981120	0.961372

Table 2.1: MNIST results. Median accuracies of training, un-augmented validation, and augmented validation set. The left most column denotes the amount of translational augmentation used during training. When performing translational augmentation on the validation set, 25% augmentation was used throughout the experiments. Bold values indicate the highest performing model in that accuracy metric for CNN and FCN, respectively.

	Training Accuracy		Validation Accuracy		Translation Accuracy	
Aug %	CNN	FCN	CNN	FCN	CNN	FCN
0.00	**1.000000**	**1.000000**	0.671083	0.647333	0.444556	0.433468
0.10	0.982156	0.973354	0.749333	0.724333	0.598538	0.561324
0.20	0.876542	0.858094	0.773250	**0.731500**	0.705225	0.653478
0.30	0.834646	0.781885	**0.774083**	0.720167	0.739163	**0.683972**
0.40	0.775188	0.725198	0.768750	0.704958	**0.742608**	0.680444
0.50	0.712146	0.657438	0.751667	0.682250	0.729419	0.661458
0.60	0.650719	0.598344	0.734042	0.659958	0.714130	0.642557
0.70	0.592385	0.545354	0.717250	0.641542	0.699555	0.624286
0.80	0.535958	0.496677	0.699667	0.626250	0.681452	0.607695
0.90	0.483635	0.449490	0.684833	0.611667	0.667801	0.593834
0.99	0.442260	0.413802	0.668667	0.599458	0.653436	0.583375

Table 2.2: CIFAR results. Median accuracies of training, un-augmented validation, and augmented validation set. The left most column denotes the amount of translational augmentation used during training. When performing translational augmentation on the validation set, 25% augmentation was used throughout the experiments. Bold values indicate the highest performing model in that accuracy metric for CNN and FCN, respectively.

2.5.1 Is weight-sharing necessary to prevent overfitting?

Both FCNs and CNNs are shown to overfit the training set, given a sufficient number of epochs. This result is evident from a divergence in training accuracy vs. validation accuracy over time (MNIST and CIFAR, Figures 2.3 and 2.4 respectively). In situations, without overfitting, one would expect training and validation scores to be close to one another. However, as the training accuracy continues to increase towards 100%, in many cases the validation accuracy declines, the telltale indication of overfitting. In the remaining cases the validation accuracy plateaus while training accuracy continues to grow. The large gap between training and validation performance in these cases suggests that the model's ability to generalize to new data is suffering, and that overfitting is likely taking place. In experiments performed on MNIST and CIFAR overfitting was observed for CNNs and FCNs when translation was not used, Figure 2.3 and 2.4 respectively. This effect manifests in CIFAR (Table 2.2) where there is more than a 30% gap between training accuracy and validation accuracy for both

CNN and FCN, with no translational augmentation (i.e., 0% translation).

In the case of MNIST, we see from Table 2.1 that FCNs, trained on augmented data (10% translation), achieve a median accuracy of 99.1857% on the validation set. Only slightly less than the 99.3071% median accuracy achieved by standard CNNs. On the more complicated CIFAR dataset, with 10% translation, FCNs can come within 2% of CNN accuracy on the validation set (Table 2.2). Thus, we can infer that data augmentation alone is sufficient to prevent overfitting, and weight-sharing need not be leveraged to achieve this.

Using augmentation during training (Figure 2.3 and 2.4) not only reduces overfitting but also leads to an increase in performance on validation sets. For FCNs on CIFAR specifically, training with translational amounts of 10, 20, and 30% increases FCN validation performance while dramatically reducing overfitting, compared to 0% translation. In human learning, translational augmentation comes as a byproduct of interacting with a changing world, which inherently provides the brain with samples of the same object translated at different positions - for example, a car driving down the street. Because FCNs cannot rely on weight-sharing to simulate the effect of translation, they must, therefore, rely on manual augmentation.

Large, translationally invariant datasets, like those produced through data augmentation, are essential for free weight networks to achieve excellent performance and avoid overfitting. If this constraint can be met, overfitting can be mitigated without weight-sharing.

2.5.2 Is weight-sharing necessary to ensure translational invariant recognition?

Learning translationally invariant representations is tested by using translational augmentation on the validation set. Figures 2.3e and 2.3f for MNIST and Figures 2.4e and 2.4f for CIFAR show results of translation on the augmented validation set.

Significant disparity between validation accuracy and translation augmented validation accuracy would signal that the network is not capable of translationally invariant recognition. For example, the FCN trained on MNIST data without augmentation saw more than a sixty percent gap between validation set accuracy and translation augmented validation set accuracy (Table 2.1). Similarly, on CIFAR, the FCN trained without translation augmentation performed 20% worse on the translation augmented validation set than the standard one (Table 2.2). Meaning these networks trained with 0% augmentation are incapable of learning translation invariant representations.

On both MNIST and CIFAR, FCNs achieve comparable results on the validation set and translation augmented validation set. This result is evident on MNIST where an FCN trained with 30% translation achieves a 0.2% difference between validation and translation augmented validation performance. Likewise, on CIFAR, the FCN trained with 40% translation comes within 2% of its validation performance on the translation augmented validation set.

The results show that as translational augmentation is used during training, the gap between the validation accuracy and translation augmented validation accuracy decreases. This confirms that FCNs are capable of learning translationally invariant representations provided sufficient, translationally augmented, data.

2.5.3 Can acceptable performance be achieved without weight-sharing?

FCNs can achieve high-performance scores on two benchmark computer vision datasets, which agrees with what has been noted previously in the literature [26]. In the absence of data augmentation (0% translation), CNNs outperform FCNs on MNIST and CIFAR. Referencing Table 2.1, CNNs achieve 99.05% validation accuracy whereas FCNs are slightly worse with 98.81% validation accuracy for MNIST. On CIFAR, Table 2.2, CNNs outperform FCNs with 67.11% and 64.73%, respectively. When FCNs are exposed to translational data, they can achieve 99.19%, 10% augmentation, and 73.15%, 20% augmentation, on MNIST and CIFAR validation sets, respectively.

The validation set accuracy observed in our simulations shows that as translational augmentation is increased, FCNs can match the performance of CNNs. Thus, there is not a fundamental necessity of weight-sharing to attain satisfactory performance.

2.5.4 Does approximate or exact weight-sharing emerge in a natural way?

To test for the emergence of approximate weight-sharing, we calculate the average euclidean distance between each filter of the FCN layer, at a specified radius. Experiments in this paper use a radius of one and four units to test the similarity of filters at different distances. Figure A.5 in the Appendix, gives a visual example. This depiction means that each filter in the FCN layer is compared to the filters one and four units, respectively, away from it. Figure 2.5 reports the euclidean distance between filters at radius one and radius four.

If weight-sharing did emerge naturally, one would expect to see a lower average euclidean distance between filters within an FCN layer throughout training. One would only expect to see this convergence in weight values if a similar stimulus is present across the input space (i.e., translationally augmented data, akin to video). Simulations conducted on the MNIST dataset confirms this. As the amount of translation increases (indicated by the legend in Figure 2.5), the parameters of the FCN layer converge to similar values; the euclidean distance between filters decreases. Furthermore, a higher degree of translation yields more similar the weight values. Conversely, when translation is not used during training, the weights diverge and become less similar from each other over time (i.e., the distance between them increases).

One would also expect filters near one another to be more similar than those farther away (i.e., filters at radius 1 should be more similar than at radius 4). Again, this is confirmed in simulations where Figure 2.5a shows lower average Euclidean distance values, per augmentation setting, compared to Figure 2.5b.

To ascertain the cause of approximate weight-sharing, FCNs were trained with noise and rotation augmentation while measuring the distance between their filters. Figure A.6 and A.7 display euclidean distance results for these experiments. These metrics confirm that only translation augmentation is sufficient to endow FCNs with approximate weight-sharing. Additional explanations are provided in Appendix A.2.5.

While not exact, the distance between FCN filters shows how approximate weight-sharing can emerge in a natural way with translationally augmented data.

2.5.5 For What Learning Tasks Are Free Convolutional Networks Most Applicable?

In CNNs, the same filter is applied to all locations in the input space. One may hypothesize that, as a result, CNNs will be focused less on the overall structure of features in spatial relation to each other, and more on identifying the features themselves. This is in contrast to FCNs, which can only operate locally within their receptive field, and so must inherently learn the global structure of features in relation to one another.

To test this hypothesis, images presented to the networks for testing need to be manipulated in such a way so as to compromise the overall global structure of the image while at the same time preserving individual features appearing within an image. Appendix A.2.3 details the quadrant swap task, in which quadrants I and III of images are interchanged with each other in the validation set. See Figure A.3 for a visual example. In this task, lower classification accuracy would indicate a higher importance placed on global structure during training. That is, it is not sufficient for learned features just to be present in an image, but the spatial relationship between them must also be preserved to some degree.

The results from the quadrant swap task are shown in Figure 2.6. One might expect severe degradation of performance after rearranging parts on an image. However, the CNN performs at nearly 50% accuracy. This performance indicates that the CNN is still able to recognize features of the altered images when making its prediction. Conversely, the FCN performs nearly 20% worse. Higher accuracy bolsters the notion that the overall global structure of an image is less important for a CNN as opposed to the FCN. *In domains where global structure is essential such as face recognition or biomedical imaging, FCNs may find substantial applicability.*

2.5.6 Is weight-sharing necessary?

The necessity of weight-sharing arises as a means of parameter reduction. Reducing the number of parameters leads to networks that are faster to train and smaller to store. Due to the limitations of modern hardware, practical applications of free weight networks are not currently viable, especially in embedded environments. Thus in situations where space is a constraint, and translationally invariant datasets are not available, weight-sharing becomes a necessity.

In terms of accuracy, FCNs have shown comparable results are possible without weight-sharing. In this regard, weight-sharing is not a necessity.

2.5.7 If weight-sharing is not necessary, are translational invariant training sets necessary?

To examine the necessity of translational datasets, additional experiments were conducted using noise and rotational augmentation during training. Intuition would suggest that only translational data is sufficient to endow FCNs with translational invariant recognition. The full results, including accuracy performance on the noise and rotation, augmented training, un-augmented validation, rotation augmented validation, noise augmented validation, and translation augmented validation set can be found in Appendix A.2.

Training using other augmentation methods that do not produce translational invariant datasets yield poor results when testing on the translationally augmented validation set. Referencing Appendix A.2 (Table A.3, A.4, and A.5), the results show that FCNs trained on non-translational data are consistently not capable of learning translationally invariant representations. Low performance indicates that only translationally augmented training data allows FCNs to learn translationally invariant representations. *Using translationally invariant datasets yields better results in validation set and augmented validation set accuracy, specifically in FCNs.*

2.6 Conclusion

The use of weight-sharing arises as a solution to parameter reduction and translationally invariant recognition in neural networks. Though weight-sharing is implausible in any biological or physical setting, it is instrumental in computer vision tasks. We have examined alternatives to weight-sharing, such as free convolutional networks, where the weight-sharing assumption is relaxed. FCNs trained with augmented datasets have been shown to match and even surpass standard CNNs in validation set accuracy. Data augmentation, specifically datasets augmented via translation, is a necessity as a means to avoid overfitting and train FCNs capable of translationally invariant recognition. Thus, in environments where data is plentiful and computational resources can cope with the large number of parameters that result from abandoning weight-sharing, FCNs provide an alternative to CNNs that can achieve potentially superior performance and higher fidelity to physical systems.

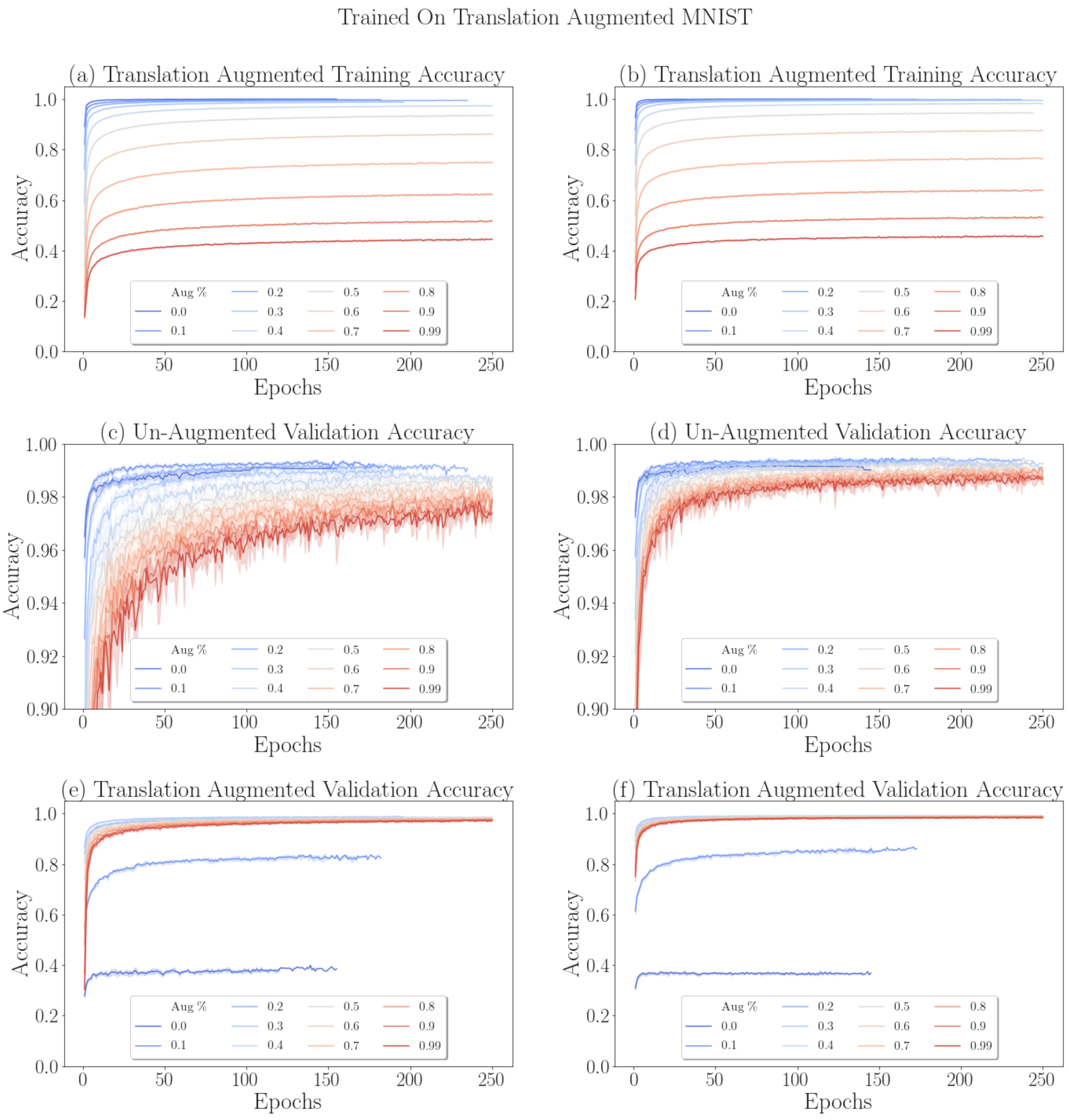


Figure 2.3: MNIST results. Shown above are FCN (left column) and CNN (right column) results trained with varying degrees of translational augmentation, indicated by the legend. Top row: training accuracy over time. Middle row: validation accuracy over time. Bottom row: accuracy on the translationally augmented validation set.

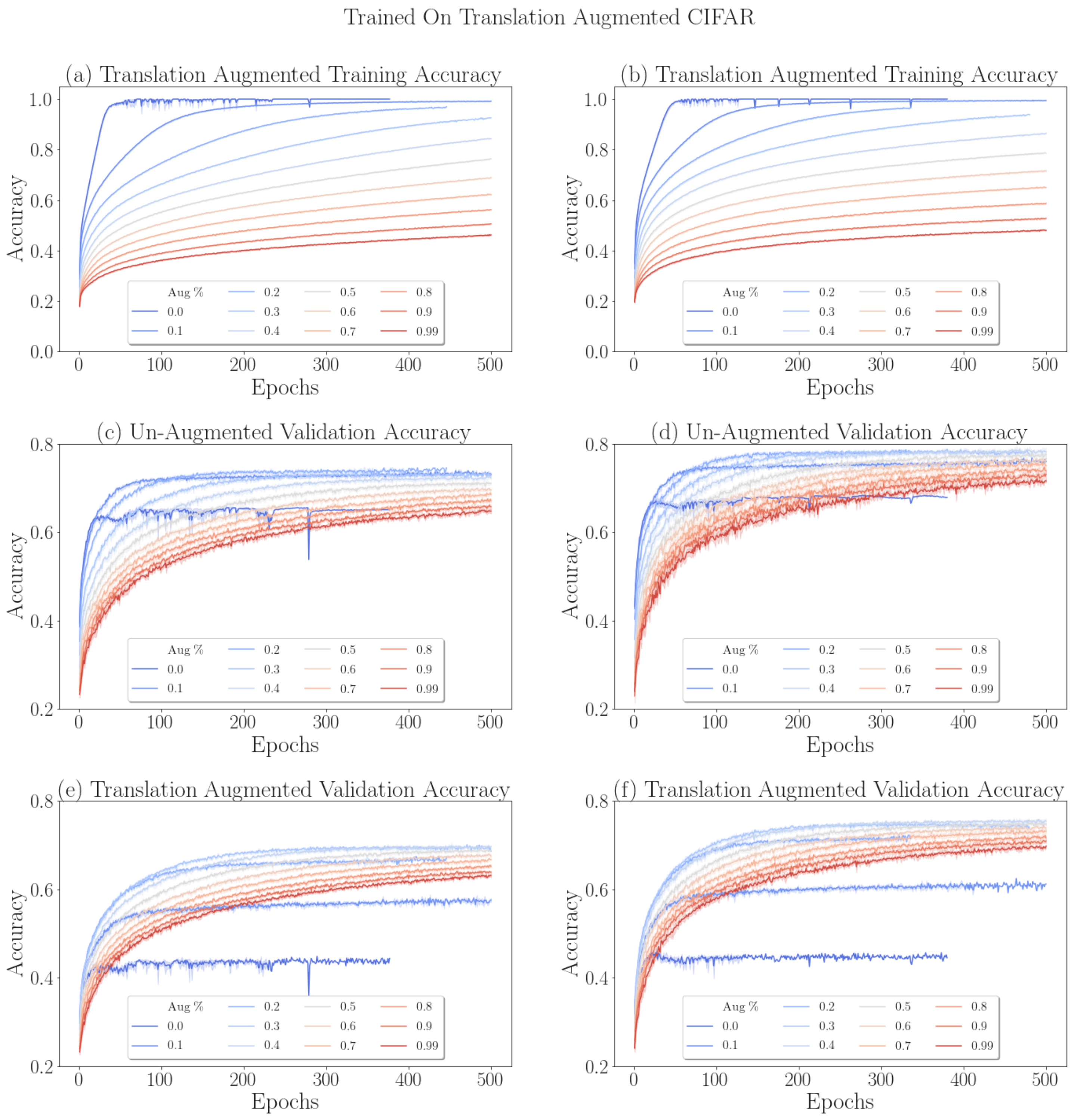


Figure 2.4: CIFAR results. Shown above are FCN (left column) and CNN (right column) results trained with varying degrees of translational augmentation, indicated by the legend. Top row: training accuracy over time. Middle row: validation accuracy over time. Bottom row: accuracy on the translationally augmented validation set.

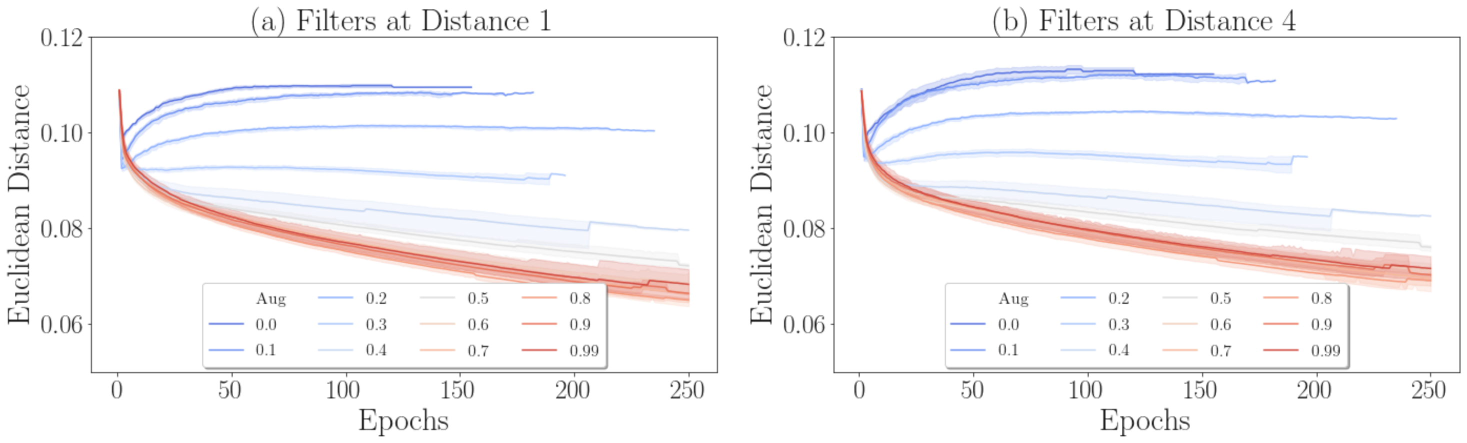


Figure 2.5: Shown above is the average Euclidean distance between weight kernels in FCN layer overtime at the specified radius away. E.g., a radius of four indicates comparing a given weight kernel to all kernels four units away. As translational augmentation is increased weights become more similar to one another. As expected, weights closer in proximity (radius 4; left figure) have a smaller average distance than weights farther away (radius 7; right figure).

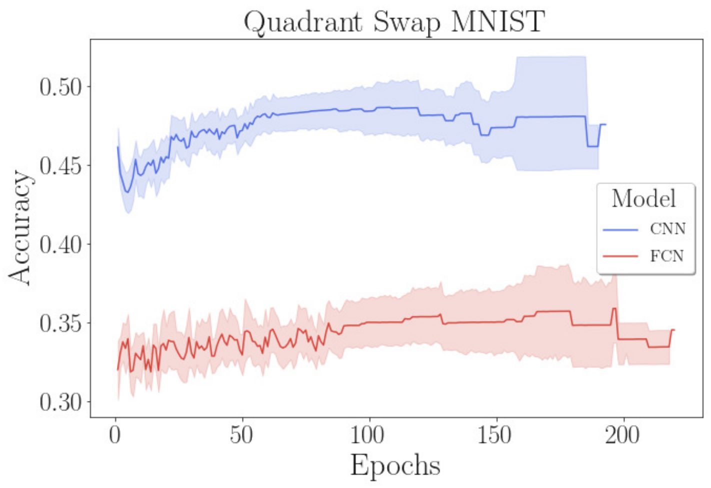


Figure 2.6: Results for CNN and FCN models trained on MNIST and tested on images where quadrant I is replaced with quadrant III and vice versa.

Chapter 3

A Fortran-Keras Deep Learning Bridge for Scientific Computing

3.1 Introduction

The Fortran programming language was originally developed in the 1950s and published in 1957. It was created to help programmers implement solutions for scientific and engineering problems on the IBM 704 computer, which at the time needed to be written in machine or assembly language. Fortran has been regarded as revolutionary and possibly one of the most influential software products in history [1]. Having evolved many times since its creation, with the most recent release in 2018, each version adds new features and capabilities. Fortran initially gained popularity and remains a widely used language due to its fast and efficient computational ability. Additionally, Fortran's strength is its backward compatibility, which allows modern compilers to build code written in the 60s and 70s.

Though not as popular as it once was, Fortran is still used in specialized fields, including oceanography, solid mechanics, computational physics, earthquake simulation, climate modeling, and aerospace. Because of Fortran's continued use, a great deal of legacy code and new code exists. Unfortunately, it is difficult to rewrite all existing code bases in more mainstream languages, due to their size and complexity. Therefore, when algorithms and extensive libraries are created in modern languages, backwards compatible methods must be developed to make them available in older legacy code, like Fortran.

In recent years, the rise of machine learning and deep learning has led to successful applications in various domains. Substantial improvements in the size of the training sets and available computing power have led to a new wave of implementations [100, 170]. In turn, this success has increased the usage and dissemination of deep learning. These methods have been applied to a variety of domains, e.g., ranging from remote sensing [216, 101] to computer vision [138, 188, 139, 140, 192], and to games [5, 176]. Specifically, within scientific computing, many advancements have been achieved through the application of neural networks. Neural networks have been augmented with physically informed capabilities [153, 31], better suiting them for conservation restrictions. Learning partial differential equations [25, 158] has proved valuable in multiple scientific domains.

The success and popularity of deep learning have inspired the creation of powerful software libraries written in several modern programming languages. However, Fortran is not among the modern languages that benefit from these deep learning libraries. This absence leaves Fortran programmers with few options to implement deep neural networks.

The implementation of deep neural networks, in Fortran, may be achieved via two primary pathways. One solution is to rewrite all existing deep learning libraries in Fortran. The second solution is to leverage existing frameworks and bridge available functionalities to Fortran. The former is extremely arduous and time consuming, considering the size and scope of existing deep learning packages and the dizzying pace of their evolution [44, 3, 145]. The latter approach, which this paper describes, is to allow users to leverage the power of existing frameworks while providing a bridge between paradigms where deep learning resources are plentiful and those where they are scarce. In this way, we can leverage aspects of currently available deep learning software libraries, like Keras [44], and bring them to large-scale scientific computing packages written in Fortran. To this end, we propose the Fortran-Keras Bridge (FKB) – A two-way bridge connecting models in Keras with ones available in Fortran. The source code is publicly available and can be found here: `https://github.com/scientific-computing/FKB`. We begin by reviewing existing Fortran projects that would benefit from the integration of FKB.

3.2 Fortran Projects

FKB can be integrated with many existing large-scale and computationally intensive projects written in Fortran. These projects will benefit from the easy integration of neural network models, which FKB makes possible.

For example, Fortran is used to do a great deal of work in climate and ocean modeling. For instance, the US-produced Community Earth System Model [84] is written in object-oriented Fortran-90; this is the most widely used climate model in the world. So are the other climate simulation codes used by the US Department of Energy [69] and the National Oceanographic and Atmospheric Administration's Geophysical Fluid Dynamics Laboratory [75]. Meanwhile, the Nucleus for European Modelling of the Ocean (NEMO) engine is used for studying ocean circulation problems on regional and global scales [186] and making future predictions, is also written in Fortran. The Hybrid Coordinate Ocean Model (HYCOM) [196], also used for ocean modeling, extends traditional ocean models to allow for a smooth transition from the deep ocean to coastal regimes. Researchers have also developed models for the modeling of waves and wind stress [54]. The Weather Research and Forecasting Model (WRF), is arguably the most widely used numerical weather prediction models for regional decision support [149]. Since its release in 2000, the number of WRF registrations has grown to over 36,000. WRF produces atmospheric simulations with support for special applications, including air chemistry, hydrology, wildland fires, hurricanes, and regional climate, and is again a Fortran-based model.

Fortran has found continued use in solid mechanics packages for implementing finite element methods. Popular packages such as ANSYS [123], ABAQUS [34], and LS-DYNA [133] are written in Fortran or accept Fortran subroutines. Similarly, in earthquake modeling, the SPECFEM3D [97] package leverages Fortran for simulations.

The list goes on. Code Saturne [11], developed by Électricité de France, and NEK5000 [146], are Fortran open-source computational fluid dynamics packages. Code_Saturne allows for user customization via Fortran subroutines, which is just one application domain for FKB. NEK5000 is actively used in the Center for Exascale Simulation of Advanced Reactors (CESAR) projects. Fortran has also been continually used for molecular modeling within chemistry and physics. The Chemistry at Harvard Macromolecular Mechanics (CHARMM) Development Project has produced a powerful molecular simulation program in Fortran [39]. This simulation program primarily targets biological systems but can also be used for inorganic materials. A similar tool, NWChem, has been developed by the Molecular Sciences Software Group at the Pacific Northwest National Laboratory [193]. NWChem is a computational chemistry software that includes quantum chemical and molecular dynamics functionalities. Within the molecular physics domain, Fluktuierende Kaskade (FLUKA) is a proprietary tool for calculations of particle transport and interactions with matter [60].

The models mentioned above and projects can leverage the FKB library to leverage neural networks within their codebases. For example, neural networks have proven useful in modeling sea surface temperature cooling for typhoon forecasting [88]. Therefore the integration of FKB with tools like NEMO, HYCOM, or WRF models is a possibility. In a recent study of computational fluid dynamics, Ling et al. solve the Reynolds-averaged Navier-Stokes equations, similar to Code_Saturne and NEK5000. By implementing deep neural networks, the authors report that the architecture improved prediction accuracy [114]. Finally, the Fluka tool contains a wide range of molecular physics applications, including dosimetry calculations. Vega-Carrillo et al. have shown neural networks aided in the calculation of neutron doses [195]. For global climate simulation, there is proof that deep neural networks can offer skillful alternatives to assumption-prone approximations of sub-grid cloud and turbulence physics in the atmosphere [155, 36]. We hope that the FKB library enables Fortran users to expand their research and projects to include neural networks.

Having reviewed several Fortran based projects that can leverage FKB, we now introduce the two sides of this bridge. The following sections will develop the foundations on which to anchor each side of this two-way bridge. We start by introducing the deep learning anchor.

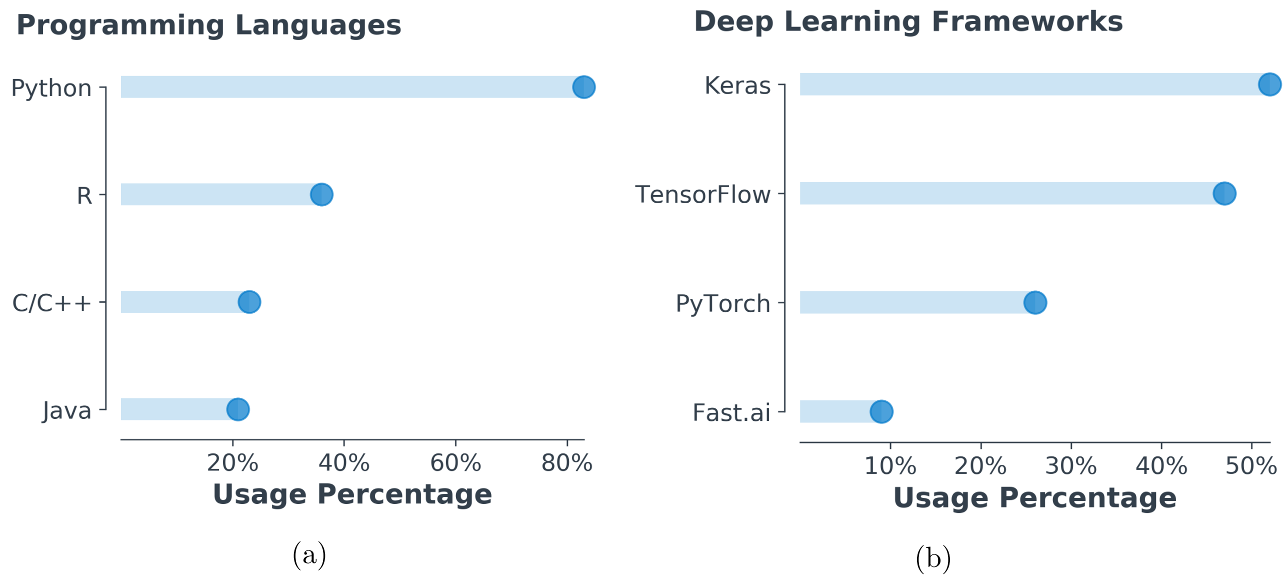


Figure 3.1: (a) Usage of programming languages for machine learning and data science. Statistics are from the 2018 Kaggle ML & DS Survey [91]. (b) Usage metrics of deep learning frameworks. Statistics are from the 2019 Kaggle State of Data Science and Machine Learning report [92].

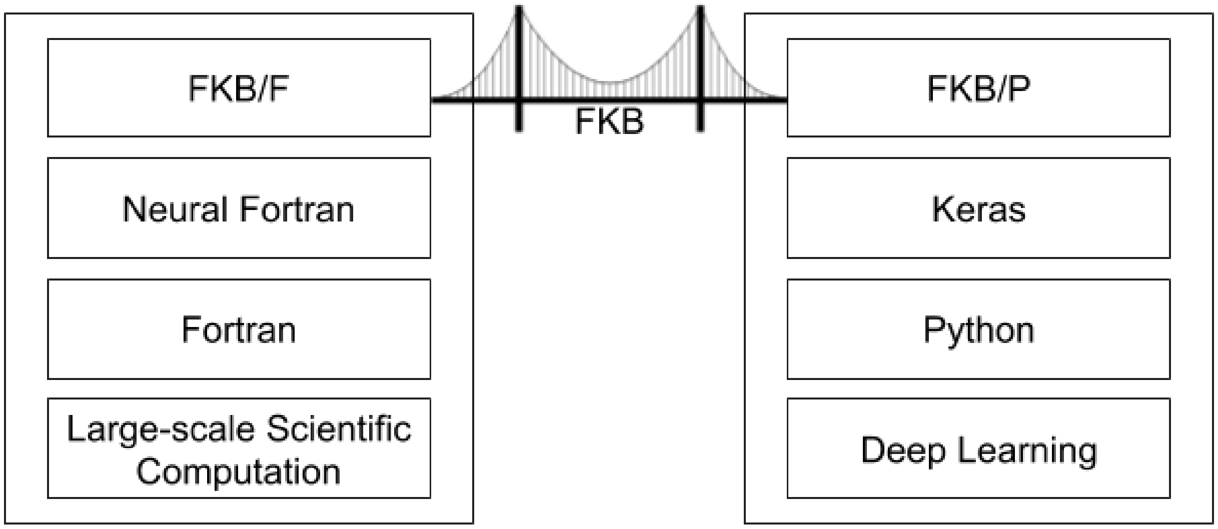


Figure 3.2: Positioning of FKB within Fortran and Python ecosystems.

3.3 The Python Anchor (Deep Learning)

Many programming languages offer tools and libraries for implementing artificial neural networks. However, in recent years, Python has emerged as the clear favorite within this domain. Metrics in Figure 3.1a display Python's dominance. Python is used nearly 50% more than the second most popular language, R. Python's ubiquitous presence in machine learning makes it the obvious choice to leverage existing libraries for Fortran. The question then becomes, which available software library within Python, is best suited to bridge to Fortran?

Of the available deep learning libraries, Keras [44] is the most popular among practitioners (Figure 3.1b). Keras is an Application Programming Interface (API) built on top of Tensorflow [3], that provides users the ability to implement quickly, train, and test networks. This convenience encapsulates much of the low-level complexity one must manage when implementing deep networks from scratch. Keras abstracts many of the complicated aspects of Tensorflow while still providing customizability and ease of use. This combination makes Keras the first choice of many for deep learning applications. As a result of its popularity and ease of use, Keras is the clear choice on which to build one end of the two-way bridge.

Figure 3.2, depicts the positioning of the Python anchor, FKB/P, within the deep learning ecosystem. The Keras API leverages Python to build deep neural networks. FKB/P resides on top of Keras to access models produced from Keras and transmit them to the Fortran anchor, FKB/F. This structure allows for integration with Fortran applications that wish to leverage deep neural network architectures. Having described the deep learning anchor within Python, the next section develops the foundation for anchoring the bridge with Fortran.

3.4 The Fortran Anchor (Scientific Computing)

Several attempts have been made to implement neural networks in Fortran, with some success [50, 28, 29, 37, 135]. However, many implementations resort to hacking a single-use neural network by hand, or binding code from other languages [135]. Along these lines, one may consider accessing Python functionality directly from Fortran, by running a Python instance within Fortran. While providing flexibility and ease of use, this is vulnerable to extreme deficiencies in speed and computational resources. As a result, this solution becomes untenable for large-scale computation projects like the ones described in Section 3.2.

There are a small number of existing neural network libraries in Fortran [135, 102, 50]. The most recent and well developed library is Neural Fortran [50], a lightweight neural network library, written natively in Fortran. The Neural Fortran library provides the ability to implement artificial neural networks of arbitrary size with data-based parallelism. Additionally, in benchmark studies, Neural Fortran was shown to have comparable compute performance with Keras while maintaining a lower memory footprint. This library offers a foundation to anchor the Fortran side of the two-way bridge, FKB/F. By extending - and building on top of - Neural Fortran, we can convert Keras models to ones readily available in Fortran and implement them in existing Fortran projects.

The positioning of FKB within the scientific computing ecosystem is shown in Figure 3.2. The Fortran anchor, FKB/F, can use models originally constructed and trained in Keras, which can then be transferred to Fortran via FKB/P. To use these models, the Fortran side of FKB implements a neural network library. This portion of FKB can be used within large-scale scientific computation software, like the projects identified in Section 3.2.

By leveraging FKB, it becomes seamless to train networks in Python and transfer them to Fortran, to run inside large scale simulations. Similarly, neural network models constructed in Fortran can be transferred to Python for additional analysis, expansion, and optimization - including hyperparameter searches using available tools in Python [78, 178, 27]. As both sides of the bridge have been properly introduced, the following section will describe the specific features and functionalities of FKB.

3.5 Features of FKB

Once a neural network is trained in high-level APIs like Keras, the practitioner has few practical avenues for using this model in Fortran-based projects. One approach may be to hard code network operations inside Fortran while manually moving parameters from the Keras model. Several examples of this can been seen in climate modeling [155, 36, 63, 62].

To provide one specific example, in [155], the authors trained a deep neural network (DNN) to represent sub-grid cloud and convective energy transport processes, in Keras. To assess its credibility, they needed to test the DNN's two-way interactions when thousands of replicates of it were embedded within a coarse-resolution global atmospheric model, written in Fortran – neural network emulated clouds interacting with determinstic physical calculations of planetary geophysical fluid dynamics. As the global atmospheric simulator does not offer native neural network support, the authors hardcoded their DNN model into the global simulation software framework. This approach has obvious disadvantages. Every minor change made to the model in Keras requires rewriting the Fortran code. If one wishes to test a suite of models in Fortran, this approach becomes untenable. As each network may require different hyperparameters and, as a result, necessitates rewriting and compiling the Fortran code for every new model. This process drastically limits the breadth of available models to be tested within the simulator. This bottleneck is currently a significant roadblock to ongoing debates in the climate simulation community, more broadly, about whether or not to use DNN representations of subgrid physics in next-generation climate modeling. Insufficient testing of diverse candidate neural networks (NN) means that little is known about how minor imperfections in the fit of one NN can amplify when the NN is coupled to fluid dynamics, which is just beginning to be explored [35].

These issues demand a solution, in the form of a bridge between Keras and Fortran. The FKB software solves these issues via two key elements. First, it provides a neural network library implemented in Fortran (FKB/F). Second, it offers the ability to parse existing Keras models into formats consistent with the Fortran neural network library (FKB/P). As a result, users can switch, seamlessly, back and forth between Python and Fortran. This context provides a way for iterative neural network tuning (Python) and testing (Fortran), with a simple way to translate between the two software environments. Additionally, FKB offers currently unavailable Fortran specific features for neural networks. It will be useful to highlight those new features while documenting the format to which FKB adheres. The following subsections describe the Python and Fortran anchors' features, FKB/P and FKB/F, respectively.

3.5.1 FKB/P

Keras models - once built, trained, and saved - are stored in Hierarchical Data Format 5 (HDF5) files. These files contain the network architecture, weights, biases, and additional information - optimizers, learning rates, gradients, etc. From the HDF5 file, FKB/P parses the network architecture, extracting the number of layers, activation functions, nodes per layer, and all weights and biases. This information is converted to match the Fortran neural network configuration in FKB/F. This allows users to build an equivalent network in Fortran, which can easily be loaded and used within a Fortran environment. If any modifications to the model are made inside Fortran, FKB/P will parse this back into the equivalent HDF5 file to be used in Keras once again.

On the other hand, networks may be initially constructed in Fortran. After initial training and testing, a user can switch to Keras for further evaluation. From Keras, users can conduct additional testing or hyperparameter tuning where these tools are readily available [78].

The ability to seamlessly pass neural network architectures between Python and Fortran is essential for any practitioner working in this space. This bridge allows users to take advantage of the high-level Keras API - training on computationally efficient GPUs - then to insert their trained model into a Fortran codebase. The functionality provided bridges the chasm between Keras and Fortran.

3.5.2 FKB/F

The Fortran anchor of FKB leverages and extends the original Neural Fortran library. Below we introduce newly implemented features to make Neural Fortran more flexible and able to communicate on the two-way bridge.

Custom Layers

To implement neural networks in Fortran, FKB leverages and extends the Neural Fortran library [50]. The prototype Neural Fortran library format that we build on was only capable of implementing a fully connected layer. Forward and backward operations occurred outside this layer - in the network module. An example of this is shown in Listing 3.1. From the listing, one can observe hard-coded matrix multiplication of layer weights, the addition of biases, and the activation functions inside the network module. This network-level subroutine accesses and modifies individual layer attributes. This rigid format is inconsistent with modern neural network implementation paradigms [44, 3, 145], but it makes it impossible to implement other layers or custom operations. To increase the library's flexibility, operations must be encapsulated inside the layer, consistent with current practice.

```
pure subroutine fwdprop(self, x)
  ! Performs the forward propagation and stores arguments to activation
  ! functions and activations themselves for use in backprop.
  class(network_type), intent(in out) :: self
  real(rk), intent(in) :: x(:)
  integer(ik) :: n
  associate(layers => self % layers)
    layers(1) % a = x
    do n = 2, size(layers)
      layers(n) % z = matmul(transpose(layers(n-1) % w), layers(n-1) % a) +
        layers(n) % b
      layers(n) % a = self % layers(n) % activation(layers(n) % z)
    end do
  end associate
end subroutine fwdprop
```

Listing 3.1: Original code from [50]. Layer operations occur inside the network module, limiting flexibility.

In FKB we introduce an extendable layer type module (Listing 3.2). To implement a layer, one simply extends the layer type and specifies the construction of the forward and backward functions. Adhering to this format offers several advantages. By restructuring the format of the library, we offer the ability to implement arbitrary layers. Additionally, in the network module, all layers are stored in an array of pointers. This leads to the encapsulated version shown in Listing 3.2 wherein a forward pass, in the network module, calls the layer-specific forward function. In this way, all operations are confined to the layer module, and the output from one layer is passed as input to the next.

```fortran
function output(self, input) result(last_layer_output)
    ...
    ! iterate through layers passing activation forward
    do n = 1, size(layers)
        call layers(n) % p % forward(layers(n-1) % p % o)
    end do
    ! get output from last layer
    last_layer_output = layers(size(layers)) % p % o
end function output
```

Listing 3.2: Forward pass in the FKB network module. Each layer simply calls its own forward function. The technical operations occur within each layer.

FKB supports fully connected or dense layers, dropout [179, 20], and batch normalization [85]. Shown in Listing 3.3 is an example of extending the layer_type to implement a Batch Normalization layer. This format translates to increased functionality and customizability to the user. As a result, more standard layers from Keras are available, while giving users the flexibility to implement their own custom operations.

```
! BatchNorm layer - extends from base layer_type
!   Implements batch normalization
type, extends(layer_type) :: BatchNorm
    ! epsilon parameter
    real(rk) :: epsilon
contains
    procedure, public, pass(self) :: forward => batchnorm_forward
    procedure, public, pass(self) :: backward => batchnorm_backward
end type BatchNorm
```

Listing 3.3: Example of extending the layer_type to implement Batch Normalization

Training in Fortran

It is necessary to distinguish between the terms *offline* versus *online* for the following section. These terms serve to distinguish two different settings in which a neural network can be used in a Fortran computing package. Both settings can make use of historical or simulated data to train an artificial network. The distinguishing feature is how the predictions of a model are used. In an online setting, predictions from the model are used to evolve a physical process. The predictions at one time step effect how the system acts at the following time step. As a result, inputs to the model will change based on how the model acted in the past. In offline settings, this is not the case. Predictions made in the past do not affect the input to the model in the future.

In many cases, offline training may be sufficient to learn a model, if enough prior data is available. However, in some cases, online training may be the method of choice. To this end, FKB is equipped to handle backpropagation for gradient descent optimization of a specified cost function.

The layer encapsulation mentioned above of forward and backward operations (Section 3.5.2) becomes extremely valuable in training. Instead of all computations occurring within the network module [50], they are contained in layer-specific functions. Much like the forward pass, backward operations occur in the layer. In this fashion, each layer is responsible for computing its gradients with respect to its parameters and returning the gradient with respect to the layer below it.

Online training can serve a variety of purposes. First, a neural network model may be learned entirely in Fortran, based on the evolving state variables during the integration of a physical dynamical system simulation, and then transferred to Keras after the fact. In this setting, the ground truth, from the simulator, is passed to the network for it to calculate its errors and update its parameters accordingly through backpropagation. Second, online training could serve to provide gentle corrections to an imperfect pretrained model, for instance, to hedge against the amplification of its imperfections that are only revealed once the NN is coupled to other physical calculations. Here a model is trained offline in Keras and transferred to Fortran (Section 3.5.1). In some cases, for a variety of reasons, the offline training data may have a differing distribution than that of the online data. In such a setting, it proves beneficial to offer slight corrections to the network. Finally, a secondary model may be constructed to learn and compensate for the deficiencies in the primary model. In this way, the two networks work together to balance out any instability issues.

The ease of use and proper format directly results from the encapsulation of layer operations. Online training offers a solution to tackle a suite of potential problems. As a result, models may be updated with slight corrections or learned entirely online.

Custom Loss Functions

In many applications, practitioners may wish to optimize a unique quantity - a function other than a mean squared error or cross-entropy. This is common when target variables interact or additional information is known about their relationship in a desired application. For example, in modeling any physical system, predictions from a neural network must not violate physical constraints - energy cannot be created or destroyed in the system. To satisfy this restriction, a loss function can be written to quantify the amount of violation of physical properties. This construction can then be minimized to alleviate constraint infractions [31].

The implementation of custom loss functions is standard for high-level APIs like Keras, Tensorflow, and PyTorch to provide this ability in their codebase [44, 3, 145]. As FKB is designed for those working in the physical sciences where environmental, physical, or application-specific constraints are common, it provides the ability to implement custom loss functions. To take advantage of this functionality, users must implement their desired loss function, just as they would in Keras. As FKB does not provide automatic differentiation, the derivatives with respect to the input are also required for training. Once these functions have been specified they can be dropped into the existing framework and run normally, much like Keras.

```
real(rk) function crossentropy_loss(self, y_true, y_pred)
   ! Given predicted and expected output, returns the scalar loss
   class(network_type), intent(in out) :: self
   real(rk), intent(in) :: y_true(:), y_pred(:)

   loss = - sum(y_true * log(y_pred))
end function loss

function d_crossentropy_loss(self, y_true, y_pred) result(loss)
```

```
    ! Given predicted and expected output
    ! returns the loss with respect to softmax input
    class(network_type), intent(in out) :: self
    real(rk), intent(in) :: y_true(:), y_pred(:)
    real(rk), allocatable :: loss(:)

    loss = y_pred - y_true
end function d_loss
```

Listing 3.4: Implementation of crossentropy loss function and the corresponding derivation with respect to the input logits.

This capability is demonstrated through the implementation of the cross-entropy loss function in Listing 3.4. To implement this previously unavailable loss function, we first declare two functions. First, the cross-entropy scalar loss is. Second, the loss with respect to the input logits is derived. These two functions are then referenced as the loss and d_loss, respectively. By providing this functionality, users may leverage a variety of loss functions that can be used to minimize application-specific quantities. Once described, they may be included with the existing framework and used during online training.

Ensembles

Ensembles consist of different models, each trained on the same, or bootstrapped, data. The output of the ensemble will be an average of all its member's predictions. In machine learning, ensembles of models typically perform better than any one of its members alone. The ensemble strategy exploits the fact that each model will make different errors. Therefore, when averaged together, these predictions become more accurate, as certain errors get smoothed out. A consensus from machine learning practitioners is ensembling gives 1-2%

improvement in performance [43].

As a result of this averaging, ensembles provide a boost in performance as well as additional robustness. In domains where physical constraint violations yield stability issues, ensembles may be applied to dampen these problems. By averaging across many networks, the instability of any one model will be drastically reduced in the presence of more sound predictions.

The functionality provided requires the user to specify a directory that contains the models of interest and a desired amount of noise. The ensemble type will read in each model and construct a network corresponding to each of them. To get a prediction from the ensemble, an input vector is passed to it. For non-zero amounts of noise, Gaussian noise is applied to the input vector each time it is passed to an ensemble member. This allows each member to see a slightly different variant of the input, increasing the robustness of prediction around that point. This operation runs in parallel using OpenMP, where each network can be given its thread to expedite computation; such an approach could easily be adapted via OpenACC for GPU-based threading of large ensemble network calculations. Following the computation, the predictions are averaged together, and the final output is given.

3.6 Case Study

The following section provides a case study demonstrating an application of FKB to experimental next-generation climate modeling. The Superparameterized Community Atmospheric Model version 3.0 (SPCAM3) is used for all simulations in this study. SuperParameterization is an approach that confronts the decades-long problem of representing subgrid cloud physics in climate models by embedding thousands of limited-domain explicit submodels of moist convection within a conventional planetary-scale model of the large scale atmosphere [70, 96, 95, 187]. This approach tends to involve two orders of magnitude more computational intensity per unit area of the simulated earth, but recently Rasp et al. used a deep neural network to emulate all of the expensive subgrid cloud resolving models' (CRM) influence on the planetary host at drastically reduced computational expense [155]. This study, along with others in the emerging climate modeling literature [36] have demonstrated the potential advantages of a data-driven approach for addressing the critical unresolved effects of clouds and convection on planetary climate, as compared to previous, heuristic based, approximations to subgrid physics. However, the idea of emulating turbulence in climate simulation is still an emerging one, with unclear trade-offs, including frequent instabilities when NN emulators are coupled with fluid dynamics, which the community is seeking to learn how to control [36]. It has even been questioned whether the offline skill of such emulators, during their training, is predictive of their online performance [154, 64], an important open question.

These questions are understudied primarily due to the lack of the simple software interface that FKB now enables for climate scientists to test diverse candidate neural networks, and ensembles within planetary climate models.

To illustrate an advance on this front we now apply FKB to shed new light on two related questions currently in debate:

1. Does offline performance translate to online model performance [154, 64]?
2. Which neural network hyperparameters most affect online performance?

Using FKB, the study can be broken into two stages. First, a suite of 108 candidate neural network models of convection are trained, via Keras, on simulated data from the SPCAM3. Second, the models are converted to Fortran and run online (i.e. coupled to planetary fluid dynamics) in the SPCAM3 simulator. The number of steps serves as a preliminary metric of performance until catastrophic failure. It is clear that in the absence of the FKB library, running hundreds of candidate neural network submodels of convection within the Fortran based model of the rest of the planet's atmosphere would be nearly impossible. As each network contains various hyperparameters, each with different weights and biases learned during training, including layer-specific properties such as optional use of dropout or batch-normalization. To leverage the FKB library with SPCAM3, we simply compile the neural network library in advance and link it to the compilation of SPCAM3. Documentation steps for the implementation of this case study are provided here: `https://github.com/scientific-computing/FKB/blob/master/SPCAM_Instructions.md`.

The input to this neural network model is a 94-dimensional vector. Features include vertically resolved vectors representing the large scale (host model) temperature, humidity, meridional wind vertical structure, surface pressure, incoming solar radiation, sensible heat flux, and latent heat flux scalars. The output of the network is a 65-dimensional vector composed of the embedded models' influence on their host – i.e. the sum of the CRM and radiative heating rates, the CRM moistening rate, the net radiative fluxes at the top of the atmosphere and surface of the earth, and the precipitation.

The training data used here are challenging to fit, as they come from an enhanced version of the CRM training data that was originally studied by [155]. In superparameterized simulations, one can control the degrees of freedom of the interior resolved scale through the room available for interesting forms of sub-grid storm organization to form. One can control the physical extent (i.e. number of columns used in) each embedded CRM array [150]. In [155], CRM arrays with only 8 columns (32-km extent, given the 4-km horizontal resolution) were used. Here we quadruple the extent (from 32 km to 128 km, i.e. from 8-columns to 32-columns) to improve its physical realism. Despite several attempts, these data have never been fit successfully. NNs trained from the enriched data tend to produce crashes within just a few simulated weeks after they are embedded in the climate model (see discussion of "NN-unstable" by [35] for details).

Name	Options	Parameter Type
Batch Normalization	[yes, no]	Choice
Dropout	[0, 0.25]	Continuous
Leaky ReLU coefficient	[0 - 0.4]	Continuous
Learning Rate	[0.00001 - 0.01]	Continuous (log)
Nodes per Layer	[128,256,512]	Discrete
Number of layers	[4 - 11]	Discrete
Optimizer	[Adam, RMSProp, SGD]	Choice

Table 3.1: Hyperparameter Space

Our working hypothesis is that historical failures in free-running tests when emulators are trained on higher quality CRM training data reflect a broader issue of insufficient hyperparameter tuning in climate model applications. To address this, we conducted neural network optimization via a random search using SHERPA [78], a Python library for hyperparameter tuning. We detail the hyperparameters of interest in Table 3.1, as well as the range of available options during the search. The hyperparameters of interest consisted of whether or not to use batch normalization, the amount of dropout, the leaky ReLU coefficient, learning rate, nodes per layer, the number of layers, and the optimizer. The random search algorithm has the advantage of making no assumptions about the structure of the hyperparameter search problem and is ideal for exploring a variety of settings.

We attained 108 candidate neural network model configurations, each trained for 25 epochs with early stopping monitoring the validation loss. Following the offline training stage, the neural network models were converted into their Fortran counterparts and ran inside SPCAM3. We underscore that this critical step would have been prohibitive using standard tools that have required manual translation of each candidate model. However, by leveraging the FKB library, each model was loaded independently into Fortran and run as the subgrid physics emulator inside SPCAM3's host planetary model, of the large-scale atmospheric state. Each model was coupled to fluid dynamics, to run a wide ensemble of prognostic tests across an unprecedented diversity of candidate neural network architectures. Each of the one hundred and eight candidate neural network models - with their various numbers of layers, layer-specific settings (batch-normalization, relu magnitude, etc), nodes per layer, weights, and biases - were run online, all without rewriting any Fortran code.

In order to address the first question and evaluate a neural network model's performance, we compare its validation MSE during training with the time-to-failure of the online tests in which 8,192 instances of the NN, spaced at regular intervals around the globe, are coupled interactively to their host global atmospheric model of large scale geophysical fluid dynamics. This yields Figure 3.4a, which sheds new light on the offline vs. online relationship.

The results in this figure demonstrate a relationship between offline validation error and online performance. There is a distinct, negative, relationship between offline MSE and online stability (Spearman correlation of -0.73; $p = 4.961e^{-19}$. Intriguingly, the mean-squared error loss of our multi-layer perceptron is a reasonable predictor of stability once coupled to the climate model, insofar as the time-to-failure is concerned. This finding is interesting in the context of the recent speculation by [154] that such a relationship might not exist using similar NNs in a similar setting, as well as the comments by [64] about similar incongruities even in reduced-order dynamical systems when emulated with GANs.

Of course, stability alone is a necessary but not a sufficient condition of prognostic success, which also requires an in-depth analysis of biases in the simulated climate. Figure 3.3 shows the time-evolution of the tropospheric temperature and humidity biases, colorized by the offline validation error. These metrics reveal that although our search has uncovered many runs that are “stable” - can run without catastrophically crashing for several months - most of these runs would not be very useful in an operational setting. Almost all NNs exhibit major errors in the simulated climate, having drifted to erroneous attractors with root-mean-square errors in temperature frequently above 10 K. However, the NN that produced the best offline validation error stands out as having the combined desired qualities of stability and skill with temperature biases of less than 2 K, competitive with [155]. Interestingly, coupling instead to the ensemble mean of a few of the best-ranked models (magenta dashed lines) does not outperform coupling to the best fit model, the value of having found it using SHERPA (Figure 3.3).

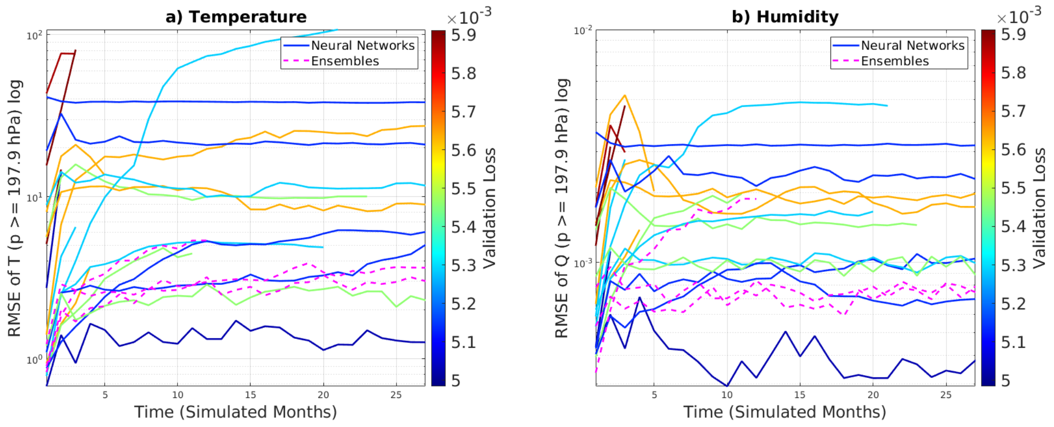
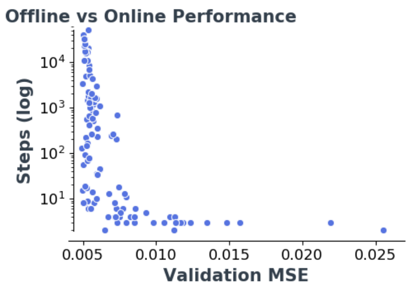


Figure 3.3: The time-evolution of the tropospheric (a) temperature and (b) humidity biases, colorized by the offline validation error

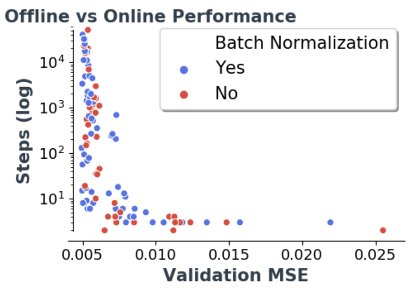
In short, we have produced a successful coupled simulation that was particularly challenging without formal hyper-parameter tuning and FKB. This result suggests that sufficient hyper-parameter tuning may be critical to solving chronic instability in climate model applications of DNNs for subgrid physics.

The second question naturally arises as to which of the hyperparameters are most impactful to the online performance. To assess this, Figure 3.4b-3.4i decomposes the sensitivity of the baseline relationship to individual hyperparameter choices. The choice of optimizer is shown to correlate most strongly with online performance (Figure 3.4i). This finding is confirmed by Spearman values, shown in Table 3.2. The optimizer hyperparameter has the largest absolute correlation value with online performance. No other hyperparameter shows as clear a distinction in correlation that is evident in the choice of optimizer, including the network depth and total number of parameters, which are known to be important to offline fits for this problem [68], but are surprisingly not as predictive of coupled skill as the choice of optimizer, whose impact has not previously been isolated (for this application).

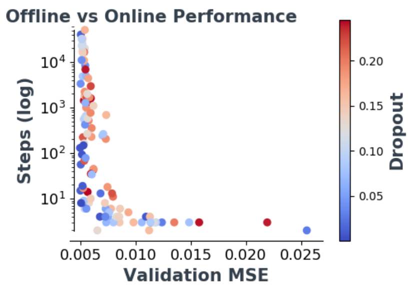
Further investigation into the specific optimizer used, reveals the SGD optimizer to perform poorly; NNs fit with SGD never run longer than 1,000 steps when coupled online (Figure 3.4i). Again the visual intuition from Figure 3.4i is confirmed by Spearman correlation values. SGD, Adam, and RMSProp have Spearman values of -0.6670, 0.5936, 0.0586 respectively. These values demonstrate that the use of SGD is negatively correlated with online performance, whereas Adam positively correlates with online performance. This result leads one to speculate that increased improvements in online skill may be realized from more advanced optimizers with enhanced gradient update schedules.



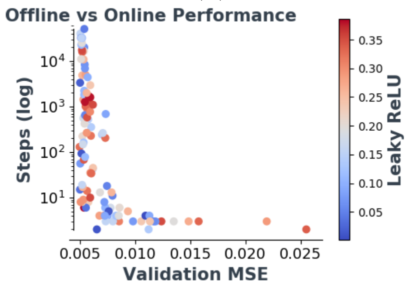
(a)



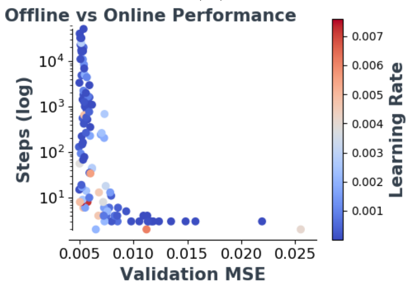
(b)



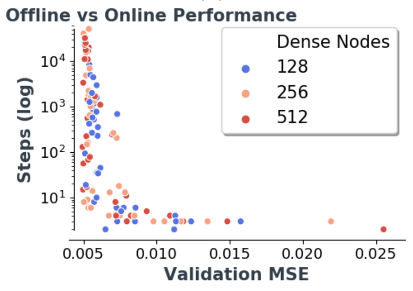
(c)



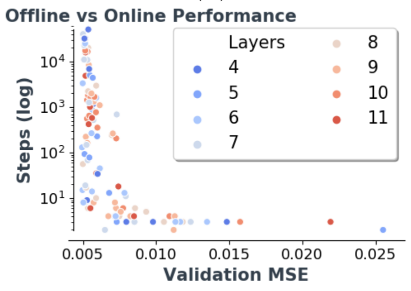
(d)



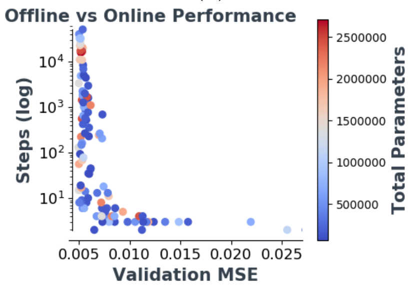
(e)



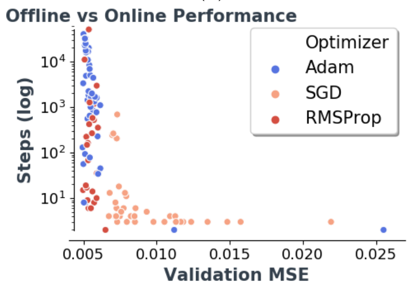
(f)



(g)



(h)



(i)

Figure 3.4: Offline performance - validation mean squared error (MSE) - vs online performance - number of steps until crash.(a) All models. (b) By batch normalization usage. (c) By Dropout amount. (d) By leaky ReLU coefficient. (e) By learning rate. (f) By number of dense nodes per layer. (g) By number of layers. (h) By total number of model parameters. (i) By optimizer type.

	Correlation	P-Value
BatchNorm	0.0859	3.7896e-01
Dropout	0.1919	4.7591e-02
Leaky ReLU	0.0055	9.5465e-01
Learning Rate	-0.2087	3.0923e-02
Dense Nodes	0.1427	1.4249e-01
Layers	0.0410	6.7491e-01
Optimizer	**-0.6998**	5.0177e-17
Parameters	0.1528	1.1609e-01

Table 3.2: Spearman correlation of corresponding hyperparameter with online performance, and associated p-value.

Finally, after answering the two questions motivating this case study, we can compare the results of the best performing model with that of previously published models of [155] when applied to the challenging limit of CRMs with 32-km horizontal extent. The model proposed by Rasp et al. was a single deep neural network. The hyperparameter space of this model was not fully explored online in large part due to the laborious process required to transfer those models into Fortran. The Rasp et al. model (provided by the authors) ran for 128 steps before crashing due to instability issues. The five best models achieved in this study ran to completion of a 5-year simulation, i.e. for 87,840 steps; of these, two of the five models further exhibited root-mean-square errors in simulated tropospheric temperature of less than 2 degrees Celsius. This dramatic improvement in stability is a direct result of the ease with which a wide variety of models (identified by SHERPA) can be transferred between Python and Fortran (thanks to FKB). We also note that this method is preferable to another approach that was recently proposed to begin stabilizing the same model, through small-amplitude Gaussian input perturbation [35] - a strategy that, while promising, adds computational expense and introduces out-of-sample extrapolation issues that can be avoided with the brute-force optimization and wide-ensemble prognostic testing path to stabilization we have outlined here.

This case study has investigated two closely entangled questions: 1) Does offline performance correspond to online model performance? 2) What neural network hyperparameters most effect online performance? Both of these questions have been answered by leveraging the FKB library. The library offers the ability to expeditiously transfer models trained in Keras to Fortran, where they may be run online in existing simulators. In the absence of FKB, neither one of these questions could be approached without unreasonable human intervention, as the operational target is a climate model with over a hundred thousand lines of code written in Fortran.

3.7 Conclusion

The ubiquitousness of deep learning has resulted from extensive free and open source libraries [44, 3, 145]. Deep learning's success and popularity merit its integration in large-scale computing packages, like those written in Fortran. Instead of rewriting all existing libraries in Fortran, we introduced a two-way bridge between low-level, Fortran, and Python through the FKB Library. The library provides researchers the ability to implement neural networks into Fortran code bases while being able to transfer them back and forth with Keras.

Fortran, which has been a staple within computationally intensive fields for decades, will undoubtedly see continued use due to its fast computational ability and vast amounts of legacy code. The FKB library enables users to access many features of the Keras API directly in Fortran, including the ability to create custom layers and loss functions to suit their needs. We demonstrate the integrability of FKB through our case study involving the SPCAM3 simulator. An advantage of FKB is its ease of use, demonstrated by its ability to be compiled in advance and once linked can be easily leveraged in existing large scale simulators, as we have illustrated for the application of multi-scale physical simulations of the global atmosphere.

Chapter 4

An end-to-end CNN with attentional mechanism applied to raw EEG in a BCI classification task

4.1 Introduction

Advances in brain science and computer technology in the past decade have led to exciting developments in Brain-Computer Interfaces (BCI), thereby making BCI a key research area in applied neuroscience and neuro-engineering [4]. Non-invasive BCI facilitates new methods of neurorehabilitation for physically disabled people (e.g., paralyzed patients and amputees) and patients with brain injuries (e.g., stroke patients) [4]. BCI systems utilize recorded brain activity to directly communicate between the brain and computers to control the environment in a manner compatible with the individual's intentions [120].

However, the ability to decode intentions is also an important tool for basic neuroscientific research. In particular, it strongly enhances the scientific armamentarium used to investigate volition [167, 171]. And, more specifically, decoding intention in real time would open the door to interesting experimental possibilities, such as interventions to facilitate or frustrate intentions [172, 104, 106] and intention-contingent stimulation [167]. Technological advances of recent decades—such as untethered, wireless recording, machine-learning-based analysis, and real-time analysis of raw EEG signal have increased the interest in electroencephalography (EEG) based BCI approaches [126].

EEG has proved to be the most popular brain-imaging method for BCI because it is inexpensive, noninvasive, directly measures neural activity (as opposed to fMRI for example), and can facilitate portability to clinical use [120]. EEG signals thus serve as pathways from the brain to various external devices, resulting in brain-controlled assistive devices for disabled people and brain-controlled rehabilitation devices for patients with strokes and other neurological deficits [4, 52, 122]. One of the most challenging topics in BCI is finding and analyzing the relations between recorded brain activity and underlying models of the human body, of biomechanics, and of cognitive processing. The investigation of relations between EEG signals and—real and imagined—upper limb movement has gained more attention in recent years [132, 47].

To implement an EEG-based BCI system for a particular application, a specific experimental protocol and paradigm must be chosen for all phases of the experiment. Typically, the participant first performs a particular task (e.g., a motor-imagery task, a visual task) to learn how to modulate their brain activity, while EEG signals are simultaneously recorded from their scalp. Using the recorded EEG as training data, a machine-learning-based neural decoder for the paradigm is then constructed [4]. Finally, the participant performs the task again, and the neural decoder is used for BCI control.

The process for BCI systems based on motor imagery (MI) is similar. Though, in this case, the participant imagines the movement rather than actually executing it [132]. Previous studies have confirmed that imagination activates areas of the brain that are responsible for generating actual movement [4, 148]. The most common MI paradigms reported in literature are based on sensorimotor rhythms (SMR) and imagined body kinematics. In the SMR paradigm (e.g., [73, 206] participants imagined kinesthetic movements of some body part—such as hands, feet, or tongue—which result in modulations of brain activity that are trackable using EEG [129]. Imagined movement in such SMR paradigms often causes event-related desynchronization (ERD) in mu (typically 8-12 Hz) and beta rhythms (roughly 12-30 Hz). In contrast, relaxing after MI results in event-related synchronization (ERS) [147]. The ERD and ERS modulations are most prominent in EEG signals acquired from electrode locations C3 and C4 (in the 10/20 international system); these electrodes are approximately above the motor cortices of both brain hemispheres.

MI classification is one of the most popular EEG-based BCI paradigms. EEG MI classification generally consists of four parts: signal acquisition, feature extraction, classification, and control. Most existing feature-extraction methods depend on manually designed features, based on human knowledge. Feature extraction and classification of EEG signals for MI tasks have been attempted in the time, frequency, and space (electrodes) domains—not necessarily mutually exclusively. Time-frequency feature extraction in EEG has focused mostly on short-time Fourier transform [157, 207] or wavelets [7, 55]. In the space domain, filter-bank common spatial-patterns (FBCSP) has achieved notable performance [8, 89]. However, FBCSP uses a fixed temporal duration, ignoring difference between participants. As such, it does not make full use of time-domain information. Moreover, these methods generally use handcrafted features and require heuristic parameter setting—e.g., predefined frequency bands—which often do not generalize well across tasks and participants [51]. As such, they often result in limited classification accuracy [7, 65, 205].

4.2 Related work

Recently, researchers have successfully used deep learning (DL) to perform automatic feature extraction [14] and classification [51, 107, 118, 169]. DL has achieved breakthrough accuracies and discovered intricate structures in various complex and high-dimensional data [189, 210]. In particular, it has provided promising results in the analysis and decoding of EEG signals [103]. Thus, NN architectures, their training procedures, regularization, optimization, and hyper-parameter settings are all active area of research in DL-based analysis of EEG, with advances often resulting in dramatic increases in decoding accuracy [103].

Recently, Zhang et al., proposed a hybrid DL architecture, which combined convolutional neural networks (CNNs) and long short-term memory (LSTM) models to handle sequential time domain data [160]. Even more recently, Dai et al., proposed an architecture composed of a CNN with a hybrid convolution scale (HS-CNN), which separates a signal into three frequency bands using bandpass filters at 4$\sim$7 Hz, 8$\sim$13 Hz, and 13$\sim$32 Hz. The three frequency bands are then fed into the convolutional layers with different filter sizes [51]. The features, including different semantic information, were concatenated and then MI classification was carried out. In another study, Zhang et al., applied an attention module to LSTM to utilize long-range information for EEG-based hand-movement classification [209].

Despite their promise, these deep NN architectures are not easy to train from scratch, because they require large amounts of training data to achieve high classification accuracy. However, it is particularly challenging to obtain a large amount of training samples for MI classification. This is because gathering high-quality data requires training and experience as well as a state-of-the-art EEG machine and a noise-free environment. MI tasks are also time consuming and fatigue-inducing for the participants. For example, during the task, participants must minimize, if not altogether avoid, eye movements and other muscle contractions, especially around the head. At the same time, they typically need to employ a great deal of concentration and attention during MI tasks. Thus, participants can only produce a limited amount of data at each session and must come in for multiple sessions to construct a large dataset of EEG MI. This often results in attrition over the course of multiple sessions.

Data augmentation (DA) can lead to considerable performance gains for DL, reducing overfitting and increasing overall accuracy and stability. DA generates new samples to augment an existing dataset by transforming existing samples in some systematic manner. Exposing the classifiers to various transformations of the training samples, as DA does, makes the models more robust and invariant to these and potentially other transformations when attempting to generalize beyond the training set [200, 208, 214].

DA is an especially important technique for EEG-based BCI because of its specific combination of two factors: the dimensionality of EEG signals tends to be high, while the number of available training samples tends to be low. In a recent systematic review on DA in EEG, Lashgari et al. collected all the papers that used DA for NN-based analysis of EEG up to and including 2019 [103]. They showed that convolutional neural networks (CNN) were the most popular NN architectures for EEG MI classification and typically resulted in accurate decoding. This is likely because CNNs are well suited to end-to-end learning, scale well to large datasets, and can exploit hierarchical structure in natural signals. The review also found that the most common input formulation for motor tasks and MI was raw EEG signals [103].

With these elements in mind, here we investigated the efficacy and generalizability of deep learning on EEG-based decoding of MI. We designed an end-to-end CNN with an attentional mechanism [194]. This is because a CNN with an attention-mechanism architecture can improve classification performance using EEG signals by focusing on essential, task-relevant features on different time-steps.

We begin by testing this architecture on 2 benchmark datasets (BCI Competition IV 2a and 2b) as well as on the dataset that we collected, which we share with the community. Then, we compare MI to ME on the dataset that we collected. Next, we tackled a common question when collecting EEG data: how many channels to record for optimal decoding accuracy? We thus compared the decoding accuracy for different numbers of channels. It has also been demonstrated that DA techniques hold promise for EEG decoding. So, we also tested how much DA can boost the accuracy of our method across the datasets. How much EEG data is needed to train deep NN is also not well understood, especially in relation to DA techniques. We therefore next investigate how the accuracy of our model depends on the amount of data on which we train and the type and amount of DA we use. Of course, structure and anatomical features vary across brains. So, we further investigated what happens to the decoding accuracy when we train and test it on EEG from single participants, on pair of participants, triplets, and so on. In the interest of understanding how well models of EEG decoding generalize to previously unseen participants, we also investigated what happens when we train the model on all but one participant and then test on that remaining participant, with and without transfer learning.

4.3 Methods

4.3.1 Proposed CNN-based neural-network architecture

Convolutional models have been successful in many signal processing applications, as they allow temporally related inputs to be processed together via a sliding-window approach (Figure 4.1). This produces shared weights, where the same weight kernel is applied across the temporal domain (for a 1D convolutional model over time). In our architecture (Figure 4.1), this reduces the number of parameters needed in such a model and enables the signal to maintain its spatial relations—across time within each electrode and across electrodes over the head. The signal from each electrode channel is fed through the same convolutional base to produce an output matrix of dimension $C \times E$, where C is the number of electrodes (or channels) and E is the size of the embedding dimension (Figure 4.1). Hence, the convolutional layers in effect reduce the dimension of the input to the embedding dimension, E.

Now, in the self-attention part of the network [194, 175], we first initialize the weights for the Query (Q), Key (K), and Value (V). The magnitudes of Q, K, and V are derived by the product of the input (I) and the weights. The second step is to calculate the attentional score (S), $S = QK^T$. The shape of S will be ($C \times C$). The Softmax (W) of S is calculated to return a vector of $C \times 1$. The third step is to find the weighted values (M), $M = WV^T$. Each input's value for M is concatenated to return a shape of $C \times C$, which will be the value for the final Attention. $Tanh$ was used to produce the alignment score. In the following, the equations show more details:

I	Input for self-attention, shape (number of channels (C) x the size of the embedding dimension (E))
Key, Value, Query	Initialize weights for key, value and query with shape of input size (C x E)
$K = I \times Key^T$ $V = I \times Value^T$ $Q = I \times Query^T$	Derive key, value, and query Shape (CxC)
$S = Q \cdot K^T$	Calculate attention score by dot product (C x 1)
$W = Softmax(S)$	Calculate Softmax (C x 1)
$M = W \times V$	Multiply scores with value
$O = tanh(M \times W^T)$	Linear transformation of M

The attention layer discussed above is added after the convolutional base (Figure 4.1), so that each electrode channel is computed with every other channel to produce a matrix of scalar values. Summing across rows and normalizing these scalars produces a vector of attention scores. These scores are used to create a linear combination of all the electrode channel vectors, which is passed to the fully connected layers of the network for classification. A valuable part of this model is therefore its interpretability [172, 46, 130]. The attention scores for each electrode channel can be examined to determine the importance of each electrode in the model's prediction. However, in this study we were not interested in the added interpretability that the attentional mechanism affords us. Instead, we relied on the attentional mechanism to improve the prediction accuracy of our architecture. This is because a CNN with attention-mechanism architecture can improve classification performance using EEG signals by focusing on essential, task-relevant features on different time-steps, via the sliding windows. Table 4.1 shows the summary of the proposed NN parameter.

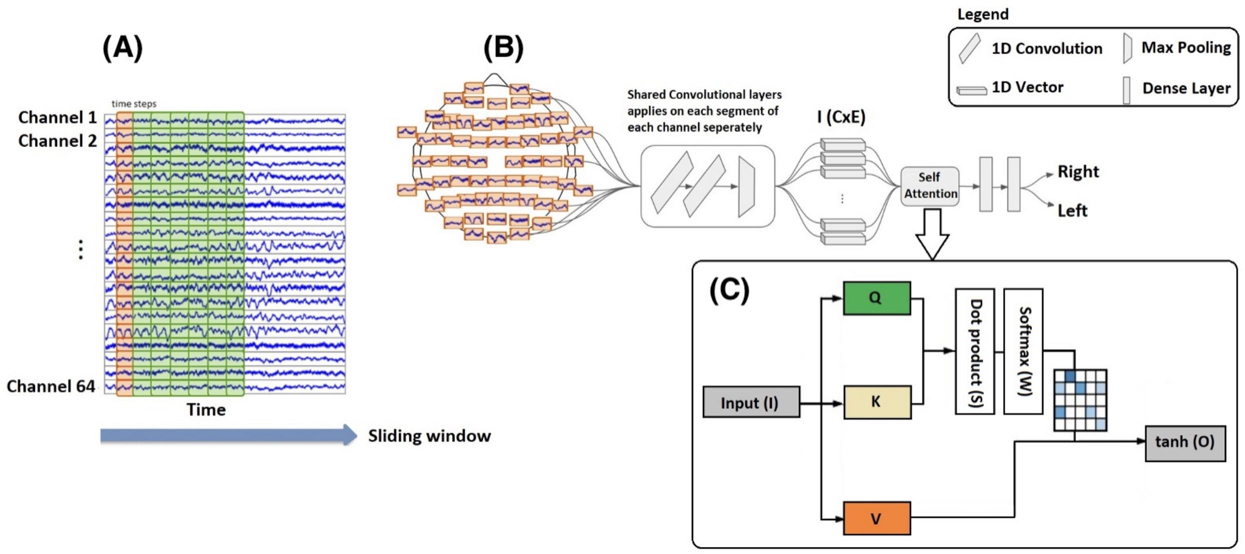


Figure 4.1: Our proposed CNN with attentional mechanism. (A) The sliding window (length is 1000ms and step-size is 100ms) applied to 64 EEG channels. (B) The 64 segments of raw EEG signal, depicted in orange in (A). Each time window and channel are separately sent through shared convolution layers. The embedded features I (CxE) applied to self-attention. The output of self-attention passes through 2 dense layers. (C) An expansion of the self-attention block.

	Layer (Type)	Param #	Shared convolutional layer
Convolution 1D	[-1 ,16, 64]	816	x64
Convolution 1D	[-1 ,16, 6]	12,816	x64
Max Pooling 1D	[-1 ,16, 3]	0	x64
1D Vector	[-1, 48]	2,304	0
Attention	[[-1, 64, 48], [-1, 64, 64]]	4,608	0
Dense	[-1,32]	98,336	0
Dense	[-1,2]	66	0
Total Parameters	977,762		

Table 4.1: Summary of the proposed CNN with attentional mechanism parameters ("-1" represents a flexible shape, essentially the batch size.

4.3.2 Hyperparameter Optimization and Training

When implementing NN there are several choices (or hyperparameters) that must be set prior to training—those range from the type of architecture to the depth and width of the layers, through to the neuronal activation-function in the different layers, and so on. Choosing hyperparameters arbitrarily is likely to lead to suboptimal results. To address this, we first created a 3-way split of our data into a training, validation, and test sets to identify reasonable architectures and parameter ranges. Then, guided by those preestablished ranges, we conducted NN optimization via a Bayesian hyperparameter search using SHERPA [79], a Python library for hyperparameter tuning. The Bayesian search has the advantage of learning a distribution over the hyperparameters of the network architecture, in relation to the task to be optimized. By employing this procedure, we were able to evaluate a large space of possible models and test many configurations.

We detail the hyperparameters of interest in Table 4.2, as well as the range of available options during the search. The hyperparameters of interest consisted of the activation function, dropout percentage, learning rate, learning rate decay, nodes per layer, and the optimizer. Additional hyperparameters for convolutional models included the number of filters and the kernel size. We tried 250 different hyperparameter settings for each network architecture

(Dense NN, Conv Net-Dense NN, Conv Net-Attention-Dense NN), for a total of 750 models over 3 different NN (Dense NN, Conv Net-Dense NN, Conv Net-Attention-Dense NN). Table 4.3 present the result of best hyperparameters tuning by SHERPA for the 3 datasets: BCI competition IV 2a (BCI 2a), BCI competition IV 2b (BCI 2b), and our dataset and for 3 different models (Dense, CNN-Dense, and CNN-Attention-Dense).

For the 3 datasets examined in this study, we adhered to the following procedure. For each set of hyperparameters sampled in the search, we partitioned each subject's data into a training and validation set. The proposed architecture was thus trained on each subject separately. Then, to evaluate the architecture, we averaged the validation accuracy scores across subjects. We then selected the network architecture with the highest average accuracy score across all subjects. Critically, this process ensures that we find architectures that perform well across subjects, but which are not tailored to specific subjects or tasks.

All networks were trained for 250 epochs using an early stopping condition—i.e., when the accuracy on the validation set did not improve for 25 epochs, training stopped. All models were trained using 10-fold cross-validation. The partitioning was stratified to ensure a constant ratio of representation amongst right and left examples—roughly 50/50—in keeping with the ratio in the data overall. This cross-validation procedure requires a given model to be trained 10 distinct times (re-initializing the network parameters each time) and ensures that, on the one hand, different subsets of the data are used for training and testing, while on the other hand, each datapoint serves as part of the training set (9 times) and in the test set (once). To be clear, when we performed cross validation, we used data partitions that were not used during the hyperparameter search. The accuracies reported below are therefore always the average accuracies across the 10 validation sets described above.

To double check our results, we carried one additional train/validation/test split of 75/15/10%, respectively. After this train/validation/test procedure, we ended up with neural architectures that were the same as those selected by the cross-validation procedure above—both

Name	Range	Type
Activation	(ReLU, ELU)	Choice
Dropout	(0, 0.9)	Continuous
Kernel Size	(25, 50, 75)	Choice
Learning Rate	(0.0001, 0.1)	Continuous
Learning Rate Decay	(0.5, 1.0)	Continuous
Number of Dense Nodes	(8, 512)	Discrete
Number of Filters	(16, 32, 64)	Choice
Optimizer	Adam, SGD, RMSProp	Choice

Table 4.2: The hyperparameter space.

in terms of the number of layers and the kernel size. This gave us confidence that our results are not due to some leakage between the training and test sets. Our cross-validation procedure allowed us to report confidence scores, in the form of average accuracies and standard deviations. It also demonstrated that we did not cherry pick a data partition in which the proposed architectures happened to perform well; rather, our models were robust across partitions.

Training took place on NVIDA Titan V GPUs with 12GB of memory. Each epoch took less than a minute to complete. Training for a single fold typically completed within 30 minutes.

4.3.3 Data augmentation

Generally, in machine learning, but especially for NN, the classification accuracy tends to critically depend on the amount of training data; limited training data typically leads to low accuracy. DA comprises the systematic generation of new samples to augment an existing dataset by transforming existing samples in a manner that increases the accuracy and stability of classification [103]. Exposing the classifiers to varied representations of its training samples typically makes the model more invariant and robust to such transformations when attempting to generalize the model to new datasets. DA for the MI task fell into 5 cate-

Dataset	Model	Kernel Size	Activation	Dropout	Learning Rate	Learning Rate Decay	Number of filters	Dense Nodes	Optimizer
BCI 2a	Dense NN	NAN	ReLU	0.171	0.017	1	NAN	27	Adam
	Conv Net-Dense NN	25	ELU	0.092	0.052	1	64	303	SGD
	Conv Net-Attention-Dense NN	25	ELU	0.9	0.1	1	32	91	SGD
BCI 2b	Dense NN	NAN	ReLU	0.845	0.001	0.864	NAN	289	Adam
	Conv Net-Dense NN	50	ReLU	0	0.1	1	16	15	SGD
	Conv Net-Attention-Dense NN	25	ELU	0	0.1	1	64	263	SGD
Our dataset	Dense NN	NAN	ReLU	0.687	0.037	1	NAN	369	SGD
	Conv Net-Dense NN	25	ReLU	0.68	0.034	0.989	32	196	SGD
	Conv Net-Attention-Dense NN	50	ELU	0.807	0.1	0.978	32	183	SGD

Table 4.3: Hyperparameters for each model and dataset.

gories in our analysis: noise addition [112, 144], GAN [72, 204, 212, 213], sliding window [169, 124, 185], Fourier transform [214], and recombination of segmentation [51]. Table 4.4 shows more details about each of these methods. We evaluate all DA techniques with a magnification $m = (2, 5, 10, 15, 20, 30, 50)$ factor for our proposed CNN.

4.3.4 Dataset and experimental protocol

We used three datasets in this study: (1) A dataset that we collected ourselves, (2) the BCI 2a dataset [40], and (3) the BCI 2b dataset [111] (Figure 4.2).

Our dataset: Seven healthy volunteers (3 male and 4 female) participated in the study, all were right-handed and between the ages of 23 to 30 (mean age 28). All participants gave written, informed consent to participate in the study. Participants were seated in a chair at a distance of 80 cm from an LCD screen with both hands resting on a table. They held a

DA methods	Details of the method
Sliding window [169, 124, 185]	Sliding window over the input of each trial, which leads to many more training examples for the network compared to using than the entire. More formally, given an original trial $X^j \in R^{E \times T}$, with E electrodes and T timesteps, we create a set of crops with crop size T' as time slices of the trial: $C^j = (X^j_{1,...E;t,...t+T'} \vert t \in 1, ...T - T')$. All of these $T - T'$ crops then become training examples for our CNN and will get the same label, y_j, as the original trial. The best results in the BCI dataset are for 1s window length. In this study, we tried to evaluate this technique with different m and 100 ms step-size.
Noise Addition [112, 144]	We found two main categories for adding noise to the EEG signals in purpose of DA: (1) Add various types of noise such as Gaussian, Poisson, Salt and pepper noise, etc. with different parameters (for instance: mean (μ) and standard deviation (σ) to the raw signal (2) Convert EEG signals to sequences of images and add noise to the images [103]. Our proposed end-to-end CNN is for raw EEG. Therefore, we add noise just on the raw EEG signal. We add Gaussian noise with different parameters (mean = 0, standard deviation $\sigma = 0.01, 0.1, 0.2, 0.5$) to all channels of raw EEG signal.
GANs [72, 204, 212, 213]	The GAN framework consists of two opposing networks trying to outplay each other [127]. The discriminator (D) is trained to distinguish between real and fake input data. The generator (G) takes a latent noise variable z as input and tries to generate fake samples that would not be recognized as fake by the discriminator. To learn a generator distribution p_g over data x, the generator builds a mapping function from a prior noise distribution $p_z(z)$ to data space as $G(z; \theta_g)$. And the discriminator, $D(x; \theta_d)$, outputs a single scalar representing the probability that x came from training data rather than pg. G and D are both trained simultaneously: we adjust parameters for G to minimize $log(1 - D(G(z)))$ and adjust parameters for D to minimize $logD(x)$ [127]. This results in a minimax game in which the generator is forced by the discriminator to produce ever better samples with value function $V(G, D)$: $$min_G max_D V(D, G) = E_{x \sim p_{data}(x)}[logD(x)] + E_{z \sim p_z(z)}[log(1 - D(G(z)))].$$ GAN can be extended to a conditional model if both generator and discriminator are conditioned on some extra information such as y. In conditional generative adversarial nets (cGANs) y could be any kind of auxiliary information, such as class labels or data from other modalities. We can perform the conditioning by feeding y into the both the discriminator and generator as additional input layer. In the generator the prior input noise $p_z(z)$, and y are combined in joint hidden representation, and the adversarial training framework allows for considerable flexibly in how this hidden representation is composed. In the discriminator x and y are presented as inputs and to a discriminative function. The objective function of a two-player minmax game would be as: $$min_G max_D V(D, G) = E_{x \sim p_{data}(x)}[logD(x \vert y)] + E_{z \sim p_z(z)}[log(1 - D(G(z \vert y)))].$$
Recombination of segmentation [51]	Perform segmentation on the input trials (i.e., left-/right-hand MI) with the same label. Each trial is segmented into three crops. The crops with the same labels are then recombined to generate new trials. For the same person and the same class, the crops at the same position from multiple trials are randomly swapped and recombined in the time/frequency domain to generate recombined trials [51].
Fourier Transform/Wavelet [214]	Apply the empirical mode-decomposition algorithm on the EEG frames and mixed their intrinsic mode functions to create new, artificial EEG frames [214]. The algorithm decomposes the original EEG signals into a finite number of functions called "intrinsic mode functions" (IMFs). Once the signal has been decomposed, we can recover it by adding all the IMFs and the residue without loss. To generate the new samples, we swapped the IMFs of the decompositions. Moreover, the intrinsic characteristics of each class (left/right) will be preserved because we mixed the IMFs of the same class. We randomly select the trials that contribute with their IMFs to generate samples for specific class.

Table 4.4: DA techniques that are used on the MI task

tennis ball in each hand and were told to remain relaxed and strive to minimize movement and eye blinks. When required to respond, they were to squeeze the tennis ball in their hand but try to avoid tensing their arms or shoulders. Each session (ME and MI—Figure 4.2) was repeated twice. The whole experiment thus consisted of four sessions. Every session lasted 30–40 minutes with 10 to 15 minutes breaks between sessions. The duration of the whole experiment, including setup, was kept below 3 hours to minimize fatigue. EEG data was recorded and sampled at 250 Hz using 64 active electrodes (BrainVision actiCHamp) placed according to the 10/20 montage. Bipolar electromyography (EMG) electrodes were placed on the Brachioradialis for both hands as a sanity check for any movement in MI session.

Sessions 1 and 3 were designed to identify EEG signals related to ME. Participants were instructed to squeeze the tennis ball with their right or left hand while fixating on the cross displayed on the screen. They were encouraged to minimize all other movement and to only use the designated hand. One hundred trials were collected for each hand.

Session 2 and 4 aimed to show that a decoding model based on actual ME, derived from the first session, could be used to decode EEG activity in the absence of execution. Participants were instructed to carry out MI of the repetitive hand movement instructed in session 1 while fixating on the cross displayed on the screen. One hundred trials were collected for both left and right imagination per each session. All other aspects of the task were identical to session 1. This session also allowed us to screen participants for the presence of motor-related EEG oscillations, and at least minimal voluntary control over these oscillations. Hence, overall, we collected 200 trials of ME and 200 trials of MI for each subject. The data underlying this study have been uploaded to figshare.

Data are available from the following link: `https://doi.org/10.6084/m9.figshare.14721297.v1`

BCI 2a: BCI 2a contains EEG data from 9 healthy participants [40], 2 sessions per participant. Each session is made up of 288 trials, resulting in 5184 trials overall. No feedback was provided. Twenty-two Ag/AGCL channels were used to record EEG. The signals were sampled with 250 Hz and bandpass filtered between 0.5-100 Hz. To compare our results with previous studies ([51, 66, 121] etc.) we focused on the C3, CZ, and C4 electrodes.

BCI 2b: BCI 2b contains EEG data from another 9 healthy participants [111]. For each participant, 5 sessions of data are collected. Each of the first 2 sessions has 120 trials and each of the last 3 sessions has 160 trials. The total number of trials is thus 6480. Two types of trials are included in these datasets: left- and right-hand MI. The first 2 sessions contain training data without feedback, while the last three sessions gave a smiley face as feedback. The EEG data is again collected over the C3, CZ, and C4 electrodes, which were placed following the international 10–20 system. The sampling frequency was 250 Hz. Table 4.5 presents the summary of three datasets.

4.4 Channel selection

Analyzing dense-array EEG is computationally expensive and complex; it also typically requires more expensive EEG systems than those with sparser electrodes. We therefore tested 4 different electrode configurations on our participants—which included 3, 7, 18, or all 64 electrodes (see Section 4.3)—to further test the effect of channel selection on classification accuracy for MI in our own dataset.

Configuration (1) C3, CZ, and C4 electrodes were chosen in accordance with the 10-20 framework [90] since these electrodes have been shown to be especially discriminatory in hand and foot movements data [94]. It should be noted that right (left) hand's MI operation is usually detected above the left (right) motor cortex underneath the C3 (C4) electrode,

	Experimental dataset	BCI 2a	BCI 2b
The dataset provided by	The Institute for Interdisciplinary Brain and Behavioral Sciences, Chapman University	The Institute for Knowledge Discovery (Laboratory of Brain-Computer Interfaces), Graz University of Technology	The Institute for Knowledge Discovery (Laboratory of Brain-Computer Interfaces), Graz University of Technology
Open-source dataset	Yes	Yes	Yes
Description of dataset	2-class MI and ME (left hand and right hand). Session 1 and 3 are ME and 2and 4 MI, No feedback	4-class MI (left hand, right hand, both feet, and tongue. No feedback	2-class MI (right hand, left hand). The first two sessions contain training data without feedback, and the last three sessions with smiley feedback.
# Channels	64 EEG channels (0.5-100Hz -BrainVision actiCHamp)	22 bipolar EEG channels (0.5-100Hz; notch filtered)	3 bipolar EEG channels (0.5-100Hz; notch filtered)
Sampling frequency	250 Hz	250 Hz	250 Hz
# Subjects	7	9	9
# Sessions per subject	4	2	5
# Trials per session	100	288	120 for first 2 sessions and 160 trials for last 3 session
Total trials for each subject	400	576	720
Total trials in the dataset	2800	5184	6480

Table 4.5: Summary of the 3 datasets used in this study: Our experimental dataset, BCI 2a, and BCI 2b

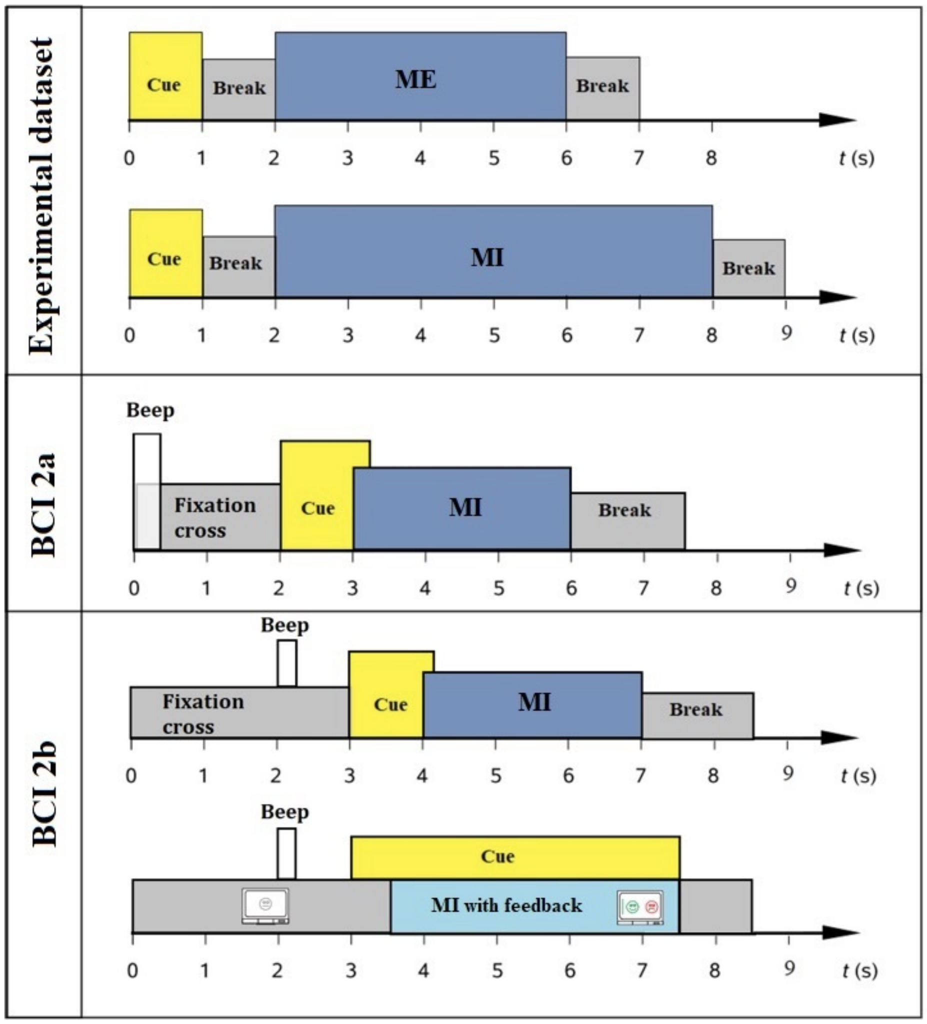


Figure 4.2: The experimental paradigms for our experimental dataset, BCI 2a, and BCI 2b.

and the foot's MI action is typically captured by the CZ electrode.

Configuration (2) The brain's frontal, central and parietal lobes are important from a neurological perspective for MI commands. We therefore also focused on these 7 electrodes (i.e. F3, F4, C3, CZ, C4, P3 and P4), which reside above these lobes of interest according to the 10-20 standard are considered in criteria 2 [90].

Configuration (3) Electrodes that are generally placed around the left and right motor cortices are included in this configuration because they are related to MI. According to 10-20 electrode montage [90], 18 electrodes lie around motor cortex. These are labelled C5, C3, C1, C2, C4, C6, CP5, CP3, CP1, CP2, CP4, CP6, P5, P3, P1, P2, P4 and P6 [162, 163].

Configuration (4) We used all 64 EEG channels.

In Figure 4.3, we showed these four configurations.

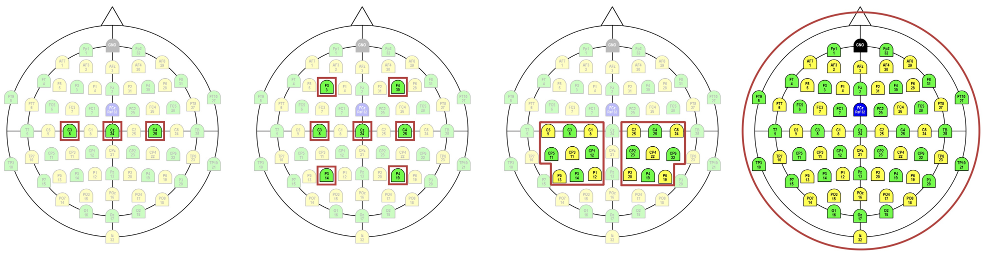


Figure 4.3: Four different electrode configurations on the actiCAP—which included 3, 7, 18, and all 64 electrodes

4.5 Results

4.5.1 Performance of the proposed CNN (Neural architectures vs. Neural architectures)

To evaluate the performance of our proposed CNN, we conducted comparisons between the Dense NN, Conv Net-Dense NN, Dai et al. [51], and Conv Net-Attention-Dense NN (Figure 4.4). The baseline Conv Net is identical to the Conv Net-Attention-Dense NN but lacks the attention module (see Methods, Figure 4.1, Table 4.1). The dense network sends all channels through 2 dense layers, then it concatenates all the vectors into a single one and sends that through 2 more dense layers. We used SHERPA for hyperparameter optimization for all 4 types of networks [79]. We also reproduced the proposed NN in [51] without the use of DA to compare it with the proposed CNN with the attentional mechanism.

Table 4.6 represents the classification results of our proposed CNN (with the attentional mechanism) without DA and with DA, which resulted in the highest accuracy for both

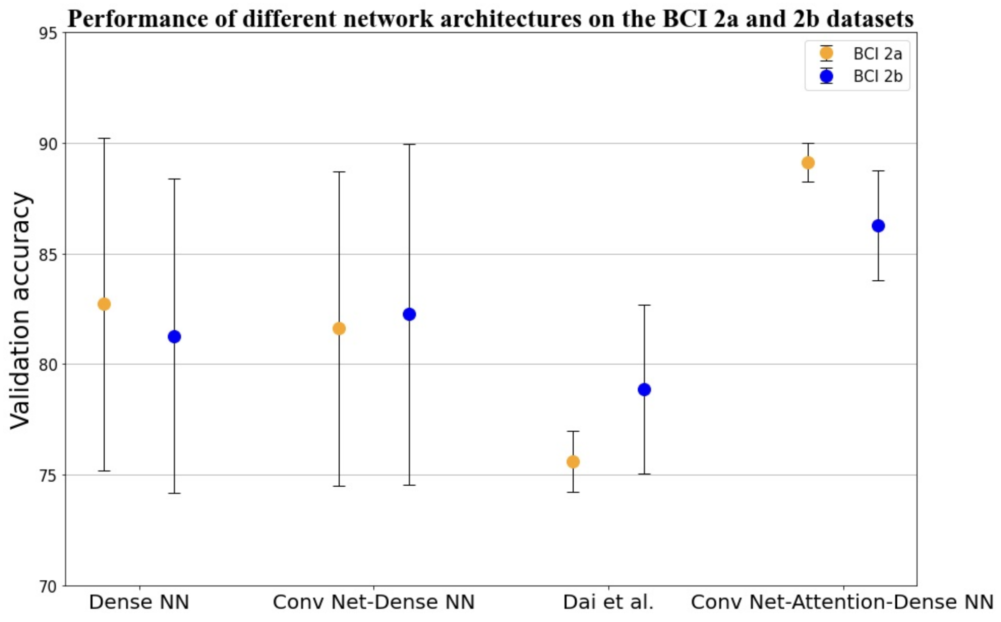


Figure 4.4: Comparison the average validation accuracy (SE) on the BCI 2a and BCI 2b datasets with Dense, CNN, Dai et al. (2020), and CNN-Attention-Dense (See section 2.1 and 2.2).

datasets. Those are further compared against the results of Dai et al. [51]. All classifications were carried out on the BCI 2a and BCI 2b datasets. The average accuracy in Dai et al. (2020) for BCI 2a and BCI 2b were 91.57% (±5.73) and 87.6% (±8.48), respectively. In comparison, our proposed method with DA (GAN and $m = 15$) achieved an average accuracy of 93.6% (±2.59) for BCI 2a and 87.83% (±6.34) for BCI 2b. Hence, our method has a higher average accuracy than Dai et al. (2020) while maintaining less variability in the accuracy across participants for both datasets. For the BCI 2a, our proposed method was 90.54% or higher for all participants while Dai et al. (2020) got this accuracy just for 5 of 9 participants (56%). Furthermore, we reproduced the NN described in [51] without the use of DA to compare with our proposed CNN with the attentional mechanism without DA. Our results on the BCI 2a and 2b datasets were 89.11% (±3.77) and 86.28% (±7.41), respectively, outperforming those of [51] at 75.61% (±14.63) and 78.88% (±11.42), respectively. Again, our results were also less variable than theirs.

Table 4.7 further compares our results with various other state-of-the-art methods. As is

	BCI 2a				BCI 2b			
Participant	Dai et al. (2020) [51]	Reproduced the result in [51] (without DA)	Proposed method without DA	Proposed method with DA (GAN m=15)	[51]	Reproduced the result (without DA)	Proposed method without DA	Proposed method with DA (sliding window m=2)
1	90.07%	69.77%	91.58%	95.38%	80.50%	70.83%	81.64%	84.13%
2	80.28%	65.62%	89.67%	91.25%	70.60%	63.24%	73.17%	77.92%
3	97.08%	97.91%	91.89%	91.25%	85.60%	62.64%	81.50%	83.64%
4	89.66%	69.45%	90.05%	96.12%	94.60%	97.84%	98.61%	99.18%
5	97.04%	62.51%	91.28%	95.05%	98.30%	80.95%	93.83%	94.97%
6	87.04%	62.48%	90.97%	94.62%	86.60%	80.28%	85.22%	85.83%
7	92.14%	66.66%	81.38%	91.22%	89.60%	84.58%	86.57%	86.57%
8	98.51%	90.64%	91.20%	90.54%	95.60%	86.05%	89.90%	90.50%
9	92.31%	95.46%	83.95%	97.50%	87.40%	83.47%	86.05%	87.73%
AVG	91.57%	75.61%	89.11%	93.60%	87.60%	78.88%	86.28%	87.83%
S.D.	5.73	14.63	3.77	2.59	8.48	11.42	7.41	6.34
S.E.	1.91	4.87	1.26	0.87	2.83	3.81	2.47	2.11

Table 4.6: Participant-by participant comparison of the proposed CNN with attentional mechanism—with and without DA— against Dai et al. [51] results on the BCI 2a and BCI 2b datasets

apparent from the table, our results outperform all others, typically by a wide margin. On average, our method is 16.44% and 7.21% more accurate than the other method for the 2a and 2b datasets, respectively. What is more, even without DA, our method has a higher average accuracy than all other methods except for Dai et al. (2020). And, with DA, our method beats all other methods, including Dai et al.'s.

4.5.2 Properties of our collected dataset

There are several available BCI datasets [40, 111, 33]. However, we wanted to investigate several open questions in neuroscience and BCI that were outside the scope of the available datasets. So, we took the time and effort to collect our own dataset, which we are now sharing with the community. First, we wanted to test and directly compare the performance of our proposed attentional CNN on ME, MI, and their combination. In particular, we wanted to track the decoding accuracy over time via a sliding-window approach. We therefore increased the duration of the motor-imagination period from 2-3s to 4-6s to gain more insight and track

	[174]	[10]	[161]	[121]	[6]	[118]	[157]	[215]	[116]	[156]	[66]	[67]	[51]	Proposed method (without DA)	Proposed method (with DA)
Dataset	2b	2b	2b	2a/2b	2b	2b	2b	2b	2a	2a/2b	2a/2b	2a	2a/2b	2a/2b	2a/2b
S1	77.0	70.0	80.0	63.69/73.2	84.6	81.0	76.0	72.5	88.9	90.28/70.3	66.7/62.8	91.5	90.07/80.5	91.58/81.64	95.38/84.13
S2	64.5	60.0	66.0	61.97/67.5	66.3	65.0	65.8	56.4	51.4	57.64/50.6	63.9/67.1	60.6	80.28/70.6	89.67/73.17	91.25/77.92
S3	61.0	61.0	53.0	91.09/63	62.9	66.0	75.3	55.6	96.5	95.14/52.8	77.8/98.7	94.2	97.08/85.6	91.89/81.50	91.25/83.64
S4	96.5	97.5	98.5	61.72/97.4	95.8	98.0	95.3	97.2	70.1	65.97/93.8	63.2/88.4	76.7	89.66/94.6	90.05/98.61	96.12/99.18
S5	82.0	92.8	93.5	63.41/95.5	89.2	93.0	83.0	88.4	54.9	61.11/63.8	72.2/96.3	58.5	97.04/98.3	91.28/93.83	95.05/94.97
S6	84.5	81.0	89.0	66.11/86.7	97.9	88.0	79.5	78.7	71.5	65.28/74.1	70.1/75.3	68.5	87.04/86.6	90.97/85.22	94.62/85.83
S7	75.0	77.5	81.5	59.57/84.7	82.1	82.0	74.5	77.5	81.3	61.11/61.9	64.6/72.2	78.6	92.14/89.6	81.38/86.57	91.22/86.57
S8	91.0	92.5	94.0	62.84/95.9	86.3	94.0	75.3	91.9	93.8	91.67/83.1	76.4/87.8	97.0	98.51/95.6	91.20/89.90	90.54/90.50
S9	87.0	87.2	90.5	84.46/92.6	97.1	91.0	73.3	83.4	93.8	86.11/77.2	77.1/85.3	93.9	92.31/87.4	83.95/86.05	97.50/87.73
AVG	80	80	83	68.32/84.1	84.7	84	77.6	78	78.01	74.92/69.7	70.2/81.6	79.93	91.57/87.6	89.11/86.28	93.60/87.83
S.D.	1.3	1.5	1.6	1.3/1.5	1.4	1.3	0.9	1.6	1.9	1.7/1.6	0.7/1.4	1.7	0.6/0.9	0.4/0.8	0.3/0.7

Table 4.7: Comparison of our proposed method (with and without data augmentation) with other state-of-the-art methods. All methods were run on the same dataset (BCI 2a and/or BCI 2b)

the changes in decoding accuracy over time.

Second, BCI datasets typically instruct subjects to make trivial movements, such as pressing a button. We wanted to test our subjects on a less trivial paradigm, that requires them to exert some force. We therefore had our subjects squeeze a tennis ball (ME) or imagine doing that (MI). We expected this to make our classifier more robust against variety of MI tasks. This is vindicated by recent evidence that decoding attempted handwriting movements results in much higher accuracy than attempted typing [201].

Third, most of the BCI datasets for MI focused on electrodes above the motor region—such as C3, C4, and Cz [111]. We wanted to test to what degree general, high-density EEG recordings across the cortex (to the extent that those brain regions are accessible to EEG) contribute to the performance of an MI classifier. This also let us investigate the extent to which channel selection is useful in MI classification. Forth, an additional goal of our study was to evaluate the role of DA in MI classification. So, we needed a large enough dataset to be able to compare classification results when training our classifier on only a portion of the dataset. Altogether we recorded 400 trials per subject (200 each for ME and MI, see

Methods).

4.5.3 Motor Imagery vs. Motor Execution

MI could be described as kinesthetic anticipation of corresponding overt ME without producing an actual motor output. Jeannerod stated that MI is functionally equivalent to its ME counterpart [87]. More specifically, MI is related to the preparation of ME and represents meaningful neurophysiological dynamics of human motor functions [217]. Consequently, both MI and ME are accompanied by activation in common sensorimotor areas, such as the primary motor area (M1), supplementary motor area (SMA), and premotor cortex (PMC) [87, 217]. The neurophysiology underlying MI may differ in healthy people and patients with motor-impairing conditions [117]. MI-based BCI may further augment the motor learning process in healthy participants [159]. What is more, in patients with impaired motor functions, MI is often the only viable option to drive rehabilitative BCI, because these patients cannot perform overt ME [117]. The individuality and severity of motor impairments impact the underlying neurophysiology; for example, post-stroke neurophysiology relies on lesion locations [134]. Additional work is needed to further delineate the roles of MI and ME in motor learning or relearning for both healthy and impaired participants to refine the design of BCI for supplementing the motor learning process.

Our own dataset enables us to directly compare ME and MI within each participant. In our task, the participants were presented with the cue for $1s$, then saw a blank screen for $1s$, and finally began ME or MI for $4s$ (see Methods). However, Dai et al. (2020), only used $2s$ of MI. To better compare our results to theirs, we ran a sliding window analysis only for the first $2s$ of the 4-s-long ME or MI period. We used window sizes of $100ms$, $300ms$, $500ms$, $1s$, and $2s$, with the step size fixed at $100ms$ (see Figure 4.4 and Methods) on the data from all 64 channels. With this analysis, we would expect to see a rise in the accuracy leading up to the moment when the participants needed to begin ME or MI. Further, as participants were supposed to execute or imagine the movement for $4s$, we expected the accuracy to then generally plateau over this after the above rise (similarly to [167] for example).

The left column in Figure 4.5 represents the average validation accuracy over all 7 participants and the right column is specifically for Participant 4. Both show the accuracy of the running-window analysis and over the first $4s$ after cue onset for 3 analyses: ME only, MI only, and the combination of ME and MI trials. The window shown at the $4s$ mark is from 3900 to $4000ms$ for the $100ms$ window, for 3700 to $4000ms$ for the $300ms$ window, and so on.

Our method's accuracy on ME is greater than on MI (Figure 4.5), which is consistent with previous findings about ME versus MI [53]. The average validation accuracy for the combination of MI and ME (All) is also greater than MI. Looking at the variability among the different window sizes, we see more variability in the ME condition than the MI or combined condition, on average. Our averaged results over all participants also align with our expectations, in that the accuracy rises from chance toward the beginning of the ME and MI periods and then generally plateaus (again, compare with [167]).

4.5.4 Channel selection

Analyzing dense-array EEG is computationally expensive and complex; it also typically requires more expensive EEG systems than those with sparser electrodes. Therefore, in this study we tested 4 different electrode configurations on our participants—which included 3, 7, 18, or all 64 electrodes (see Methods)—to further test the effect of channel selection on classification accuracy for MI in our own dataset.

The validation accuracy of the 7 participants for the 4 different channel-configurations are

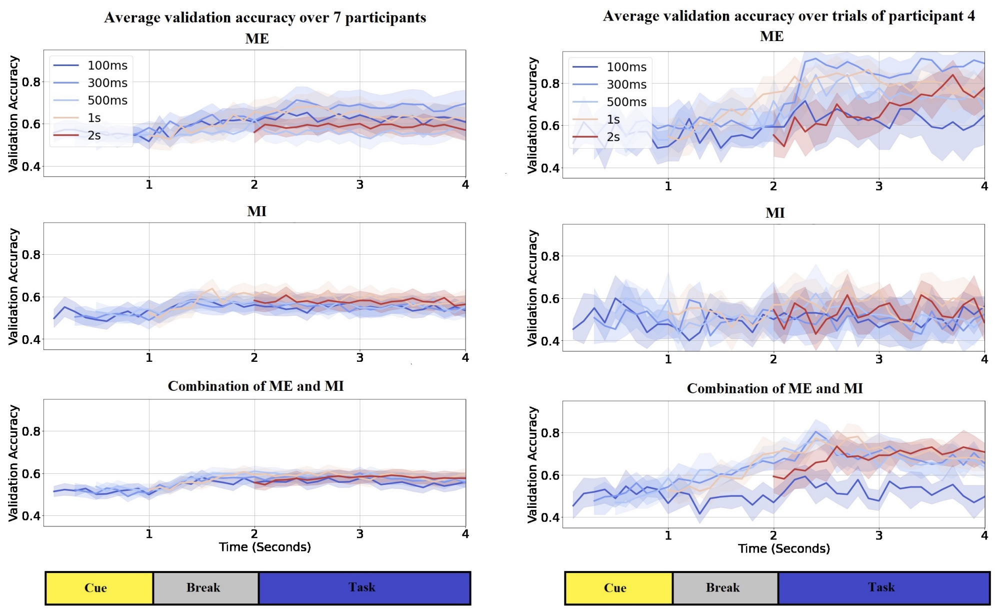


Figure 4.5: Validation accuracy of sliding-window analysis in ME (top), MI (middle), and ME and MI combined (bottom). The left column is the accuracy over time averaged across all 7 participants. The right column depicts the accuracy for the participant with the highest overall accuracy in the ME condition (Participant 4).

shown in Figure 4.6. In Table 4.8, the validation accuracy for each participant and the average accuracy across all participants are shown. The 18-channel layout had the highest accuracy, at 81.73% (±2.5).

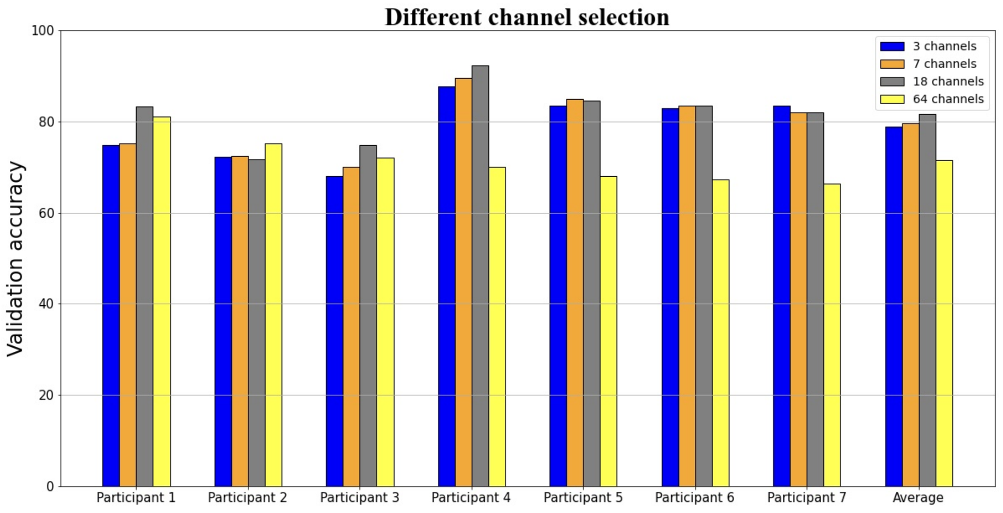


Figure 4.6: Validation accuracy for different channel configurations on the 7 participants of our dataset.

Participant	3 channels	7 channels	18 channels	64 channels
1	74.75(± 4.3)	75.25(±2.2)	83.25(±4.1)	81.18(±8.9)
2	72.25(± 4.2)	72.50(±4.9)	71.75(±4.1)	75.22(±4.3)
3	68.01(± 3.9)	70.01(±4.1)	74.75(±4.3)	72.05(±3.2)
4	87.69(± 5.4)	89.62(±3.2)	92.31(±3.6)	70.03(±3.1)
5	83.50(± 6.3)	85.01(±3.3)	84.50(±5.7)	68.08(±2.2)
6	83.00(± 4.2)	83.50(±6.7)	83.51(±5.8)	67.33(±1.6)
7	83.50(± 3.4)	82.01(±6.7)	82.01(±5.9)	66.41(±2.4)
AVG (± S.E.)	78.95(±2.7)	79.70(±2.7)	81.73(±2.5)	71.47(±1.9)

Table 4.8: Validation accuracy for different channel selections on our dataset for single participants and the average over all participants. For each participant, we present mean ± SE over trials. In the bottom row, we present mean ± SE over participants.

4.5.5 Data Augmentation

We used 5 types of DA for the MI task: noise addition [112, 144], GAN [72, 204, 212, 213], sliding window [169, 124, 185], Fourier transform [214], and recombination of segmentation [51]. Table 4.9 represents the result of different DA techniques on the BCI 2a, BCI 2b and our dataset for 64 channels and 18 channels. We evaluate all DA techniques with magnification factor $m = (2, 5, 10, 15, 20, 30, 50)$ for the proposed CNN. For Fourier transform, we used the same technique as in [214]. For noise addition, we opted for Gaussian noise with $\mu = 0$, $\sigma = (0.1, 0.2, 0.5)$.

cGANs allow generation based on a class assignment [79]. In this study, the GAN had 2 different conditions that were implemented: In order to provide context about the task, the first GAN model generates a sample conditioned on the participant's decision—i.e., left vs. right. The second GAN model applies finer granularity by conditioning not only on left vs right but also the electrode channel. When generating data, the conditional inputs provide additional information and allow the model to tailor its outputs with greater detail (see Table 4.4). Figure 4.7 illustrates the architecture of cGAN in our work.

We also evaluated sliding-window technique (lengths $l = 1000ms$ with sampling frequency 250 Hz and step-size $100ms$). Table 4.9 demonstrated that GAN (conditional left vs. right

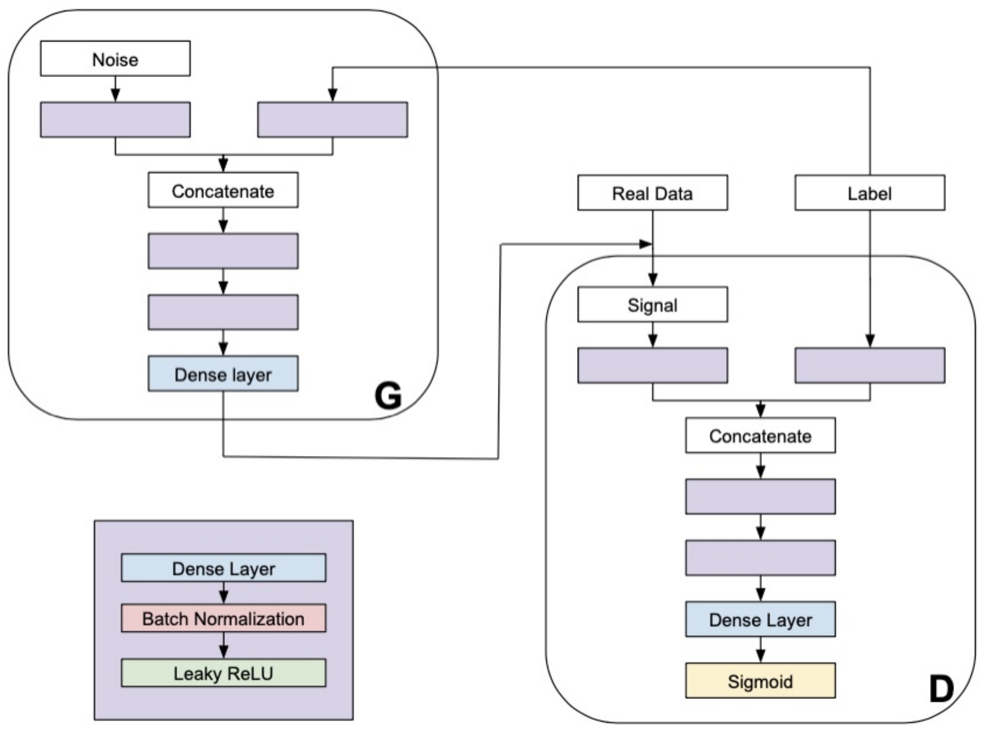


Figure 4.7: Our proposed cGAN model. In the generator (G), the prior input noise and label are combined into a hidden representation. In the discriminator (D), Real Data (i.e., raw EEG data) and the Label are presented as inputs to a discriminative function. The contents of all purple boxes in the architecture are the same and are expanded at the bottom left.

and channels) with $m = 15$ resulted in the best accuracy (93.6%) for BCI 2a dataset while Sliding Window (500ms windows and 100ms step size) with $m = 2$ achieved the best accuracy (87.83%) for BCI 2b dataset. For our dataset, Fourier Transform with $m = 15$ for 64 (86.61%) and 18 (83.42%) channels, respectively. The BCI 2a dataset had a magnification factor of 15 for the best result compared to a magnification factor of only 2 for BCI 2b. This might be because we did not include neurofeedback within our experimental paradigm. Decoding neurofeedback dataset has less complexity which is why BCI 2b dataset was seen to have a smaller magnification factor of 2. Our dataset did not include neurofeedback in the paradigm similarly to the BCI 2a dataset.

Dataset	DA Techniques		Fourier Transform	Noise Addition			GAN		Sliding Window
	parameter for each		(EMD)	σ=0.1	σ=0.2	σ=0.5	Conditional (left vs. right)	Conditional (left vs. right and channels)	Sliding Window of length 1s (step-size: 100 ms)
BCI 2a	Magnification Factor	2	0.8671	0.9056	0.8982	0.8768	0.9133	0.9025	0.8948
		5	0.8652	0.8999	0.8849	0.8908	0.9240	0.9092	0.8904
		10	0.8822	0.8902	0.8920	0.8721	0.9087	0.9217	0.8992
		15	0.8858	0.8988	0.8756	0.8750	0.9358	**0.9360**	0.8949
		20	0.8932	0.8898	0.8975	0.8904	0.9193	0.9300	0.9092
BCI2b	Magnification Factor	2	0.8535	0.8647	0.8614	0.8575	0.7939	0.8511	**0.8783**
		5	0.8391	0.8746	0.8696	0.8558	0.7747	0.8624	0.8747
		10	0.8339	0.8677	0.8668	0.8560	0.7733	0.8582	0.8726
		15	0.8228	0.8660	0.8717	0.8551	0.7601	0.8646	0.8749
		20	0.8217	0.8736	0.8677	0.8535	0.7611	0.8708	0.8691
Our Dataset (64 channels)	Magnification Factor	2	0.8442	0.8146	0.7548	0.7720	0.7914	0.8159	0.7904
		5	0.8305	0.7743	0.7844	0.7897	0.8377	0.7945	0.7933
		10	0.8377	0.7907	0.7885	0.7793	0.8024	0.8044	0.8033
		15	**0.8661**	0.7775	0.7541	0.7556	0.8184	0.7824	0.8362
		20	0.8560	0.7521	0.7826	0.7886	0.7994	0.8052	0.7990
Our Dataset (18 channels)	Magnification Factor	2	0.8124	0.8051	0 .8056	0.8079	0.8045	0.8174	0.8190
		5	0.8010	0.8179	0.8121	0.8090	0.7969	0.8156	0.8224
		10	0.7988	0.8123	0.8162	0.8048	0.7965	0.8020	0.8312
		15	0.7954	0.8203	0.8141	0.8047	0.7842	0.8015	**0.8342**
		20	0.7963	0.8209	0.8051	0.8048	0.7875	0.8102	0.8277

Table 4.9: Comparison of different DA techniques with different magnification factors and hyperparameters for BCI 2a, BCI 2b, and our experimental dataset (for 64 channels and 18 channels)

4.5.6 Different portions of dataset

A dearth of data is a common problem when training machine-learning models on neuroimaging data. We therefore wanted to systematically test to what degree DA can compensate for the reduced availability of data. We thus randomly selected 100%, 75%, 50%, or 25% of the samples in our dataset. And we tested the accuracy of DA on these different proportions of our dataset for different DA techniques and magnification factors (Table 4.10). Fourier transform resulted in the best accuracy for 100%, 75%, and 50% of the data, with 86.61%, 88.26%, and 86.18% accuracy, under magnification factors 15, 5, and 10, respectively. When using only 25% of the data, GAN (conditional left vs. right and channels) was the best DA technique in terms of accuracy, with 82.18% and a magnification factor of 15.

Proportion of Dataset	DA Techniques		Fourier Transform	Noise Addition			GAN		Sliding Window
	parameter for each DA		(EMD)	σ=0.1	σ=0.2	σ=0.5	Conditional (left vs. right)	Conditional (left vs. right and channels)	Sliding Window of length 125
100%	Magnification Factor	2	0.8442	0.8146	0.7548	0.7720	0.7914	0.8159	0.7904
		5	0.8305	0.7743	0.7844	0.7897	0.8377	0.7945	0.7933
		10	0.8377	0.7907	0.7885	0.7793	0.8024	0.8044	0.8033
		15	**0.8661**	0.7775	0.7541	0.7556	0.8184	0.7824	0.8362
		20	0.8560	0.7521	0.7826	0.7886	0.7994	0.8052	0.7990
75%	Magnification Factor	2	0.8644	0.7975	0.7886	0.8129	0.7772	0.7927	0.7695
		5	**0.8826**	0.7856	0.7877	0.7987	0.7997	0.8045	0.7998
		10	0.8707	0.8096	0.7743	0.7921	0.8040	0.7950	0.7980
		15	0.8732	0.7735	0.8013	0.7741	0.7780	0.8057	0.8104
		20	0.8625	0.8066	0.7838	0.7814	0.8223	0.8159	0.8158
50%	Magnification Factor	2	0.8346	0.8116	0.7957	0.7909	0.7743	0.7560	0.7669
		5	0.8536	0.7672	0.7687	0.7820	0.7754	0.8063	0.7656
		10	**0.8618**	0.8067	0.8222	0.7695	0.8034	0.7943	0.7503
		15	0.8474	0.8037	0.7969	0.7687	0.7671	0.8151	0.7426
		20	0.8128	0.7560	0.8010	0.7539	0.8247	0.8069	0.8039
25%	Magnification Factor	2	0.7422	0.7980	0.7868	0.8057	0.7595	0.7731	0.7387
		5	0.7683	0.8016	0.7569	0.7755	0.7714	0.7821	0.7202
		10	0.7417	0.7838	0.7767	0.8087	0.7643	0.8204	0.7256
		15	0.7909	0.7643	0.8187	0.7584	0.7737	**0.8218**	**0.7138**
		20	0.7826	0.7643	0.7814	0.7513	0.7731	0.7982	0.7501

Table 4.10: Accuracies for different proportions of our dataset with different DA techniques

4.5.7 Combination of participants' EEG signals

The variability in brain anatomy and even more so functionality among different individuals is well known, e.g. [131]. Strong structure-function correspondences is therefore typically derived only at the aggregate level [82]. For example, Smith et al. delineated structural differences, suggesting that the number of folds and thickness of the cortex could be associated with whole-brain functional network [177]. Furthermore, inter-participant variability in brain topography may also occurs due to participant-specific cognitive styles and the strategies that different participants use to perform the task [151]. This might augment the underlying learning processes—e.g., motor and perceptual learning [80]. Intra- and inter-participant variability might be explained by scale-dependent brain networks in spatial, temporal and topological domains [30].

Motor variability due to variability in human kinematic parameters—e.g., force field adaptation, speed and trajectory, and motivational factors such as level of user engagement, arousal and feelings of competence, necessary for performing a motor task—is an integral part of the motor learning process [56, 59, 165]. What is more, EEG signals are of course measured from the scalp rather than directly inside the brain, so they suffer from various signal distortions and technical limitations [119]. Given the above, the extent to which machine-learning models can be transferred between participants is not completely understood. The EEG patterns associated with motor variability could partly explain intra-individual variability in SMR-based BCI [136]. The neurophysiological processes underpinning the SMR often vary over time and across participants. Inherent intra- and inter-participant variability causes covariate shift in data distributions that impede the transferability of model parameters among sessions/participants.

Given the above, we evaluate the performance of the proposed NN on combinations of data across participants. The validation accuracy was averaged over every possible combination for each dataset—e.g., all participant pairs, all triplets, etc. After finding all the possible combinations, the data was split into training and test for each combination to compute the validation accuracy. The averages of the validation accuracy over all the states for the three datasets are reported in Figure 4.8 (top) and differences between group (bottom). As we add more participants, the accuracy decreases—but the decreases become smaller. In Figure 4.8 (bottom), for the BCI 2a and 2b datasets, after combining 6 or more participants, we can see the curves plateau. This suggest that our proposed CNN was able to learn the important variations of the different EEG signals among the different subjects thus achieving stable accuracy.

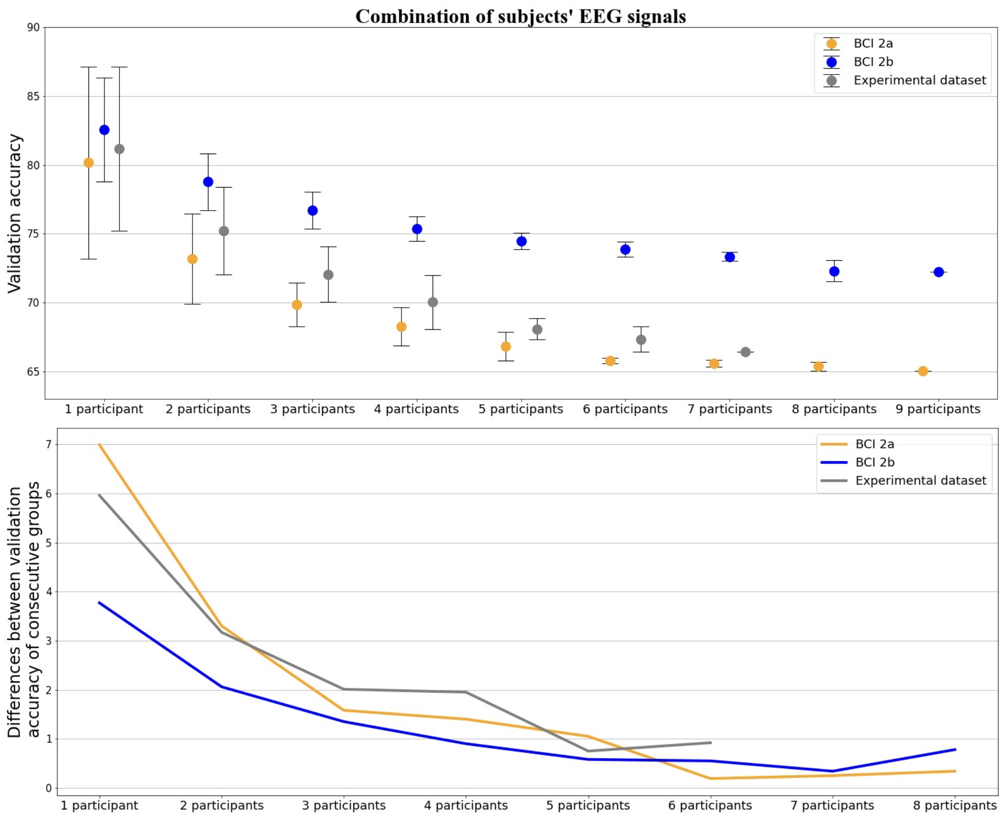


Figure 4.8: (Top) Validation accuracies for combinations of participants for BCI 2a, BCI 2b, and our experimental dataset. (Bottom) line plots of differences between mean validation accuracies of consecutive groups for the 3 datasets. The x axis labels are the smaller groups; so, differences between 2 participants and one are plotted above the label ”1 participant”, between 3 and 2 participants above ”2 participants”, and so on.

4.5.8 Leave-one-participant out and transfer learning

This subsection addresses two separates but closely related tasks. The first, leave-one-out, trains a NN on $n-1$ participants and tests on the remaining n^{th} participant. This task addresses the question of how information is shared between different participants' EEG signals (see section 3.7 Figure 4.6, on the x-axis, 8 participants for BCI 2a, BCI 2b and 6 participants for our dataset).

The second task, transfer learning, pretrains a NN on $n-1$ participants and fine-tunes to the n^{th} participant [152]. The pre-training phase orients the network weights to extract meaningful representations from the data. Then the fine-tuning, where the learning rate is decreased, adjusts to the task of interest, the n^{th} participant. For transfer learning, 10-fold cross validation over the n^{th} participant was used. Each fold fine-tunes on 9 folds and tests on the held-out 10^{th} fold. Table 4.11 shows the result of transfer learning on the BCI 2a, BCI 2b, and our dataset (64 channels and 18 channels). Figure 4.9 compared the result with and without transfer learning for all 3 datasets. For instance, the validation accuracy without transfer learning on participant n is defined by the trained model based on combination of the other $n-1$ participants and is tested on the complete dataset of participant 9. However, the validation accuracy with transfer learning on participant n is tuned to the trained model based on combination of the other $n-1$ participants based on 10% of the n^{th} participant and is tested on 90% of participant n.

4.6 Discussion

In this study, we proposed an end-to-end CNN architecture for EEG-based MI classification. This proposed mechanism is used to automatically extract features from raw EEG data (Figure 4.1 and Table 4.1). The NN optimization used the SHERPA Bayesian hy-

Train (participants index)	Finetune (participant index)	BCI 2a (with transfer learning for different participants)	BCI 2b (with transfer learning for different participants)
2-3-4-5-6-7-8-9	1	78.12	78.75
3-4-5-6-7-8-9-1	2	76.38	71.62
4-5-6-7-8-9-1-2	3	89.53	79.17
5-6-7-8-9-1-2-3	4	77.77	97.02
6-7-8-9-1-2-3-4	5	77.41	83.10
7-8-9-1-2-3-4-5	6	78.83	81.94
8-9-1-2-3-4-5-6	7	80.58	81.67
9-1-2-3-4-5-6-7	8	81.60	87.36
1-2-3-4-5-6-7-8	9	90.63	84.44
Train (participants index)	**Finetune (participant index)**	**Our dataset 64 channels**	**Our dataset 18 channels**
2-3-4-5-7-6	1	83.25	83.75
3-4-5-6-7-1	2	73.01	87.25
4-5-6-7-1-2	3	76.50	77.50
5-6-7-1-2-3	4	91.15	92.70
6-7-1-2-3-4	5	91.01	85.50
1-2-3-4-5-7	6	82.10	82.50
1-2-3-4-5-6	7	84.50	86.50

Table 4.11: Leave-one-out and transfer-learning validation accuracy for BCI 2a, BCI 2b, and our dataset (64 and 18 channels)

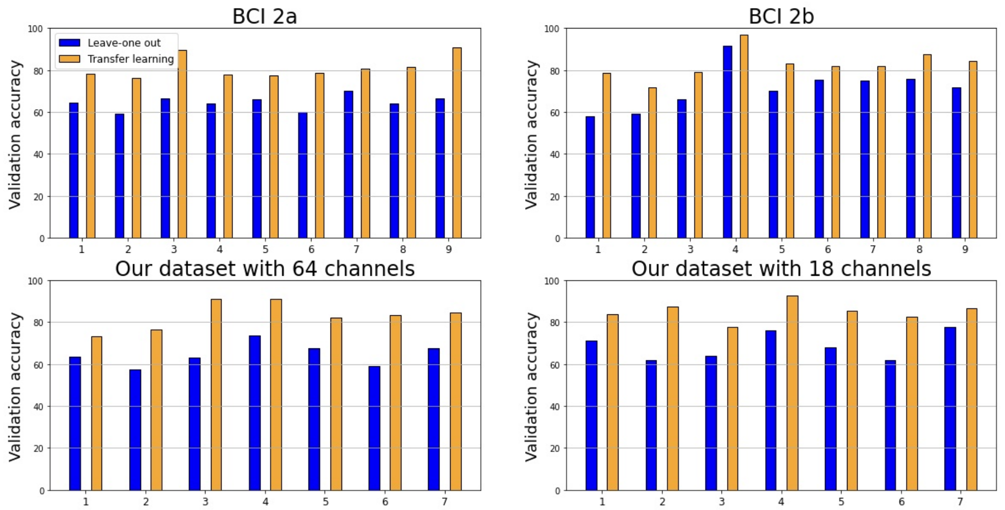


Figure 4.9: Validation accuracy for BCI 2a, BCI 2b, and our dataset (64 and 18 channels) with and without transfer learning.

perparameter search on 3 datasets: the BCI Competition IV 2a and BCI Competition IV 2b, which have become benchmarks in the field, and a dataset that we collected ourselves (Figure 4.2; see Methods). We began by comparing the architecture we favored, Conv Net-

Attention-Dense NN, to two other baseline architectures—a Dense NN and Conv Net-Dense NN—as well as to what was, to the best of our knowledge, the top result in the field on the benchmark datasets—the architecture described in Dai et al. (2020) (see Figure 4.4). Our CNN-Attention-Dense achieved 93.6% (S.E.: ±0.87) and 87.8% (S.E.: ± 2.11) accuracy over the BCI 2a and 2b datasets, respectively (Table 4.6). That is 6.4% to 13.5% and 4.03% to 5% better than the other architectures for BCI 2a and 2b, respectively (Figure 4.4). We further compared our results with all the papers we could find that classified the BCI 2a and 2b datasets and reported participant-by-participant results. For the BCI 2a dataset, our proposed EEG MI classification method achieved an improvement of 2.03% to 25.28% over all other methods (Table 4.7). For the BCI 2b dataset, our proposed method achieved an average improvement of 0.23% to 18.13% over previous methods (Table 4.7).

To the best of our knowledge, our CNN-Attention-Dense architecture achieved the highest accuracy thus far for the 2 benchmark datasets—BCI 2a and 2b. On top of that, an additional strength of our approach is its automated features extraction, directly from raw EEG. This contrasts with most methods, which tend to use handcrafted features and require heuristic parameter setting (e.g., predefined frequency bands). Automated features have the advantage of often generalizing better across tasks and participants [51]. Another potential advantage of our architecture is that the attentional mechanism could potentially lead to more interpretable results. However, we leave the explainable-AI facet of our architecture for further, future research.

The dataset that we collected for this study used 64 electrodes (according to the 10/20 montage; Figure 4.3). It included both ME and MI tasks and enabled us to compare the two tasks. Having all 3 datasets further enabled us to compare MI with and without neurofeedback training (datasets 2b and 2a, respectively) as well as imagining button presses versus squeezing tennis balls (datasets 2a and 2b versus our own dataset, respectively).

A long-standing question in neuroscience and motor control is the extent of shared neural mechanisms between MI and ME [53]; though there is a general consensus that MI and ME at least share some important neural mechanisms. This similarity has been used in the MI-decoding literature, where some attempts to decode MI have relied on ME as training data [167]. Our results suggest that it is easier to decode ME than MI, at least when using EEG and relying on our decoding methods (Figure 4.5). Furthermore, we found that, on average, the decoding accuracy started at chance and then rose toward the time that participants were required to move or to imagine moving. After that it more or less plateaued. Interestingly, though perhaps not surprisingly, the accuracy level at the plateau, when using sliding windows, was lower than the accuracy for the full 4 s of ME (compare Figures 4.5 and 4.6). A likely contributing factor to this is that the sliding-window analysis decoded the EEG over shorter time windows than the full 4 s.

Another long-standing question when decoding EEG, and especially dense-array EEG, relates to how many and which electrodes (or channels) to use when recording the task. On the one hand, when using all channels (64, in our case), the set-up time for the task is longer, analyzing the larger dataset is more complex and computationally expensive, and brain signals unrelated to the task and noise are perhaps more often introduced. On the other hand, using only a limited number of channels, there may not be full coverage of brain regions that may be involved in the decision-making and action-preparation processes. We therefore wanted to identify the appropriate channels relevant to the MI task. We thus selected different combinations of channels, according to 10-20 system standard, based on what is known about the neurophysiology of decision making and action formation, [167, 172, 13]. Hence we included different EEG configurations in our study (see Results), with 3, 7, or 18, channels around the motor cortex (see Methods), or with all 64 channels [90]. Our analysis suggests that, without DA, the 18-channels configuration had the best average accuracy (81.73% ± 2.5), at least on our dataset (Figure 4.6, Table 4.8), while using all 64 channels resulted in the worse accuracy (71.47% ± 1.9). Our results therefore suggest that, for MI decoding, it may be best to use only the 18 channels around the left and right motor region rather than all the channels. However, that result should be taken with a grain of salt, because when including DA, the tables were flipped, and it was the 64-channel configuration that did best, as described above.

One of the EEG configurations we tested included only 3 channels (C3, Cz, and C4)—this thus let us more directly compare our dataset to the two benchmark ones and the results of other studies. On those 3 electrodes, we achieved a mean accuracy of 79.95% for our dataset, while our analysis resulted in an accuracy of 89.11% and 86.28% for BCI 2a and 2b, respectively—all without DA. The higher accuracy for the benchmark datasets over our dataset might be due to the difference in tasks, the inclusion of neurofeedback (in BCI 2b), or that they perhaps ran participants who were better able to elicit good EEG data.

One general challenge of EEG decoding, especially with deep NNs, is obtaining enough data to train the numerous parameters in these large statistical models. The problem is compounded for MI tasks, because they are highly cognitively demanding. So, participants are easily fatigued and thus cannot produce a large amount of data in each experimental session. Bringing participants in for multiple sessions runs into issues of participant attrition for example. Another issue with collecting EEG over multiple session is the non-stationary nature of EEG signals [48]—i.e., the statistics of the EEG signals vary across time. As a result, a classifier trained at a specific time would tend to generalize increasingly poorly to data recorded at another time that was increasingly temporally removed—even for the same participant. This is a challenge for real-life applications of EEG, which must often work train on only limited amounts of data.

Some studies indeed strived for very lengthy data collection paradigms. One study, investigating MI control of 3D movement, had participants come back for up to 50 experimental sessions, which amounted to more than 20 hours of training per participant in some cases [125]. In another study, focusing on an EEG-based stroke-rehabilitation system [182], it took 12 weeks to collect enough data for three MI tasks, with each participant participating in

2 sessions per week [182]. While these are extreme examples, they highlight how common it is for participants to become fatigued after as little as 1 hour or less of data collection [9, 98, 184].

A promising solution to this dearth of data is to use DA, especially when using DL models on EEG data [103]. We therefore tested 5 disfferent DA techniques: sliding window, noise addition, GAN, Recombination of segmentation, and Fourier transform/wavelet. We further tested different magnification factors and hyperparameters (e.g., different window sizes for sliding window, various standard deviations for noise addition) for each technique we evaluated. Based on the guidelines in Lashgari et al. (2020) we evaluated the accuracy of the proposed method before and after DA. Our main objective was to find the best DA technique for each of the 3 datasets above. As far as we know, this is the first study to compare these various DA techniques as well as the different hyperparameters of the various techniques on benchmark datasets BCI 2a and 2b (see Table 4.9). We found that different techniques work best for different datasets. For BCI 2a, GAN (conditional left vs. right and channels, m = 15) achieved the best accuracy, 93.6%. In contrast, sliding window (m = 2) gave the best accuracy for BCI 2b, at 87.83%. The DA step thus clearly boosted the performance of our proposed CNN (Table 4.6) as discussed below.

Interestingly, the BCI 2a dataset did not include neurofeedback training for the participants, while BCI 2b did. At the same time, the DA method that worked best for BCI 2a was a highly complex GAN with a large magnification factor, while that for BCI 2b it was a simple sliding window with a small magnification factor. So, one possible conclusion is that the neurofeedback training in BCI 2b, which effectively trained the participants to emit neural activity that would be better classified by the classifier, may have led to the superior accuracy from a simpler DA technique.

We also tested different DA techniques on our own dataset, which included 64 channels (see Methods). This achieved an accuracy of 86.61% (m = 15) with Fourier transform (Table

4.9). Using only 18 channels and the sliding-window DA technique (m = 15), we achieved an accuracy of 83.42%. Hence, using DA, we achieved higher accuracy with 64 channels than with 18 channels. Interestingly, without DA, the situation was flipped: the 64-channel data had lower accuracy 71.47% (±1.9) than the 18-channel data 81.73% (±2.5) (see Table 4.8). This suggests that, if one dataset has lower accuracy than another without DA, it does not necessarily mean that the first dataset would also have lower accuracy than the second after DA.

As noted above, our accuracies were higher than those of Dai et al. (2020) (Table 4.6)—which was the top result in the field. Besides higher accuracies on average, our accuracies for individual participants were 90.54% or higher (Table 4.6), while Dai et al. (2020) achieved this accuracy or higher for just for 5 of the 9 participants. Further, we were interested in the effect of DA on the accuracy of their results. But they did not report that for BCI 2a. And we were unable to obtain their code. What is more, they did not specify the details of their DA techniques. We therefore reimplemented their architecture from their paper, as per the details in their methods, without DA, to compare it with our architecture without DA. The accuracy of our proposed CNN without DA—at 89.11% (±3.8; SE here and below) and 86.28% (±7.4)—outperformed the NN reproduced from Dai et al. (2020)—at 75.61% (±14.6) and 78.88% (± 11.4)—for BCI 2a and 2b datasets, respectively.

Following the above, an exciting potential use of DA is to replace lengthy, multi-session data-acquisition efforts [125, 182]. For brain-imaging studies, it would decrease the time and funds that researchers need to spend on data collection and reduce the inconvenience of participants. This is especially pertinent for situations where gathering additional data is financially, ethically, or otherwise difficult. Though DA would of course come at the expense of additional training time for the statistical models. We tested this by training on only some of the training set—25, 50, 75, or 100% (see Table 4.10)—while testing different DA techniques on the remaining data.

We therefore tested the extent to which data augmentation could replace gathering more data, at least for the dataset that we collected (Table 4.10). More specifically, we collected 400 trials from each participant (see Methods) and used different proportions of the MI dataset (100%, 75%, 50% and 25%) to train the model. We then augmented those different proportions of the dataset with various DA techniques that have different magnification factors. Our aim was to test the effects of those DA parameters on classification accuracy (Table 4.10). With 100%, 75% and 50% of the data, m = 15, 5, and 10, using Fourier transform achieved the highest accuracies, that were overall similar, at 86.61%, 88.26%, and 86.18% accuracy. Yet, classification based on just 25% of the data, m=15 and GAN (conditional left vs. right and channels) resulted in a lower accuracy, 82.18%. It might be that the smaller dataset required a more sophisticated DA technique that for the other proportions was needed to achieve its best accuracy. Though this accuracy was clearly lower than for the other proportions of data. This hints at the limits of DA for EEG.

It is well known that there is general anatomical similarity as well as structure-function correspondence among humans. But the anatomy of different brains also differs, at least to some extent, as does the structure-function correspondence. So, brain science typically operates at the aggregate level [82]. In particular, Smith et al. delineated structural differences, suggesting that the number of folds and thickness of the cortex could be associated with whole-brain functional networks [177]. Furthermore, inter-participant variability in topography occurs due to participant-specific cognitive style and strategy to perform a task over time [173], which could augment the underlying learning processes, e.g., motor and perceptual learning [80].

This question has clear implications for the analysis of EEG over groups of participants. We therefore wanted to investigate to what extent the number of participants over which we trained and tested our machine-learning model reduced the classification accuracy of the statistical model over that group. We thus trained and tested our model on all individual participants, on all pairs of participants, all triplets, quadruplets, and so on (Figure 4.8). It appears that, for all 3 datasets, the accuracy dropped most markedly between training and testing on individual participants to training and testing on pairs. Then there were diminishing decreases going from pairs to triplets, triplets to quadruplets, and so on, leading to roughly a plateau from groups of 6 participants and on. This suggests that the costs associated with inter-individual differences in brain structure and activity outweigh the benefits of the additional data when training over a group of participants. Though the decoding accuracy appeared to stop decreasing and reached somewhat of a plateau after around 6 participants. Future work, with a larger number of participants, could test the hypothesis that the accuracy would begin to rise again when training and testing over enough additional participants. One reason that this could happen is that the introduction of an ever-increasing number of additional participants might end up more than compensating for the neural variability between different brains. In other words, the advantages of the increasingly larger data available to train the model would outweigh the disadvantages of the variability across additional brains. Testing this hypothesis is left for future studies.

Following the discussion of inter-participant brain variability above, another key question in EEG analysis and especially for classification using DL is the extent to which a machine-learning model that was training on one group of participants could be generalized to new participants [166]. Put differently, we were wondering to what extent transfer learning, which has been increasingly used in the machine-learning literature, especially of late [110, 202, 211], would be useful for EEG classification using DL. We tested this by directly comparing two analyses. In the first, we trained a model on all but one participant and then tested it on that remaining participant (i.e., leave-one-participant-out classification). The second analysis comprised of again training on all but one participant, but then using transfer learning and finetuning the model on one part of the left-out participant. Finally, we tested the model on independent data from that participant (see Results 3.7). Our results clearly indicated that transfer learning led to higher accuracy than leave one out (Figure 4.9)—an increase in accuracy of 16.66%, 11.35%, and 18.6% for BCI 2a, BCI 2b, and for our dataset, respectively. This demonstrates the clear advantages of transfer learning for EEG analysis using DL. With DL models getting increasingly complex, the ability to finetune them for new participants rather than retrain them from scratch becomes increasingly important. In addition, our results suggest that the BCI community could use transfer learning with EEG to train a model on an existing dataset and then improve its performance for a new participant using only finetuning of the model [110, 58]. According to our results, this could markedly improve the performance of BCI classifiers.

Due to the good classification performance of our proposed neural-network architecture and the relatively simple data processing, without prior manual feature extraction, our method holds promise for online, real-time, EEG-based classification of MI. It is left to future work to test how well the system will work in real time. Further, based on our results, it seems useful to use transfer learning between participants in a real-time paradigm. Furthermore, our neural-network architecture uses an attentional mechanism that helps identify the most salient brain regions that drive the network's classification ability. However, we leave the analysis of these brain regions for future work.

Acknowledgements

This publication was made possible in part through the support of a joint grant from the John Templeton Foundation and the Fetzer Institute, BIAL foundation and Boston Scientific Corporation. The opinions expressed in this publication are those of the author(s) and do not necessarily reflect the views of the John Templeton Foundation or the Fetzer Institute. This publication was also made possible in part by the support of the Boston Scientific Investigator-Sponsored Research Program. The authors would like to acknowledge the assistance of Natalie Nichols and Kai Lee Hague for graphics in this research.

Ethical statement

The research was conducted in accordance with the principles embodied in the Declaration of Helsinki and in accordance with local statutory requirements. All subjects gave written, informed consent to participate in the study, which was approved by the Chapman University IRB (IRB-18-104).

Chapter 5

Conclusion

We have presented contributions to the field of deep learning. Both in theoretical understanding, motivated from biological plausibility constraints, as well as applications to problems in the physical sciences. Findings in the realm of deep learning theory have demonstrated that while weight sharing is a pragmatic convenience for time and memory efficiency it is not required from a learning perspective. These findings nicely intertwine with well known facts of biology and neurocircuitry.

Our contributions in applied machine learning have dramatically impacted the availability of deep learning resources in specific high performance computing landscapes. The FKB framework has become a useful tool for those seeking to apply neural networks in FORTRAN based simulators.

Appendix A

Appendix

A.1 Experimental Settings

A.1.1 Hyperparameter Search

We conducted a hyperparameter search to explore the space of possible architectures. CNN and FCNs trained on MNIST and CIFAR, using a grid search to find the optimal setting. The search was executed using SHERPA [76], a Python library for hyperparameter tuning. We detail the hyperparameters of interest in Table A.1, as well as the range of available options during the search.

During the hyperparameter search, MNIST trained for 100 epochs with a patience of 25 monitoring the validation accuracy. CIFAR trained for 200 epochs with a patience of 50

Name	Options	Parameter Type
Learning Rate	10^{-i}, $i \in [1, 6]$	Choice
Number of layers	[2, 3]	Discrete

Table A.1: Hyperparameter Space for the conducted grid search.

monitoring the validation accuracy. We show the hyperparameters of the best-performing networks in Table A.2. All networks achieved the best performance with three layers. The ultimate difference between MNIST and CIFAR architectures is the learning rate, 10^{-3} and 10^{-4}, respectively.

A.1.2 Network Architectures

Table A.2 describes the architectures used in this paper. MNIST networks have three convolutional or free convolutional layers respective of the architecture. All weight kernels are of size 3x3 with 32, 64, and 128 filters for the three layers. This output feeds into a fully connected layer of 1024 nodes, followed by a softmax layer for classification.

CIFAR networks have three convolutional or free convolutional layers respective of the architecture. Layer one has a 5x5 weight kernel, 64 filters, and a stride of 2. Layer two has a 5x5 weight kernels, 128 filters, and a stride of 2. Layer three has 3x3 weight kernels, 256 filters, and a stride of 1. Following these layers are a fully connected layer of 1024 nodes, followed by a softmax layer for classification.

A.1.3 Implementation Details

Random seeds were set to zero for the Tensorflow backend and Numpy. Batch sizes were 256 and 128 for MNIST and CIFAR, respectively. Weights for all layers were initialized using the Xavier Uniform Initialization. The source code for all experiments has been made publicly available at: `https://github.com/Learning-In-The-Machine/Weight-Sharing`

	CNN	FCN

MNIST	Convolutional(3x3,32,2)
	Convolutional(3x3,64,2)
	Convolutional(3x3,128,1)
	Fully Connected(1024)
	Softmax(10)
	
	Learning Rate: 10^{-3}
		Free Convolutional(3x3,32,2)
		Free Convolutional(3x3,64,2)
		Free Convolutional(3x3,128,1)
		Fully Connected(1024)
		Softmax(10)
		
		Learning Rate: 10^{-3}

CIFAR	Convolutional(5x5,64,2)
	Convolutional(5x5,128,2)
	Convolutional(3x3,256,1)
	Fully Connected(1024)
	Softmax(10)
	
	Learning Rate: 10^{-4}
		Free Convolutional(5x5,64,2)
		Free Convolutional(5x5,128,2)
		Free Convolutional(3x3,256,1)
		Fully Connected(1024)
		Softmax(10)
		
		Learning Rate: 10^{-4}

Table A.2: Architecture specification resulting from the grid search. The format for convolutional layers is (kernel size, number of output channels, stride). All layers, except the output, use ReLU activations.

A.2 Other Augmentation Methods

To examine the necessity of translational datasets, additional experiments were conducted using noise and rotational augmentation during training. The hypothesis being that only translational data is sufficient to endow FCNs with translational invariant recognition. The full results for edge noise, noise, and rotation augmentation can be found in Tables B.1, B.2, and B.3, respectively.

Training using other augmentation methods that do not produce translational invariant datasets yield poor results when testing on the translationally augmented validation set. This result indicates that only translationally augmented training data allows FCNs to learn translationally invariant representations. *Using translationally invariant datasets yields better results in validation set and augmented validation set accuracy, specifically in FCNs.*

A.2.1 Edge Noise

Augmentation with edge noise represents adding Gaussian noise to the periphery of an image. For experiments in this paper, the periphery is defined as the 5 pixels bordering an image (for both CIFAR and MNIST). This type of augmentation serves to test the network's resiliency to noise on the fringe. For datasets like MNIST and CIFAR that contain the object of interest in center focus, this noise is unlikely to corrupt the object. Figure B.1 shows the effect of edge noise on the periphery, starting from zero noise, Figure B.1a, up to a standard deviation of 0.99, Figure B.1k.

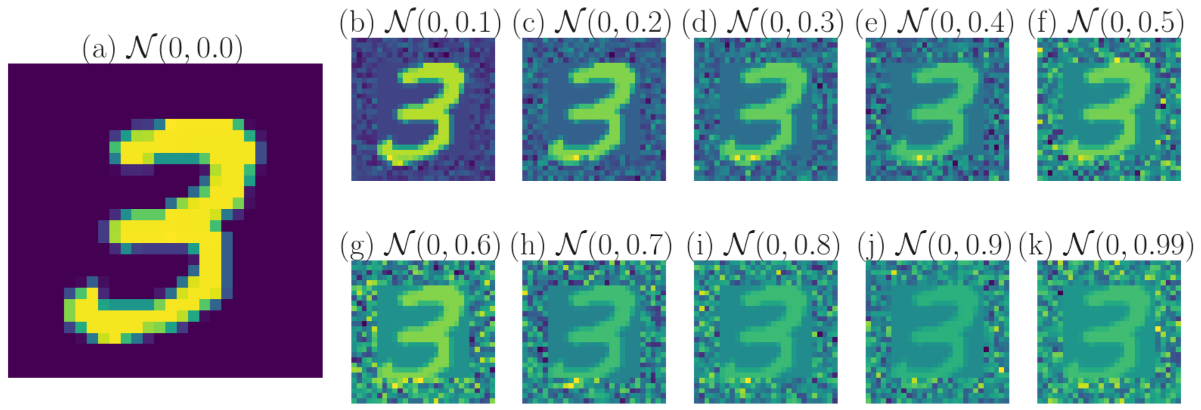

(a) $\mathcal{N}(0, 0.0)$
(b) $\mathcal{N}(0, 0.1)$
(c) $\mathcal{N}(0, 0.2)$
(d) $\mathcal{N}(0, 0.3)$
(e) $\mathcal{N}(0, 0.4)$
(f) $\mathcal{N}(0, 0.5)$
(g) $\mathcal{N}(0, 0.6)$
(h) $\mathcal{N}(0, 0.7)$
(i) $\mathcal{N}(0, 0.8)$
(j) $\mathcal{N}(0, 0.9)$
(k) $\mathcal{N}(0, 0.99)$

Figure A.1: Examples of edge noise augmentation on MNIST. (a) 0 variance noise, equivalent to a un-altered MNIST image. (b-k) Gradually increasing the variance of noise augmentation by .1 each time.

A.2.2 Noise

Augmentation with noise represents adding Gaussian noise to the image. This type of augmentation serves to test the network's resiliency to noise corruption. Figure B.2 shows the effect of edge noise on the periphery, starting from zero noise, Figure B.2a, up to a standard deviation of 0.99, Figure B.2k.

	MNIST				CIFAR			
	Validation Acc		Translation Acc		Validation Acc		Translation Acc	
Aug %	CNN	FCN	CNN	FCN	CNN	FCN	CNN	FCN
0.00	0.99054	**0.98807**	**0.36654**	0.37229	**0.67108**	**0.64733**	**0.44456**	**0.43347**
0.10	**0.99100**	0.98786	0.36263	**0.37666**	0.64475	0.62633	0.42742	0.42179
0.20	0.98921	0.98714	0.35503	0.37587	0.63567	0.62375	0.42288	0.42011
0.30	0.98943	0.98714	0.35113	0.36777	0.62992	0.62350	0.41944	0.41986
0.40	0.98950	0.98750	0.34368	0.36053	0.61225	0.62342	0.40835	0.41952
0.50	0.98864	0.98664	0.34107	0.35236	0.60917	0.62167	0.40348	0.41734
0.60	0.98893	0.98750	0.33550	0.34968	0.59083	0.61617	0.39365	0.41541
0.70	0.98843	0.98707	0.33565	0.34650	0.59308	0.61558	0.39415	0.41473
0.80	0.98821	0.98671	0.33200	0.34165	0.58392	0.61375	0.38794	0.41322
0.90	0.98800	0.98679	0.33377	0.33970	0.58058	0.60925	0.38458	0.41112
0.99	0.98829	0.98650	0.32986	0.33767	0.57383	0.60996	0.37912	0.41053

Table A.3: Edge noise results. Median accuracy of un-augmented validation and translation augmented validation set. The left most column denotes the amount variance of noise used during training. When performing translational augmentation on the validation set, 25% augmentation was used throughout the experiments. Bold values indicate the highest performing model in that accuracy metric for CNN and FCN, respectively.

	MNIST				CIFAR			
	Validation Acc		Translation Acc		Validation Acc		Translation Acc	
Aug %	CNN	FCN	CNN	FCN	CNN	FCN	CNN	FCN
0.00	**0.99054**	0.98807	**0.36654**	0.37229	**0.67108**	**0.64733**	**0.44456**	**0.43347**
0.10	0.99000	0.98750	0.35757	0.37004	0.63092	0.63425	0.41507	0.41919
0.20	0.98957	0.98779	0.35880	**0.37410**	0.59258	0.61825	0.39037	0.40381
0.30	0.99007	0.98829	0.36328	0.36957	0.56258	0.59404	0.37433	0.38899
0.40	0.98979	**0.98879**	0.35793	0.36769	0.53521	0.56983	0.35652	0.37576
0.50	0.98964	0.98857	0.35084	0.34990	0.51142	0.55075	0.34232	0.36794
0.60	0.98900	0.98836	0.33623	0.33261	0.48083	0.53192	0.32451	0.35685
0.70	0.98807	0.98750	0.31854	0.31782	0.44858	0.50850	0.30612	0.34341
0.80	0.98614	0.98586	0.30433	0.30136	0.42417	0.47683	0.29255	0.32502
0.90	0.98407	0.98350	0.29080	0.28516	0.41179	0.44933	0.28642	0.30855
0.99	0.98164	0.98086	0.27590	0.27235	0.38250	0.40042	0.27050	0.28154

Table A.4: Noise results. Median accuracy of un-augmented validation and translation augmented validation set. The left most column denotes the amount variance of noise used during training. When performing translational augmentation on the validation set, 25% augmentation was used throughout the experiments. Bold values indicate the highest performing model in that accuracy metric for CNN and FCN, respectively.

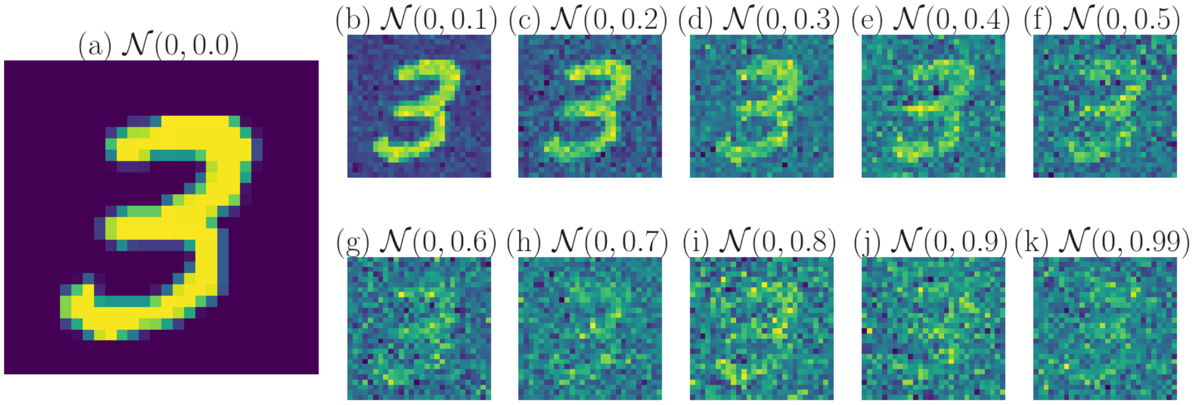

(a) $\mathcal{N}(0, 0.0)$
(b) $\mathcal{N}(0, 0.1)$
(c) $\mathcal{N}(0, 0.2)$
(d) $\mathcal{N}(0, 0.3)$
(e) $\mathcal{N}(0, 0.4)$
(f) $\mathcal{N}(0, 0.5)$
(g) $\mathcal{N}(0, 0.6)$
(h) $\mathcal{N}(0, 0.7)$
(i) $\mathcal{N}(0, 0.8)$
(j) $\mathcal{N}(0, 0.9)$
(k) $\mathcal{N}(0, 0.99)$

Figure A.2: Examples of noise augmentation on MNIST. (a) 0 variance noise, equivalent to a un-altered MNIST image. (b-k) Gradually increasing the variance of noise augmentation by .1 each time.

A.2.3 Quadrant Swap

Using the convention from Cartesian geometry, quadrant I and III of images are swapped. This type of image deformation serves to test the networks reliance on features as opposed to global structure. The hypothesis being that because of CNNs shared weight paradigm, the location of features matters less opposed to the presence of the feature itself. Conversely, FCN filters can only operate locally within their receptive field, rendering the global structure of the image essential.

To test this hypothesis, features of images need to be manipulated in such a way to compromise the overall structure of the image but preserve individual features. Figure B.3 shows a visual example of the swap procedure.

The results from this task, shown in Figure 2.6, indicate CNNs score a higher accuracy on Quadrant swapped images compared to FCNs. This indicates that CNNs are still able to recognize features of the altered images when making their prediction. This bolsters the notion that the overall structure of an image is less important for a CNN as opposed to the

FCN, that performs nearly 20% worse. In this task, a lower score indicates higher importance on the global structure.

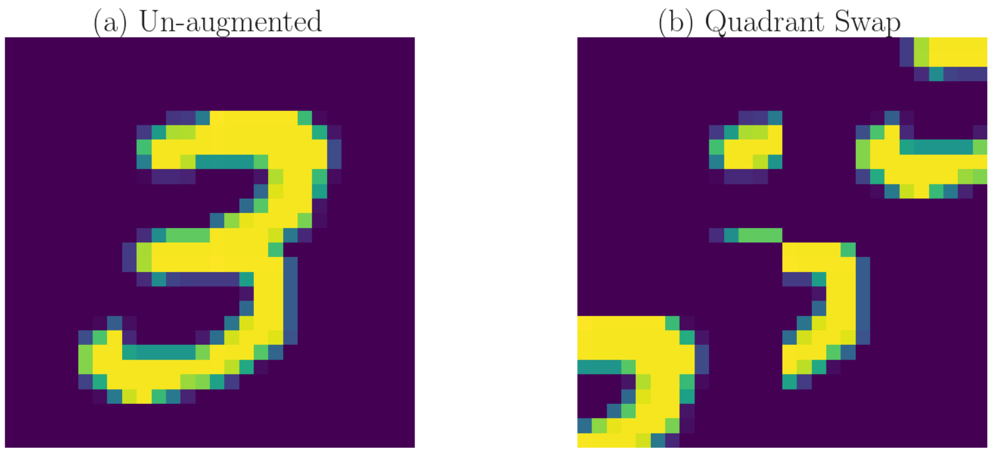


Figure A.3: Quadrant swap on MNIST. (a) Un-augmented image. (b) Quadrant swap augmentation. Where quadrant I is replaced with quadrant III and III with I

A.2.4 Rotation

Rotation augmentation is accomplished via rotating images clockwise about the center point. Figure B.4 shows the effect of rotating an MNIST image from 0%, Figure B.4a, up through 99%, Figure B.4k. Table B.3 displays the results of training networks with rotation augmentation.

A.2.5 Approximate Weight-Sharing

The distance between weight kernels within an FCN layer was measured during training with other types of augmentation. This experiment tests if translation augmentation is the cause of approximate weight-sharing in FCNs. Figure B.6 and B.7 show the results of training with rotation and noise augmentation, respectively.

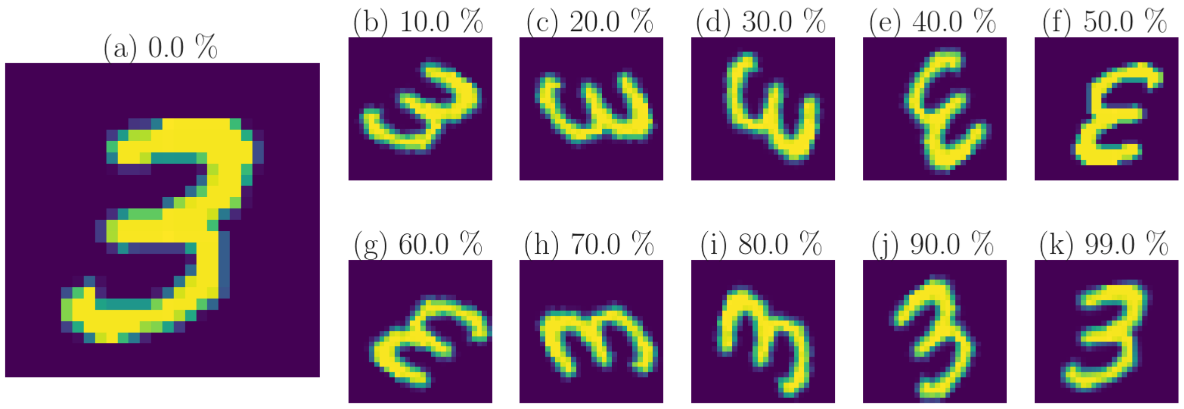


Figure A.4: Examples of rotation augmentation on MNIST. (a) 0% rotation, equivalent to a un-altered MNIST image. (b-k) Gradually increasing the degree rotation by 10% each time.

The results show that for all augmentation settings of rotation, the distance between filters increases over time. Additionally, in noise augmented training, the average Euclidean distance increases in all cases except very high values of noise (0.8, 0.9, 0.99). Indicating this increase in filter similarity is due to the high noise levels across the image. This is confirmed visually by examining Figure B.2i, B.2j, and B.2k. Also, the decrease in euclidean distance for noise is not as dramatic as observed for translation, Figure 2.5.

A.3 Variable Connection Patterns

Also implemented in this study are neurons with variable connection patterns in FCNs. At the start of training, a chosen percentage of weights are randomly set to 0, representing the absence of a dendritic connection. These missing weights do not contribute to the output of the layer, and their values are not updated during backpropagation. The resulting connection patterns are maintained throughout training and testing. There are multiple options for implementing neurons with variable connection patterns. For computational simplicity, the implementation used in this paper is to turn off connections within each

	MNIST				CIFAR			
	Validation Acc		Translation Acc		Validation Acc		Translation Acc	
Aug %	CNN	FCN	CNN	FCN	CNN	FCN	CNN	FCN
0.00	0.99054	0.98807	0.36654	0.37229	0.67108	0.64733	0.44456	0.43347
0.10	**0.99079**	**0.98836**	**0.38527**	**0.37753**	**0.68233**	**0.65500**	**0.48660**	**0.46211**
0.20	0.98907	0.98571	0.36372	0.35055	0.65746	0.62608	0.47450	0.44766
0.30	0.98779	0.98436	0.33587	0.30107	0.64708	0.61000	0.47106	0.43364
0.40	0.98636	0.98329	0.32147	0.28877	0.63625	0.59767	0.46455	0.42981
0.50	0.98439	0.98114	0.32183	0.29492	0.62242	0.58550	0.46337	0.42465
0.60	0.98400	0.97979	0.32154	0.29854	0.61533	0.57458	0.45682	0.41919
0.70	0.98336	0.97864	0.32060	0.29337	0.60717	0.57100	0.44750	0.41465
0.80	0.98236	0.97786	0.31547	0.29015	0.60875	0.56567	0.44758	0.40961
0.90	0.98071	0.97686	0.31040	0.28566	0.59958	0.55767	0.44414	0.40692
0.99	0.98350	0.97975	0.32429	0.29239	0.61842	0.58108	0.46102	0.42221

Table A.5: Rotation results. Median accuracies of un-augmented validation and translation augmented validation set. The left most column denotes the percentage of rotation used during training. When performing translational augmentation on the validation set, 25% augmentation was used throughout the experiments. Bold values indicate the highest performing model in that accuracy metric for CNN and FCN, respectively.

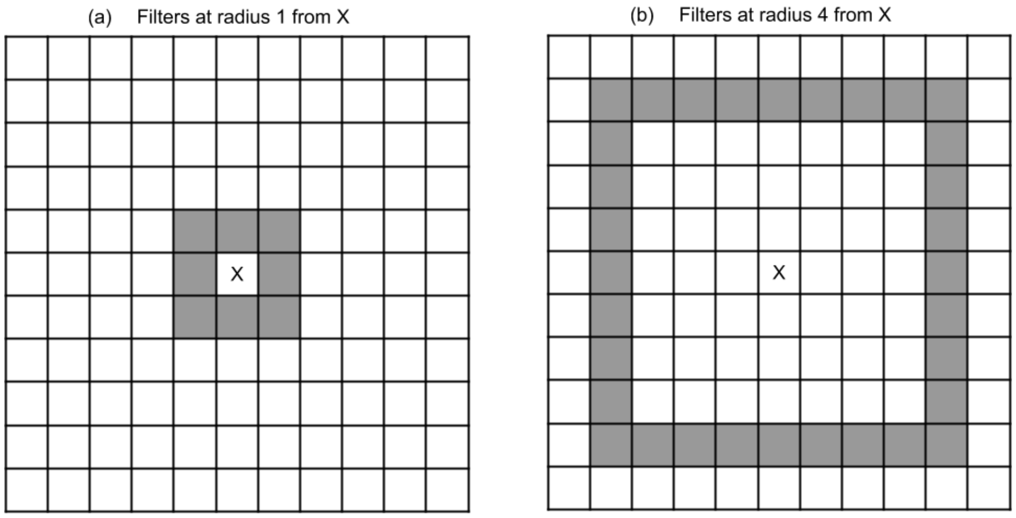

(a) Filters at radius 1 from X
X
(b) Filters at radius 4 from X
X

Figure A.5: Filters at a specified radius. (a) Filters at radius 1 from the desired filter, marked by an X. (b) Filters at radius 4 from the desired filter, marked by an X.

square filter given some probability. In simulations, we vary the probability from 0 to 99% by increments of 10%. The results from these simulations are reported in Figure C.1 and C.2 for MNIST and CIFAR, respectively. Both datasets use 30% translational augmentation

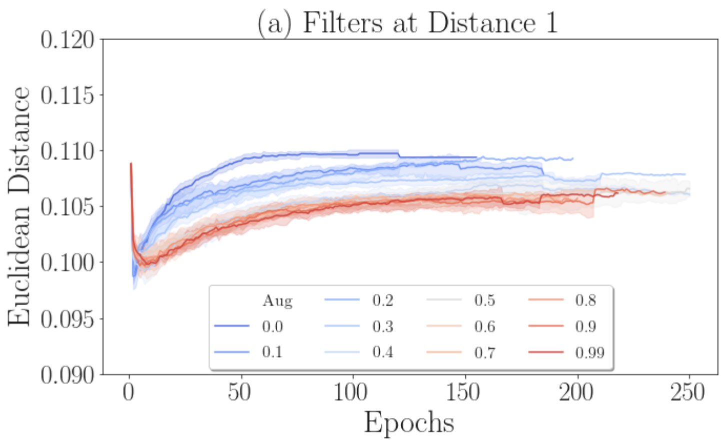


Figure A.6: Euclidean distance between filters of an FCN layer, trained on MNIST with rotation augmentation. (a) Average euclidean distance between filters one unit away. i.e. all adjacent filters in the layer. (b) Average euclidean distance between filters four units away.

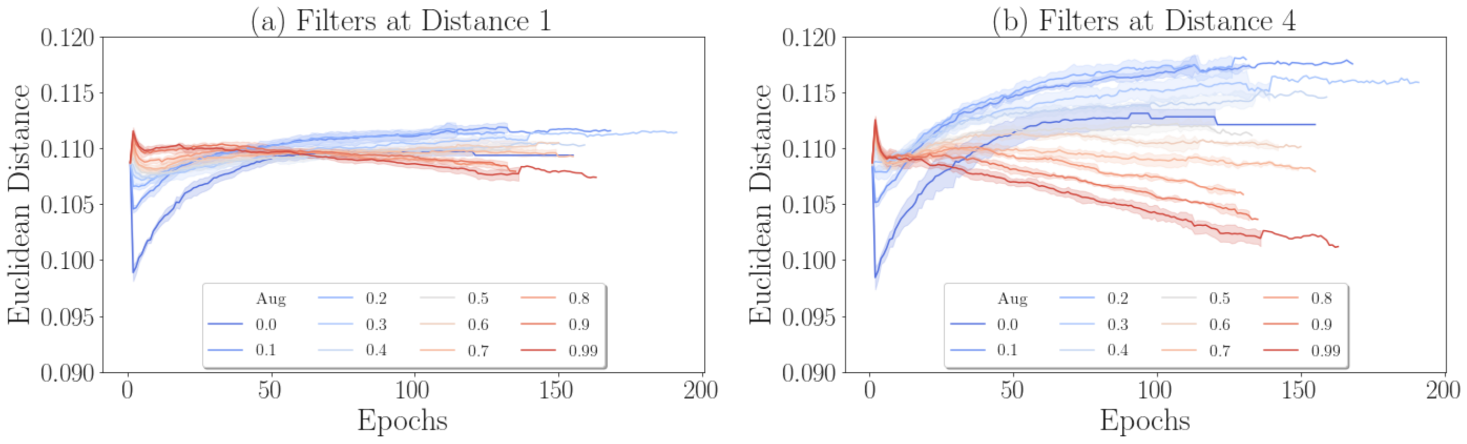


Figure A.7: Euclidean distance between filters of an FCN layer, trained on MNIST with noise augmentation. (a) Average euclidean distance between filters one unit away. i.e. all adjacent filters in the layer. (b) Average euclidean distance between filters four units away.

for these experiments.

The variable connection probability is varied from 0% to 99% as indicated by the legend (VCP %). The results shown in Figure C.1 and C.2 were trained using 20% translation augmentation. Accuracy on the validation set indicates FCNs are robust to even large amounts of missing connections. Even when 80% of connections are absent, the FCN is able to perform comparably well on both MNIST and CIFAR. FCNs are shown to be robust to a high degree of missing connections. Performance degrades rapidly beyond 90%.

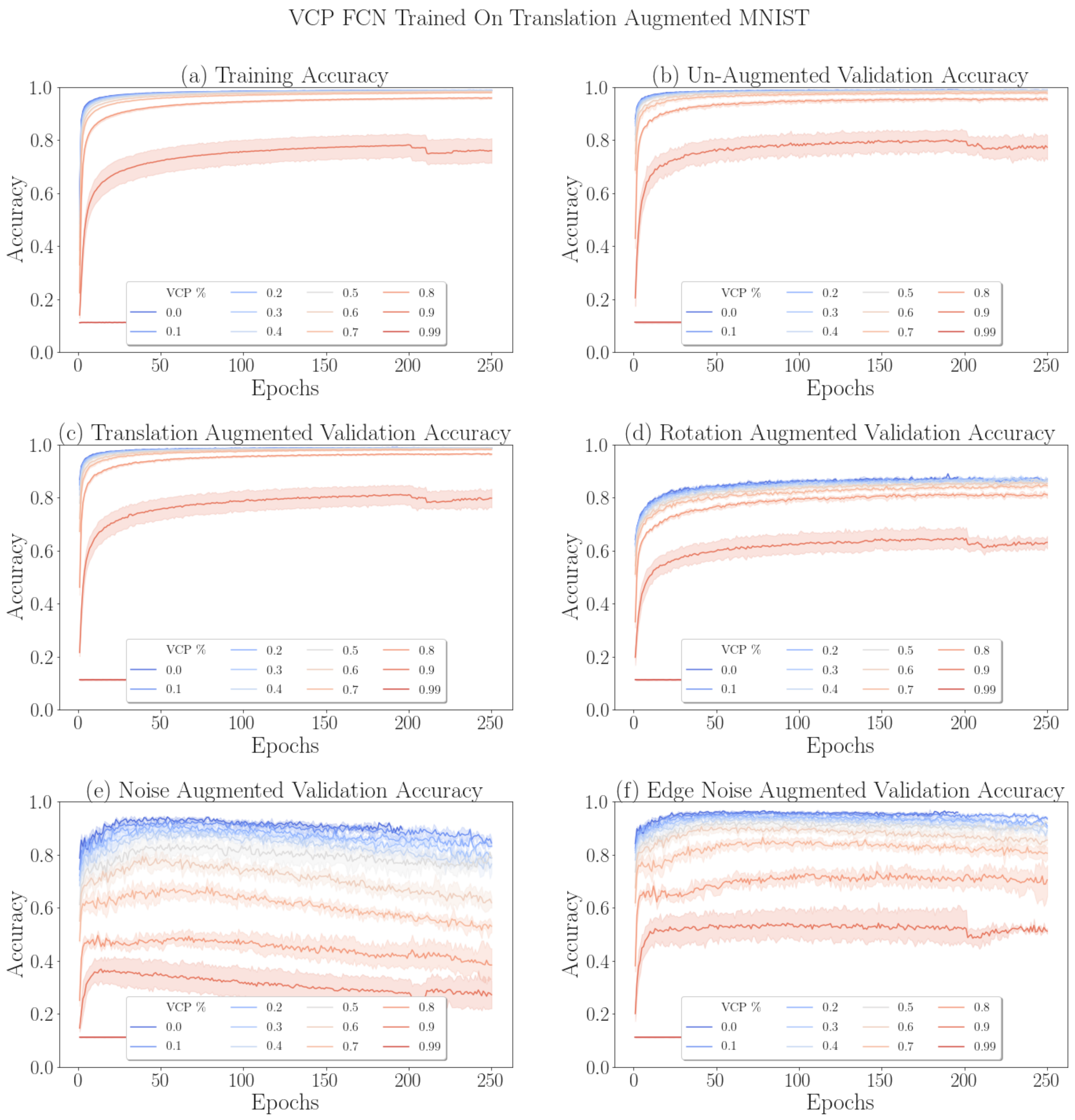


Figure A.8: FCNs trained with variable connection patterns on MNIST. The legends indicate the probability of absent dentritic connections in a FCN filter. (a) Accuracy on translation augmented training set, 30% translations. (b) Accuracy on the un-augmented validation set. (c) Accuracy on the translation augmented validation set, using 25% translations. (d) Accuracy on the rotation augmented validation set. (e) Accuracy on the noise augmented validation set. (f) Accuracy on the edge noise augmented validation set.

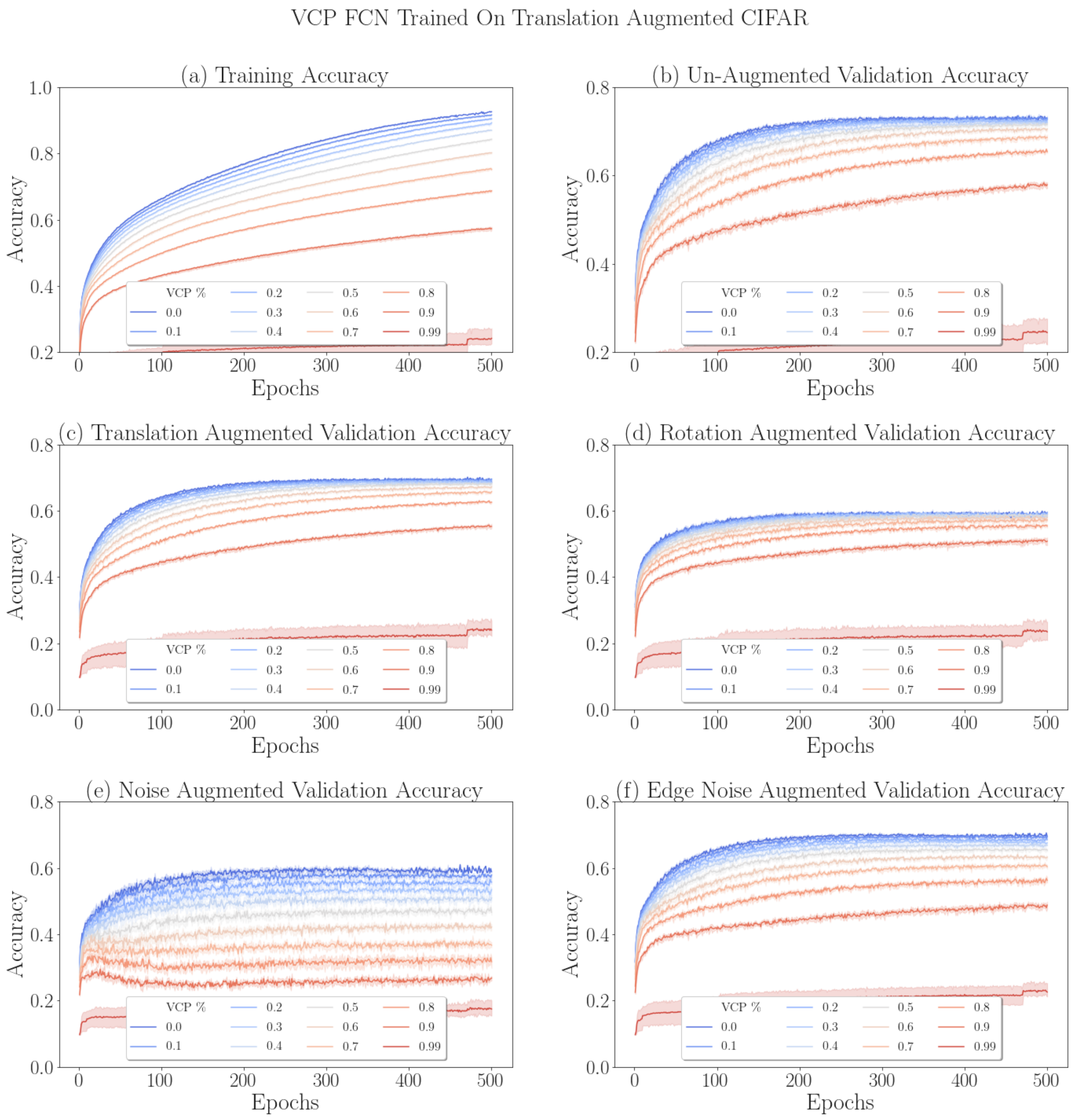


Figure A.9: FCNs trained with variable connection patterns on CIFAR. The legends indicate the probability of absent dentritic connections in a FCN filter. (a) Accuracy on translation augmented training set, 30% translations. (b) Accuracy on the un-augmented validation set. (c) Accuracy on the translation augmented validation set, using 25% translations. (d) Accuracy on the rotation augmented validation set. (e) Accuracy on the noise augmented validation set. (f) Accuracy on the edge noise augmented validation set.

Bibliography

[1] Fortran, Mar. 2011.

[2] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[3] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th Symposium on Operating Systems Design and Implementation 2016)*, pages 265–283, 2016.

[4] R. Abiri, S. Borhani, E. W. Sellers, Y. Jiang, and X. Zhao. A comprehensive review of eeg-based brain–computer interface paradigms, topical review, journal of neural engineering. *Journal of Neural Engineering*, 16(1):011001, 2019.

[5] F. Agostinelli, S. McAleer, A. Shmakov, and P. Baldi. Solving the rubik's cube with deep reinforcement learning and search. *Nature Machine Intelligence*, 1(8):356–363, 2019.

[6] M. ai Li, W. Zhu, H. na Liu, and J.-F. Yang. Adaptive feature extraction of motor imagery eeg with optimal wavelet packets and se-isomap. *Applied Sciences*, 7:390, 2017.

[7] H. U. Amin, A. Malik, R. F. Ahmad, N. Badruddin, N. Kamel, M. Hussain, and W.-T. Chooi. Feature extraction and classification for eeg signals using wavelet transform and machine learning techniques. *Australasian Physical & Engineering Sciences in Medicine*, 38, 02 2015.

[8] K. Ang, Z. Chin, H. Zhang, and C. Guan. Filter bank common spatial pattern (fbcsp) in brain-computer interface. pages 2390 – 2397, 07 2008.

[9] K. Ang, K. Chua, K. S. Phua, C. Wang, Z. Chin, C. Kuah, W. Low, and C. Guan. A randomized controlled trial of eeg-based motor imagery brain-computer interface

robotic rehabilitation for stroke. *Clinical EEG and neuroscience: official journal of the EEG and Clinical Neuroscience Society (ENCS)*, 46:310–320, 10 2015.

[10] K. K. Ang, Z. Y. Chin, C. C. Wang, C. Guan, and H. Zhang. Filter bank common spatial pattern algorithm on bci competition iv datasets 2a and 2b. *Frontiers in Neuroscience*, 6, 2012.

[11] F. Archambeau, N. Méchitoua, and M. Sakiz. Code Saturne: A Finite Volume Code for the computation of turbulent incompressible flows - Industrial Applications. *International Journal on Finite Volumes*, 1(1):http://www.latp.univ–mrs.fr/IJFV/spip.php?article3, Feb. 2004.

[12] A. Aurisano, A. Radovic, D. Rocco, A. Himmel, M. Messier, E. Niner, G. Pawloski, F. Psihas, A. Sousa, and P. Vahle. A convolutional neural network neutrino event classifier. *Journal of Instrumentation*, 11(09):P09001, 2016.

[13] O. Bai, V. Rathi, P. Lin, D. Huang, H. Battapady, D.-Y. Fei, L. Schneider, E. Houdayer, X. Chen, and M. Hallett. Prediction of human voluntary movement before it occurs. *Clinical neurophysiology : official journal of the International Federation of Clinical Neurophysiology*, 122:364–72, 02 2011.

[14] P. Baldi. *Deep Learning in Science*. Cambridge University Press, 2021.

[15] P. Baldi, K. Bauer, C. Eng, P. Sadowski, and D. Whiteson. Jet substructure classification in high-energy physics with deep neural networks. *Physical Review D*, 93(9):094034, 2016.

[16] P. Baldi and Y. Chauvin. Neural networks for fingerprint recognition. *Neural Computation*, 5(3):402–418, 1993.

[17] P. Baldi, Z. Lu, and P. Sadowski. Learning in the machine: the symmetries of the deep learning channel. *Neural Networks*, 95:110–133, 2017.

[18] P. Baldi, Z. Lu, and P. Sadowski. Learning in the machine: Random backpropagation and the deep learning channel. *Artificial Intelligence*, 260:1–35, 2018. Also: arXiv:1612.02734.

[19] P. Baldi and P. Sadowski. The dropout learning algorithm. *Artificial Intelligence*, 210C:78–122, 2014.

[20] P. Baldi and P. Sadowski. The dropout learning algorithm. *Artificial intelligence*, 210:78–122, 2014.

[21] P. Baldi and P. Sadowski. A theory of local learning, the learning channel, and the optimality of backpropagation. *Neural Networks*, 2016. To appear.

[22] P. Baldi and P. Sadowski. Learning in the machine: Recirculation is random backpropagation. *Neural Networks*, 108:479–494, 2018.

[23] P. Baldi, P. Sadowski, and Z. Lu. Learning in the machine: The symmetries of the deep learning channel. *Neural Networks*, 95:110–133, 2017.

[24] P. Baldi, P. Sadowski, and Z. Lu. Learning in the machine: Random backpropagation and the deep learning channel. *Artificial intelligence*, 260:1–35, 2018.

[25] Y. Bar-Sinai, S. Hoyer, J. Hickey, and M. P. Brenner. Learning data-driven discretizations for partial differential equations. *Proceedings of the National Academy of Sciences*, 116(31):15344–15349, 2019.

[26] S. Bartunov, A. Santoro, B. Richards, L. Marris, G. E. Hinton, and T. Lillicrap. Assessing the scalability of biologically-motivated deep learning algorithms and architectures. In *Advances in Neural Information Processing Systems*, pages 9368–9378, 2018.

[27] J. Bergstra, D. Yamins, and D. D. Cox. Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In *Proceedings of the 12th Python in science conference*, pages 13–20. Citeseer, 2013.

[28] J. Bernal. Neurbt: A program for computing neural networks for classification using batch learning, 02 2015.

[29] J. Bernal and J. Torres-Jimenez. Sagrad: A program for neural network training with simulated annealing and the conjugate gradient method. *Journal of research of the National Institute of Standards and Technology*, 120:113, 06 2015.

[30] R. F. Betzel, M. A. Bertolero, E. M. Gordon, C. Gratton, N. U. F. Dosenbach, and D. S. Bassett. The community structure of functional brain networks exhibits scale-specific patterns of inter- and intra-subject variability. *NeuroImage*, 202, 2019.

[31] T. Beucler, M. Pritchard, S. Rasp, P. Gentine, J. Ott, and P. Baldi. Enforcing analytic constraints in neural-networks emulating physical systems. *arXiv preprint arXiv:1909.00912*, 2020.

[32] T. Beucler, M. Pritchard, S. Rasp, J. Ott, P. Baldi, and P. Gentine. Enforcing analytic constraints in neural networks emulating physical systems. *Physical Review Letters*, 126(9):098302, 2021.

[33] B. Blankertz, G. Dornhege, M. Krauledat, K.-R. Müller, and G. Curio. The non-invasive berlin brain–computer interface: Fast acquisition of effective performance in untrained subjects. *NeuroImage*, 37:539–550, 2007.

[34] L. Börgesson. Abaqus. In *Developments in geotechnical engineering*, volume 79, pages 565–570. Elsevier, 1996.

[35] N. D. Brenowitz, T. Beucler, M. Pritchard, and C. S. Bretherton. Interpreting and stabilizing machine-learning parametrizations of convection, 2020.

[36] N. D. Brenowitz and C. S. Bretherton. Prognostic validation of a neural network unified physics parameterization. *Geophysical Research Letters*, 45(12):6289–6298, 2018.

[37] P. Brierley. Fortran90 mlp backprop code.

[38] J. Bromley, J. W. Bentz, L. Bottou, I. Guyon, Y. LeCun, C. Moore, E. Sackinger, and R. Shah. Signature verification using a siamese time delay neural network. *International Journal of Pattern Recognition and Artificial Intelligence*, 7(4), August 1993.

[39] B. Brooks, R. Bruccoleri, B. Olafson, D. States, S. Swaminathan, and M. Karplus. Charmm: A program for macromolecular energy, minimization, and dynamics calculations. *Journal of Computational Chemistry*, 4:187 – 217, 09 2004.

[40] C. Brunner, R. Leeb, G. Müller-Putz, A. Schlögl, and G. Pfurtscheller. Bci competition 2008–graz data set a. *Institute for Knowledge Discovery (Laboratory of Brain-Computer Interfaces), Graz University of Technology*, 16:1–6, 2008.

[41] C. F. Cadieu, H. Hong, D. L. K. Yamins, N. Pinto, D. Ardila, E. A. Solomon, N. J. Majaj, and J. J. DiCarlo. Deep neural networks rival the representation of primate it cortex for core visual object recognition. *PLOS Computational Biology*, 2014.

[42] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[43] F. Chollet. *Deep Learning mit Python und Keras: Das Praxis-Handbuch vom Entwickler der Keras-Bibliothek*. MITP-Verlags GmbH & Co. KG, 2018.

[44] F. Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015.

[45] D. Cireş, A. Giusti, L. M. Gambardella, and J. Schmidhuber. Deep neural networks segment neuronal membranes in electron microscopy images. In *Advances in neural information processing systems*, pages 2843–2851, 2012.

[46] G. Cisotto, A. Zanga, J. Chlebus, I. Zoppis, S. Manzoni, and U. Markowska-Kaczmar. Comparison of attention-based deep learning models for eeg classification. *ArXiv*, abs/2012.01074, 2020.

[47] J. Contreras-Vidal, A. Presacco, H. Agashe, and A. Paek. Restoration of whole body movement toward a noninvasive brain-machine interface system. *IEEE pulse*, 3:34–7, 01 2012.

[48] A. Craik, Y. He, and J. Contreras-Vidal. Deep learning for electroencephalogram (eeg) classification tasks: A review. *Journal of Neural Engineering*, 16, 02 2019.

[49] Y. L. Cun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel. Handwritten digit recognition with a back-propagation network. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, pages 396–404. Morgan Kaufmann, San Mateo, CA, 1990.

[50] M. Curcic. A parallel fortran framework for neural networks and deep learning. In *ACM SIGPLAN Fortran Forum*, volume 38, pages 4–21. ACM, 2019.

[51] G. Dai, J. Zhou, J. Huang, and N. Wang. Hs-cnn: A cnn with hybrid convolution scale for eeg motor imagery classification. *Journal of Neural Engineering*, 17, 09 2019.

[52] J. Daly and J. Wolpaw. Brain-computer interfaces in neurological rehabilitation. *Lancet neurology*, 7:1032–43, 11 2008.

[53] F. P. de Lange. Neural mechanisms of motor imagery. 2008.

[54] M. A. Donelan, M. Curcic, S. S. Chen, and A. K. Magnusson. Modeling waves and wind stress. *Journal of Geophysical Research: Oceans*, 117(C11), 2012.

[55] B. Edelman, B. Baxter, and B. He. Eeg source imaging enhances the decoding of complex right hand motor imagery tasks. *IEEE transactions on bio-medical engineering*, 63, 08 2015.

[56] B. J. Edelman, J. Meng, D. Suma, C. Zurn, E. Nagarajan, B. S. Baxter, C. C. Cline, and B. He. Noninvasive neuroimaging enhances continuous neural tracking for robotic device control. *Science Robotics*, 4, 2019.

[57] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115–118, 2017.

[58] F. Fahimi, Z. Zhang, B. Goh, T.-S. Lee, K. Ang, and C. Guan. Inter-subject transfer learning with end-to-end deep convolutional neural network for eeg-based bci. *Journal of Neural Engineering*, 16, 11 2018.

[59] J. Faller, J. Cummings, S. Saproo, and P. Sajda. Regulation of arousal via online neurofeedback improves human performance in a demanding sensory-motor task. *Proceedings of the National Academy of Sciences of the United States of America*, 116:6482 – 6490, 2019.

[60] A. Ferrari, P. Sala, A. Fasso, and J. Ranft. Fluka: a multi-particle transport code. *CERN Yellow report*, 2005-10, 01 2005.

[61] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.

[62] D. J. Gagne, C.-C. Chen, and A. Gettelman. Emulation of bin microphysical processes with machine learning. In *100th American Meteorological Society Annual Meeting*. AMS, 2020.

[63] D. J. Gagne, T. McCandless, B. Kosovic, A. DeCastro, R. Loft, S. E. Haupt, and B. Yang. Machine learning parameterization of the surface layer: Bridging the observation-modeling gap. *AGUFM*, 2019:IN44A–04, 2019.

[64] D. J. Gagne II, H. M. Christensen, A. C. Subramanian, and A. H. Monahan. Machine learning for stochastic parameterization: Generative adversarial networks in the lorenz '96 model. *Journal of Advances in Modeling Earth Systems*, 12(3):e2019MS001896, 2020. e2019MS001896 10.1029/2019MS001896.

[65] T. Gandhi, B. Panigrahi, and S. Anand. A comparative study of wavelet families for eeg signal classification. *Neurocomputing*, 74:3051–3057, 10 2011.

[66] P. Gaur, R. B. Pachori, H. Wang, and G. Prasad. An empirical mode decomposition based filtering method for classification of motor-imagery eeg signals for enhancing brain-computer interface. *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7, 2015.

[67] P. Gaur, R. B. Pachori, H. Wang, and G. Prasad. A multi-class eeg-based bci classification using multivariate empirical mode decomposition based filtering and riemannian geometry. *Expert Syst. Appl.*, 95:201–211, 2018.

[68] P. Gentine, M. Pritchard, S. Rasp, G. Reinaudi, and G. Yacalis. Could machine learning break the convection parameterization deadlock? *Geophysical Research Letters*, 45(11):5742–5751, 2018.

[69] J.-C. Golaz, P. M. Caldwell, L. P. Van Roekel, M. R. Petersen, Q. Tang, J. D. Wolfe, G. Abeshu, V. Anantharaj, X. S. Asay-Davis, D. C. Bader, et al. The doe e3sm coupled model version 1: Overview and evaluation at standard resolution. *Journal of Advances in Modeling Earth Systems*, 11(7):2089–2129, 2019.

[70] W. W. Grabowski. Coupling cloud processes with the large-scale dynamics using the cloud-resolving convection paramaterization (CRCP). *Journal of the Atmospheric Sciences*, 58(9):978–997, 2001.

[71] V. Gulshan, L. Peng, M. Coram, M. C. Stumpe, D. Wu, A. Narayanaswamy, S. Venugopalan, K. Widner, T. Madams, J. Cuadros, et al. Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs. *Jama*, 316(22):2402–2410, 2016.

[72] K. G. Hartmann, R. T. Schirrmeister, and T. Ball. Eeg-gan: Generative adversarial networks for electroencephalograhic (eeg) brain signals. *ArXiv*, abs/1806.01875, 2018.

[73] B. He, B. Baxter, B. Edelman, C. Cline, and W. Ye. Noninvasive brain-computer interfaces based on sensorimotor rhythms. *Proceedings of the IEEE*, 103:907–925, 06 2015.

[74] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[75] I. Held, H. Guo, A. Adcroft, J. Dunne, L. Horowitz, J. Krasting, E. Shevliakova, M. Winton, M. Zhao, M. Bushuk, et al. Structure and performance of gfdl's cm4. 0 climate model. *Journal of Advances in Modeling Earth Systems*, 11(11):3691–3727, 2019.

[76] L. Hertel, J. Collado, P. Sadowski, and P. Baldi. Sherpa: Hyperparameter optimization for machine learning models. 2018.

[77] L. Hertel, J. Collado, P. Sadowski, J. Ott, and P. Baldi. Sherpa: Robust hyperparameter optimization for machine learning. *SoftwareX*, 12:100591, 2020.

[78] L. Hertel, J. Collado, P. Sadowski, J. Ott, and P. Baldi. Sherpa: Robust hyperparameter optimization for machine learning. *Submitted to SoftwareX*, 2020.

[79] L. Hertel, J. Collado, P. Sadowski, J. Ott, and P. Baldi. Sherpa: Robust hyperparameter optimization for machine learning. *SoftwareX*, 12:100591, 2020.

[80] D. J. Herzfeld and R. Shadmehr. Motor variability is not noise, but grist for the learning mill. *Nature Neuroscience*, 17:149–150, 2014.

[81] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[82] C. Honey, J.-P. Thivierge, and O. Sporns. Can structure predict function in the human brain? computational models of the brain. *NeuroImage*, 52:766–76, 09 2010.

[83] D. H. Hubel and T. N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106, 1962.

[84] J. W. Hurrell, M. M. Holland, P. R. Gent, S. Ghan, J. E. Kay, P. J. Kushner, J.-F. Lamarque, W. G. Large, D. Lawrence, K. Lindsay, et al. The community earth system model: a framework for collaborative research. *Bulletin of the American Meteorological Society*, 94(9):1339–1360, 2013.

[85] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[86] Y.-N. Jan and L. Y. Jan. Branching out: mechanisms of dendritic arborization. *Nat Rev Neurosci*, 11(5):316–328, 05 2010.

[87] M. Jeannerod. Mental imagery in the motor context. *Neuropsychologia*, 33:1419–1432, 1995.

[88] G.-Q. Jiang, J. Xu, and J. Wei. A deep learning algorithm of neural network for the parameterization of typhoon-ocean feedback in typhoon forecast models. *Geophysical Research Letters*, 45(8):3706–3716, 2018.

[89] M. Jianjun, S. Zhang, A. Bekyo, J. Olsoe, B. Baxter, and B. He. Noninvasive electroencephalogram based control of a robotic arm for reach and grasp tasks. *Scientific Reports*, 6, 12 2016.

[90] V. Jurcak, D. Tsuzuki, and I. Dan. 10/20, 10/10, and 10/5 systems revisited: Their validity as relative head-surface-based positioning systems. *NeuroImage*, 34:1600–1611, 2007.

[91] Kaggle. 2018 kaggle ml & ds survey, 2018.

[92] Kaggle. State of data science and machine learning 2019, 2019.

[93] M. Kayala and P. Baldi. Reactionpredictor: Prediction of complex chemical reactions at the mechanistic level using machine learning. *Journal of Chemical Information and Modeling*, 52(10):2526–2540, 2012.

[94] J. Kevric and A. Subasi. Comparison of signal decomposition methods in classification of eeg signals for motor-imagery bci system. *Biomed. Signal Process. Control.*, 31:398–406, 2017.

[95] M. Khairoutdinov, C. DeMott, and D. Randall. Evaluation of the simulated interannual and subseasonal variability in an amip-style simulation using the csu multiscale modeling framework. *Journal of Climate*, 21(3):413–431, 2008.

[96] M. Khairoutdinov, D. Randall, and C. DeMott. Simulations of the atmospheric general circulation using a cloud-resolving model as a superparameterization of physical processes. *Journal of the Atmospheric Sciences*, 62(7):2136–2154, 2005.

[97] D. Komatitsch, J.-P. Vilotte, J. Tromp, J.-P. Ampuero, K. Bai, P. Basini, C. Blitz, E. Bozdag, E. Casarotti, J. Charles, M. Chen, P. Galvez, D. Goddeke, V. Hjorleifsdottir, J. Labarta, N. Le Goff, P. Le Loher, M. Lefebvre, Q. Liu, Y. Luo, A. Maggi, F. Magnoni, R. Martin, R. Matzen, D. McRitchie, M. Meschede, P. Messmer, D. Michea, S. Nadh Somala, T. Nissen-Meyer, D. Peter, M. Rietmann, E. de Andrade, B. Savage, B. Schuberth, A. Sieminski, L. Strand, C. Tape, Z. Xie, and H. Zhu. Specfem3d cartesian v2.0.2 [software], 2012.

[98] B. Koo, H.-G. Lee, Y. Nam, H. Kang, C. S. Koh, H.-C. Shin, and S. Choi. A hybrid nirs-eeg system for self-paced brain computer interface with online motor imagery. *Journal of neuroscience methods*, 244, 05 2014.

[99] A. Krizhevsky, V. Nair, and G. Hinton. Cifar-10 (canadian institute for advanced research).

[100] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[101] N. LaHaye, J. Ott, M. J. Garay, H. M. El-Askary, and E. Linstead. Multi-modal object tracking and image fusion with unsupervised deep learning. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 12(8):3056–3066, 2019.

[102] D. Lary, M. Müller, and H. Mussa. Using neural networks to describe tracer correlations. *Atmospheric Chemistry and Physics*, 4(1):143–146, 2004.

[103] E. Lashgari, D. Liang, and U. Maoz. Data augmentation for deep-learning-based electroencephalography. *Journal of Neuroscience Methods*, 346, 2020.

[104] E. Lashgari and U. Maoz. Electromyography classification during reach-to-grasp motion using manifold learning, 07 2020.

[105] E. Lashgari, J. Ott, A. Connelly, P. Baldi, and U. Maoz. An end-to-end cnn with attentional mechanism applied to raw eeg in a bci classification task. *Journal of Neural Engineering*, 18(4):0460e3, 2021.

[106] E. Lashgari, A. Pouya, and U. Maoz. Decoding object weight from electromyography during human grasping, 03 2021.

[107] V. Lawhern, A. Solon, N. Waytowich, S. Gordon, C. Hung, and B. Lance. Eegnet: A compact convolutional network for eeg-based brain-computer interfaces. *Journal of Neural Engineering*, 15, 11 2016.

[108] Y. LeCun and C. Cortes. MNIST handwritten digit database. 2010.

[109] D.-H. Lee, S. Zhang, A. Fischer, and Y. Bengio. Difference target propagation. In *Joint european conference on machine learning and knowledge discovery in databases*, pages 498–515. Springer, 2015.

[110] D.-Y. Lee, J.-H. Jeong, K.-H. Shim, and S.-W. Lee. Decoding movement imagination and execution from eeg signals using bci-transfer learning method based on relation network. pages 1354–1358, 05 2020.

[111] R. Leeb, C. Brunner, G. Müller-Putz, A. Schlögl, and G. Pfurtscheller. Bci competition 2008–graz data set b. *Graz University of Technology, Austria*, pages 1–6, 2008.

[112] Y. Li, X. Zhang, B. Zhang, M.-Y. Lei, W.-G. Cui, and Y.-Z. Guo. A channel-projection mixed-scale convolutional neural network for motor imagery eeg decoding. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 27:1170–1180, 2019.

[113] T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature communications*, 7:13276, 2016.

[114] J. Ling, A. Kurzawski, and J. Templeton. Reynolds averaged turbulence modelling using deep neural networks with embedded invariance. *Journal of Fluid Mechanics*, 807:155–166, 2016.

[115] J. Liu, J. Ott, J. Collado, B. Jargowsky, W. Wu, J. Bian, and P. Baldi. Deep-learning-based kinematic reconstruction for dune. *arXiv preprint arXiv:2012.06181*, 2020.

[116] F. Lotte and C. Guan. Regularizing common spatial patterns to improve bci designs: Unified theory and new algorithms. *IEEE Transactions on Biomedical Engineering*, 58:355–362, 2011.

[117] M. Lotze, H. Flor, W. Grodd, W. Larbig, and N. Birbaumer. Phantom movements and pain. an fmri study in upper limb amputees. *Brain : a journal of neurology*, 124 Pt 11:2268–77, 2001.

[118] N. Lu, T. Li, X. Ren, and H. Miao. A deep learning scheme for motor imagery classification based on restricted boltzmann machines. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 25:1–1, 08 2016.

[119] S. Luck. *An Introduction to the Event-Related Potential Technique.* A Bradford Book. MIT Press, 2014.

[120] L. Luis and J. Gomez-Gil. Brain computer interfaces, a review. *Sensors (Basel, Switzerland)*, 12:1211–79, 12 2012.

[121] J. Luo, Z. Feng, J. Zhang, and N. Lu. Dynamic frequency feature selection based approach for classification of motor imageries. *Computers in biology and medicine*, 75:45–53, 2016.

[122] S. Machado, L. Almada, and R. N. Annavarapu. Progress and prospects in eeg-based brain-computer interface: Clinical applications in neurorehabilitation. *Journal of Rehabilitation Robotics*, 2013:28–41, 01 2013.

[123] E. Madenci and I. Guven. *The finite element method and applications in engineering using ANSYS®.* Springer, 2015.

[124] I. Majidov and T. K. Whangbo. Efficient classification of motor imagery electroencephalography signals using deep learning methods. *Sensors (Basel, Switzerland)*, 19, 2019.

[125] D. McFarland, W. Sarnacki, and J. Wolpaw. Electroencephalographic (eeg) control of three-dimensional movement. *Journal of neural engineering*, 7:036007, 06 2010.

[126] D. Mcfarland and J. Wolpaw. Brain-computer interfaces for communication and control. *Communications of the ACM*, 54:60–66, 05 2011.

[127] M. Mirza and S. Osindero. Conditional generative adversarial nets. *ArXiv*, abs/1411.1784, 2014.

[128] G. Mooers, M. Pritchard, T. Beucler, J. Ott, G. Yacalis, P. Baldi, and P. Gentine. Assessing the potential of deep learning for emulating cloud superparameterization in climate models with real-geography boundary conditions. *Journal of Advances in Modeling Earth Systems*, 13(5):e2020MS002385, 2021.

[129] V. Morash, O. Bai, S. Furlani, P. Lin, and M. Hallett. Classifying eeg signals preceding right hand, left hand, tongue, and right foot movements and motor imageries. *Clinical neurophysiology : official journal of the International Federation of Clinical Neurophysiology*, 119:2570–8, 11 2008.

[130] K. Mrini, F. Dernoncourt, T. Bui, W. Chang, and N. Nakashole. Rethinking self-attention: An interpretable self-attentive encoder-decoder parser. *ArXiv*, abs/1911.03875, 2019.

[131] S. Mueller, D. Wang, M. D. Fox, B. T. T. Yeo, J. Sepulcre, M. R. Sabuncu, R. Shafee, J. Lu, and H. Liu. Individual variability in functional connectivity architecture of the human brain. *Neuron*, 77:586–595, 2013.

[132] T. Mulder. Motor imagery and action observation: Cognitive tools for rehabilitation. *Journal of neural transmission (Vienna, Austria : 1996)*, 114:1265–78, 02 2007.

[133] Y. D. Murray et al. Users manual for ls-dyna concrete material model 159. Technical report, United States. Federal Highway Administration. Office of Research ..., 2007.

[134] I. K. Niazi, N. Jiang, M. Jochumsen, J. F. Nielsen, K. Dremstrup, and D. Farina. Detection of movement-related cortical potentials based on subject-independent training. *Medical & Biological Engineering & Computing*, 51:507 – 512, 2012.

[135] S. Nissen. Implementation of a fast artificial neural network library (fann). 12 2003.

[136] D. J. Ostry and P. L. Gribble. Sensory plasticity in human motor learning. *Trends in Neurosciences*, 39:114–123, 2016.

[137] J. Ott. Questions to guide the future of artificial intelligence research. *arXiv preprint arXiv:1912.10305*, 2019.

[138] J. Ott, A. Atchison, P. Harnack, A. Bergh, and E. Linstead. A deep learning approach to identifying source code in images and video. In *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*, pages 376–386. IEEE, 2018.

[139] J. Ott, A. Atchison, P. Harnack, N. Best, H. Anderson, C. Firmani, and E. Linstead. Learning lexical features of programming languages from imagery using convolutional neural networks. In *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*, pages 336–3363. IEEE, 2018.

[140] J. Ott, A. Atchison, and E. J. Linstead. Exploring the applicability of low-shot learning in mining software repositories. *Journal of Big Data*, 6(1):1–10, 2019.

[141] J. Ott, D. Bruyette, C. Arbuckle, D. Balsz, S. Hecht, L. Shubitz, and P. Baldi. Detecting pulmonary coccidioidomycosis with deep convolutional neural networks. *Machine Learning with Applications*, 5:100040, 2021.

[142] J. Ott, E. Linstead, N. LaHaye, and P. Baldi. Learning in the machine: To share or not to share? *Neural Networks*, 126:235–249, 2020.

[143] J. Ott, M. Pritchard, N. Best, E. Linstead, M. Curcic, and P. Baldi. A fortran-keras deep learning bridge for scientific computing. *Scientific Programming*, 2020, 2020.

[144] M. Parvan, A. R. Ghiasi, T. Y. Rezaii, and A. Farzamnia. Transfer learning based motor imagery classification using convolutional neural networks. *2019 27th Iranian Conference on Electrical Engineering (ICEE)*, pages 1825–1828, 2019.

[145] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. 2017.

[146] J. W. L. Paul F. Fischer and S. G. Kerkemeier. nek5000 Web page, 2008. http://nek5000.mcs.anl.gov.

[147] G. Pfurtscheller and F. Lopes da Silva. Event-related eeg-meg synchronization and desynchronization: Basic principles. *Clin. Neurophysiol.*, 72:250–258, 01 1999.

[148] G. Pfurtscheller and C. Neuper. Neuper, c.: Motor imagery activates primary sensorimotor area in humans. neurosci. lett. 239, 65-68. *Neuroscience letters*, 239:65–8, 01 1998.

[149] J. G. Powers, J. B. Klemp, W. C. Skamarock, C. A. Davis, J. Dudhia, D. O. Gill, J. L. Coen, D. J. Gochis, R. Ahmadov, S. E. Peckham, G. A. Grell, J. Michalakes, S. Trahan, S. G. Benjamin, C. R. Alexander, G. J. Dimego, W. Wang, C. S. Schwartz, G. S. Romine, Z. Liu, C. Snyder, F. Chen, M. J. Barlage, W. Yu, M. G., and Duda. The weather research and forecasting model: Overview, system efforts, and future directions. *Bulletin of the American Meteorological Society*, 98(8):1717–1737, 2017.

[150] M. S. Pritchard, C. S. Bretherton, and C. A. DeMott. Restricting 32–128 km horizontal scales hardly affects the mjo in the superparameterized community atmosphere model v. 3.0 but the number of cloud-resolving grid columns constrains vertical mixing. *Journal of Advances in Modeling Earth Systems*, 6(3):723–739, 2014.

[151] A. J. Quinn, D. Vidaurre, R. G. Abeysuriya, R. Becker, A. C. Nobre, and M. W. Woolrich. Task-evoked dynamic network analysis through hidden markov modeling. *Frontiers in Neuroscience*, 12, 2018.

[152] A. Radford and K. Narasimhan. Improving language understanding by generative pre-training. 2018.

[153] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[154] S. Rasp. Online learning as a way to tackle instabilities and biases in neural network parameterizations. *arXiv preprint arXiv:1907.01351*, 2019.

[155] S. Rasp, M. S. Pritchard, and P. Gentine. Deep learning to represent subgrid processes in climate models. *Proceedings of the National Academy of Sciences*, 115(39):9684–9689, 2018.

[156] H. Raza, H. Cecotti, Y. Li, and G. Prasad. Adaptive learning with covariate shift-detection for motor imagery-based brain–computer interface. *Soft Computing*, 20:3085–3096, 2016.

[157] Y. Rezaeitabar and U. Halici. A novel deep learning approach for classification of eeg motor imagery signals. *Journal of Neural Engineering*, 14:016003, 02 2017.

[158] S. H. Rudy, S. L. Brunton, J. L. Proctor, and J. N. Kutz. Data-driven discovery of partial differential equations. *Science Advances*, 3(4):e1602614, 2017.

[159] C. Ruffino, C. Papaxanthis, and F. Lebon. Neural plasticity during motor learning with motor imagery practice: Review and perspectives. *Neuroscience*, 341:61–78, 2017.

[160] Z. Ruilong, Z. Qun, L. Dou, and Z. Xinyi. A novel hybrid deep learning scheme for four-class motor imagery classification. *Journal of neural engineering*, 2019.

[161] J. F. D. Saa and M. Çetin. A latent discriminative model-based approach for classification of imaginary motor tasks from eeg data. *Journal of neural engineering*, 9 2:026020, 2012.

[162] M. T. Sadiq, X. Yu, Z. Yuan, Z. Fan, A. ur Rehman, G. Li, and G. Xiao. Motor imagery eeg signals classification based on mode amplitude and frequency components using empirical wavelet transform. *IEEE Access*, 7:127678–127692, 2019.

[163] M. T. Sadiq, X. Yu, Z. Yuan, F. Zeming, A. ur Rehman, I. Ullah, G. Li, and G. Xiao. Motor imagery eeg signals decoding by multivariate empirical wavelet transform-based framework for robust brain–computer interfaces. *IEEE Access*, 7:171431–171451, 2019.

[164] P. Sadowski, B. Radics, Ananya, Y. Yamazaki, and P. Baldi. Efficient antihydrogen detection in antimatter physics by deep learning. *Journal of Physics Communications*, 1(2):025001, 2017.

[165] S. Saha and M. Baumert. Intra- and inter-subject variability in eeg-based sensorimotor brain computer interface: A review. *Frontiers in Computational Neuroscience*, 13, 2019.

[166] S. Saha and M. Baumert. Intra- and inter-subject variability in eeg-based sensorimotor brain computer interface: A review. *Frontiers in Computational Neuroscience*, 13, 12 2019.

[167] M. Salvaris and P. Haggard. Decoding intention at sensorimotor timescales. *PloS one*, 9:e85100, 02 2014.

[168] A. Samadi, T. P. Lillicrap, and D. B. Tweed. Deep learning with dynamic spiking neurons and fixed feedback weights. *Neural computation*, 29(3):578–602, 2017.

[169] R. T. Schirrmeister, J. T. Springenberg, L. D. J. Fiederer, M. Glasstetter, K. Eggensperger, M. Tangermann, F. Hutter, W. Burgard, and T. Ball. Deep learning with convolutional neural networks for eeg decoding and visualization. *Human Brain Mapping*, 38(11):5391–5420, 2017.

[170] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[171] L. Schneider, E. Houdayer, O. Bai, and M. Hallett. What we think before a voluntary movement. *Journal of cognitive neuroscience*, 25, 01 2013.

[172] M. Schultze-Kraft, D. Birman, M. Rusconi, C. Allefeld, K. Görgen, S. Dähne, B. Blankertz, and J.-D. Haynes. The point of no return in vetoing self-initiated movements. *proceedings of the national academy of sciences of the united states of america.*, 113, 12 2015.

[173] M. Seghier and C. Price. Interpreting and utilising intersubject variability in brain function. *Trends in Cognitive Sciences*, 22, 03 2018.

[174] S. Shahid and G. Prasad. Bispectrum-based feature extraction technique for devising a practical brain-computer interface. *Journal of neural engineering*, 8 2:025014, 2011.

[175] P. Shaw, J. Uszkoreit, and A. Vaswani. Self-attention with relative position representations. In *NAACL*, 2018.

[176] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[177] S. M. Smith, E. P. Duff, A. R. Groves, T. E. Nichols, S. Jbabdi, L. T. Westlye, C. K. Tamnes, A. Engvig, K. B. Walhovd, A. M. Fjell, H. Johansen-Berg, and G. Douaud. Structural variability in the human brain reflects fine-grained functional architecture at the population level. *The Journal of Neuroscience*, 39:6136 – 6149, 2019.

[178] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[179] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[180] R. K. Srivastava, K. Greff, and J. Schmidhuber. Training very deep networks. In *Advances in Neural Information Processing Systems*, pages 2368–2376, 2015.

[181] M. Stanfield, J. Ott, C. Gardner, N. F. Beier, D. M. Farinella, C. A. Mancuso, P. Baldi, and F. Dollar. Real-time reconstruction of high energy, ultrafast laser pulses using deep learning. *Scientific Reports*, 12(1):1–11, 2022.

[182] A. Suwannarat, S. Pan-ngum, and P. Israsena. Comparison of eeg measurement of upper limb movement in motor imagery training system. *BioMedical Engineering OnLine*, 17, 08 2018.

[183] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[184] W.-k. Tam, R. K.-Y. Tong, F. Meng, and X. Gao. A minimal set of electrodes for motor imagery bci to control an assistive device in chronic stroke subjects: A multi-session study. *IEEE transactions on neural systems and rehabilitation engineering : a*

publication of the IEEE Engineering in Medicine and Biology Society, 19:617–27, 12 2011.

[185] Z. Tayeb, J. Fedjaev, N. Ghaboosi, C. Richter, L. Everding, X. Qu, Y. Wu, G. Cheng, and J. Conradt. Validating deep neural networks for online decoding of motor imagery movements from eeg signals. *Sensors (Basel, Switzerland)*, 19, 2019.

[186] N. S. Team. *NEMO ocean engine.*

[187] K. Thayer-Calder and D. A. Randall. The role of convective moistening in the madden–julian oscillation. *Journal of the Atmospheric Sciences*, 66(11):3297–3312, 2009.

[188] J. Tompson, M. Stein, Y. Lecun, and K. Perlin. Real-time continuous pose recovery of human hands using convolutional networks. *ACM Transactions on Graphics (ToG)*, 33(5):169, 2014.

[189] S. Tortora, S. Ghidoni, C. Chisari, S. Micera, and F. Artoni. Deep learning-based bci for gait decoding from eeg with lstm recurrent neural network. *Journal of neural engineering*, 2020.

[190] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. Learning spatiotemporal features with 3d convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 4489–4497, 2015.

[191] G. Urban, P. Tripathi, T. Alkayali, M. Mittal, F. Jalali, W. Karnes, and P. Baldi. Deep Learning Achieves near Human-level Polyp Detection in Screening Colonoscopy. *Gastroenterology*, 2018. In press.

[192] G. Urban, P. Tripathi, T. Alkayali, M. Mittal, F. Jalali, W. Karnes, and P. Baldi. Deep Learning Achieves near Human-level Polyp Detection in Screening Colonoscopy. *Gastroenterology*, 155(4):1069–1078, 2018.

[193] M. Valiev, E. J. Bylaska, N. Govind, K. Kowalski, T. P. Straatsma, H. J. J. Van Dam, D. Wang, J. Nieplocha, E. Apra, T. L. Windus, and W. A. de Jong. Nwchem: a comprehensive and scalable open-source solution for large scale molecular simulations. *Computer Physics Communications*, 181(9):1477–1489, 2010.

[194] A. Vaswani, N. M. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *ArXiv*, abs/1706.03762, 2017.

[195] H. R. Vega-Carrillo, V. M. Hernández-Dávila, E. Manzanares-Acuña, G. A. Mercado, E. Gallego, A. Lorente, W. A. Perales-Muñoz, and J. A. Robles-Rodríguez. Artificial neural networks in neutron dosimetry. *Radiation Protection Dosimetry*, 118(3):251–259, 10 2005.

[196] A. Wallcraft, H. Hurlburt, E. Metzger, J. Cummings, E. Chassignet, and O. Smedstad. Global ocean prediction using hycom. pages 259–262, 07 2007.

[197] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus. Regularization of neural networks using dropconnect. In *International Conference on Machine Learning*, pages 1058–1066, 2013.

[198] J. Wang, H. Ding, F. Azamian, B. Zhou, C. Iribarren, S. Molloi, and P. Baldi. Detecting cardiovascular disease from mammograms with deep learning. *IEEE Transactions on Medical Imaging*, 36(5):1172–1181, 2017.

[199] J. Wang, Z. Fang, N. Lang, H. Yuan, M.-Y. Su, and P. Baldi. A multi-resolution approach for spinal metastasis detection using deep siamese neural networks. *Computers in Biology and Medicine*, 84:137–146, 2017.

[200] S. Wang, W. Liu, J. Wu, L. Cao, Q. Meng, and P. J. Kennedy. Training deep neural networks on imbalanced data sets. *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 4368–4374, 2016.

[201] F. R. Willett, D. T. Avansino, L. R. Hochberg, J. M. Henderson, and K. V. Shenoy. High-performance brain-to-text communication via handwriting. *Nature*, 593 7858:249–254, 2021.

[202] D. Wu, Y. Xu, and B.-L. Lu. Transfer learning for eeg-based brain-computer interfaces: A review of progress made since 2016. *IEEE Transactions on Cognitive and Developmental Systems*, PP:1–1, 07 2020.

[203] D. L. K. Yamins, H. Hong, C. F. Cadieu, E. A. Solomon, D. Seibert, and J. J. DiCarlo. Performance-optimized hierarchical models predict neural responses in higher visual cortex. *Proceedings of the National Academy of Sciences*, 111(23):8619–8624, 2014.

[204] B. Yang, C. Fan, C. Guan, X. Gu, and M. Zheng. A framework on optimization strategy for eeg motor imagery recognition. *2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 774–777, 2019.

[205] Y. Yang, S. Chevallier, J. Wiart, and I. Bloch. Time-frequency selection in two bipolar channels for improving the classification of motor imagery eeg. *Conference proceedings : ... Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Conference*, 2012:2744–7, 08 2012.

[206] H. Yuan and B. He. Brain-computer interfaces using sensorimotor rhythms: Current state and future perspectives. *IEEE transactions on bio-medical engineering*, 61:1425–35, 05 2014.

[207] A. Zabidi, W. Mansor, Y. K. Lee, and C. Fadzal. Short-time fourier transform analysis of eeg signal generated during imagined writing. pages 1–4, 09 2012.

[208] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning requires rethinking generalization. *ArXiv*, abs/1611.03530, 2017.

[209] G. Zhang, V. Davoodnia, A. Sepas-Moghaddam, Y. Zhang, and A. Etemad. Classification of hand movements from eeg using a deep attention-based lstm network. *IEEE Sensors Journal*, 20:3113–3122, 2020.

[210] H. Zhang, X. Zhao, Z. Wu, B. Sun, and T. Li. Motor imagery recognition with automatic eeg channel selection and deep learning. *Journal of Neural Engineering*, 18, 2020.

[211] K. Zhang, N. Robinson, S.-W. Lee, and C. Guan. Adaptive transfer learning for eeg motor imagery classification with deep convolutional neural network. *Neural Networks*, 136, 12 2020.

[212] Q. Zhang and Y. Liu. Improving brain computer interface performance by data augmentation with conditional deep convolutional generative adversarial networks. *ArXiv*, abs/1806.07108, 2018.

[213] X. Zhang, Z. Wang, D. Liu, and Q. Ling. Dada: Deep adversarial data augmentation for extremely low data regime classification. *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2807–2811, 2019.

[214] Z. Zhang, F. Duan, J. Solé-Casals, J. Dinarès-Ferran, A. Cichocki, Z. Yang, and Z. Sun. A novel deep learning approach with data augmentation to classify motor imagery signals. *IEEE Access*, 7:15945–15954, 2019.

[215] Q. Zheng, F. Zhu, and P.-A. Heng. Robust support matrix machine for single trial eeg classification. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 26:551–562, 2018.

[216] X. X. Zhu, D. Tuia, L. Mou, G.-S. Xia, L. Zhang, F. Xu, and F. Fraundorfer. Deep learning in remote sensing: A comprehensive review and list of resources. *IEEE Geoscience and Remote Sensing Magazine*, 5(4):8–36, 2017.

[217] C. Zich, S. Debener, C. Kranczioch, M. G. Bleichner, I. Gutberlet, and M. D. Vos. Real-time eeg feedback during simultaneous eeg–fmri identifies the cortical signature of motor imagery. *NeuroImage*, 114:438–447, 2015.

[218] D. Zipser and R. A. Andersen. A back-propagation programmed network that simulates response properties of a subset of posterior parietal neurons. *Nature*, 331(6158):679–684, 1988.