Improving the Accuracy and Robustness of CNNs Using a Deep CCA Neural Data Regularizer

by

Cassidy Pirlot

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

STATISTICAL MACHINE LEARNING

Department of Mathematical and Statistical Sciences
University of Alberta

Abstract

Convolution Neural Networks (CNNs) have rapidly evolved since their neuroscience beginnings. These models efficiently and accurately classify images by optimizing the model's hidden representations to these images through training. These representations have been shown to resemble neural data from the primate visual system as the accuracy of the model improves. Works have been produced to exploit these findings to examine if the more (mammalian) brain-like a model's hidden representations are, the more (mammalian) brain-like the model's performances will be. Further, performance from a model that is human-like would achieve high accuracy, high super-class accuracy, and robustness. We expand on this work by using a neural data (ND) regularizer that uses Deep Canonical Correlation Analysis (DCCA). The regularizer optimizes the resemblance between the CNN's hidden representations to an image and the representations found in the mammalian visual pathway to the same image. Compared to CNNs without the ND regularizer, the ND regularized CNN resulted in higher accuracy and super-class accuracy, as well as becoming more robust to adversarial examples. These outcomes provide evidence that pushing CNNs to become more brain-like is not only achievable, but will also result in a better performing model.

Keywords: Machine Learning, Neuroscience, Convolutional Neural Networks, Computer Vision

Preface

This thesis is an original work by Cassidy Pirlot. No part of this thesis has been previously published.

To my Flora. Even though she was a cat, she taught me more about life than any human could.

Acknowledgments

I would like to thank Dr. Alona Fyshe for taking a chance on me and supporting, guiding, and inspiring me in this incredible field.

Also, thank-you to Joel Zylberberg, Richard Gerum, Adam Kashlak, Tonie Bodley, and my lab-mates for continuously providing feedback, insights, and encouragement.

Next, I am so grateful to my family. I owe a lot of what I achieve to my parents and sisters. They have always believed in me and have loved me (truly) unconditionally.

Lastly, thank-you to those who motivated, comforted, or helped me. If you think this applies to you, it does.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

The human brain is capable of extraordinary feats. From rapid development in the early stages of life to the ability to understand galaxies far from our reach, the skills humans possess leave us wonderstruck. To translate these abilities into machine learning algorithms is an attractive task for many researchers. The brain's aptitude to process what it sees quickly and accurately has been reproduced in many machine learning models, including **Convolutional Neural Networks** (CNNs) [25].

These networks have been used in an array of applications such as image recognition and object detection [3]. Neural networks are being used increasingly more to assist people in their daily lives. This emphasizes the importance of having these algorithms make appropriate choices for their users. Therefore, there is a need to create models that will make more human-like decisions. In classification models, a few human-like goals are improving the accuracy, super-class accuracy, and robustness to adversarial examples.

In the context of object classification from images, these tasks can be defined as:

- **Accuracy** refers to the proportion of images a network classifies correctly. Humans have very high accuracy; they are able to identify the objects within images even if they have not seen that particular image before.

- **Super-Class Accuracy** measures how reasonable mistakes are. If the model labels an image as the incorrect class, we prefer the chosen class to be related to the image's true class. For example, an image of a shrew could be misclassified as a tractor or a mouse. Clearly, mistaking a shrew for a mouse is more human-like than mistaking a shrew for a tractor.

- **Robustness to Adversarial Examples** is a robustness task. A robustness task tests a model on its ability to perform classification on a dataset or images the model has not seen. Adversarial examples are images designed to fool the specific model of interest and are often unmistakable to humans. A popular example is putting a small black rectangle on the image [2]. Often, a human could still determine what object is in the image by the uncovered portion of the image. However, this technique often causes models to misclassify images.

Previous studies have shown the relatedness that neuroscience and CNNs have with each other. Recent works have tried to take advantage of these findings to make CNNs more brain-like. However, these studies lack either significant improvements or concentrate on improving brain-likeness in only one task. In this thesis, we look to overcome these weaknesses using a **Neural Data (ND) Regularizer**. This regularizer is a deep network called **Deep Canonical Correlation Analysis (DCCA)**. We attach DCCA to the CNN we want to improve through a shared node. We evaluate of our model on the three tasks defined above: Accuracy, Super-Class Accuracy, and Robustness to Adversarial Examples.

1.2 Objectives

In this thesis, our contributions can be summarized as the following:

- With our ND regularizer, a CNN will show improvement in the following tasks:

    1. Accuracy (Section 5.1)

2. Super-Class Accuracy (Section 5.2)

3. Robustness to Adversarial Examples (Section 5.3)

- Using our ND regularizer a CNN achieves higher accuracy and super-class accuracy than previous works that attempt to make a model more brain-like (Section 5.1.1, Section 5.2.1).

- We show our ND regularizer works best on unaltered neural data but also performs well on generated neural data (Section 5.1.2).

- We find different contribution magnitudes of the regularizer on the CNN's cost function yields different results (Chapter 5).

1.3 Outline

This thesis is structured into 4 main chapters:

Chapter 2 will take a brief look at work that has inspired ours. An overview of the important fundamentals and unique properties of CNNs will be presented. We will discuss neuroscience findings that influenced the conception and creation of CNNs. Research on CNNs' ability to predict neural representations will also be discussed. Then, a review of previous works that have attempted to achieve brain-like behaviour in CNNs will be done. Lastly, we will seek a deeper understanding of an RSM (representational similarity matrix) method used to achieve brain-likeness [9] that heavily influences ours.

In Chapter 3, we will introduce the tools needed to successfully use our regularizer. We first give an overview of Canonical Correlation Analysis (CCA) [14], a data reduction and covariance finding technique, before looking at its solution mathematically. After discussing the limitations of the previous method and of CCA, we introduce a deep version of CCA- DCCA [4]. Then, we examine DCCA's benefits as well as its technical fundamentals. Next, we introduce the CNN we improve upon, CORnetZ

[17], as well as the two datasets we use. The first dataset [16] will be classified in our model, and the second dataset [6] will be used in the ND regularizer. Lastly, we give the procedure for preprocessing the second dataset.

In Chapter 4, we detail how the ND regularizer and the CNN are algorithmically attached and work together. We then describe the tuning of the DCCA model separate from the CNN. Lastly we discuss the experimental setup, before describing the generated datasets we use as controls in our experiments.

Next, in Chapter 5, three experiments are examined, and the results are reported. The experiments are **Accuracy**, **Super-Class Accuracy**, and **Robustness to Adversarial Examples**. We explore the effect of the ND regularizer in these experiments with different contribution magnitudes of the DCCA loss and with the generated datasets. Figures, interpretations, and discussions are also provided for each experiment.

Finally, in Chapter 6 we concisely remind the reader of what was achieved and the contributions provided in this thesis. Lastly, limitations and future works are addressed.

Chapter 2

Previous Work

Here we review previous works that influence ours. First, we give an overview of what a CNN is and how it works. Then, we explore studies that give justification that constraining machine learning models, specifically classification models, to achieve brain-like representations is a logical objective. Next, we look at previous attempts to create brain-like models and discuss their successes and shortcomings. Then, we take a closer look at one study by Federer et al. [9] that influences and guides our work. Lastly, we explore limitations and how to potentially improve upon Federer et al.'s method.

2.1 Convolutional Neural Networks

Machine learning is a field of artificial intelligence that involves the use of algorithms. These algorithms use vast amounts of data and many iterations to learn patterns in order to make predictions on previously unseen data. This works by using a cost function, which is specific to the task the model is trying to solve, and an optimization algorithm. Inputs, X, are given to the model and are transformed to predict Y. Mathematically this looks like:

$$h(X) = Y$$

where h is the transforming function or the machine learning algorithm.

One machine learning model of interest is the Convolutional Neural Network (CNN).

CNNs are primarily used in a computer vision task called image recognition to detect and classify objects within images. CNNs are supervised models, which means that it makes use of a labelled dataset and the training task is to predict those labels. These labels are used in the cost function each epoch and the result determines the changes of weights in the different levels (layers) of the network for the next epoch. This change in weights is implemented with backpropagation. Backpropagation involves calculating the gradient or partial derivatives of each weight and bias in the network. This calculation enables every weight and bias to be individually updated to reduce the cost. The identifying layers of convolutional neural networks are convolutional and pooling layers.

A convolution is the application of a filter to an image input that helps it perform feature-extraction or learn features in images. Feature-extraction refers to a transformation of data that reduces it while maintaining the data's important features. The filters are two-dimensional arrays of weights and they are smaller than the input image. Therefore, a filter is applied to patches of the image until each pixel belonging to the image has encountered the filter at least once. The dot-product between the filter and the patch of pixels is calculated and a single value is returned as shown in Figure 2.1. When done repeatedly an array of values, called a feature map, is returned. Convolving allows a filter to detect and indicate the location and strength of a specific feature in the image. Usually many filters are used in a single convolutional layer so it learns many features at once.

A pooling layer follows a convolutional layer in a CNN. As the name suggests, a pooling layer applies a pooling function to its input (see Figure 2.2). A pooling function downsamples the feature map which creates invariance to translation. Invariance to translation means that a model can prevent small changes, such as rotation and cropping of the input image, from creating strikingly different feature maps. Some common pooling functions used are max-pooling, min-pooling, and average-pooling.

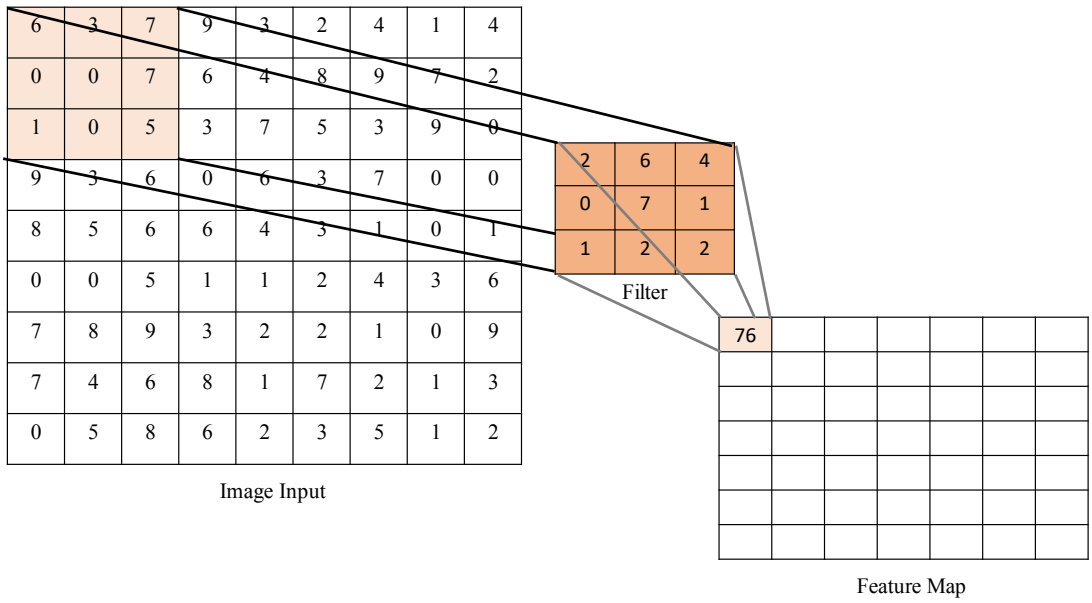


Figure 2.1: An illustration of a filter convolving over an input image. We see that the dot product is performed on the filter and a subset of the image's pixels. Once every pixel in the image is covered at least once by the filter the result is a feature map. Recall that usually a convolutional layer will have multiple filters and thus multiple feature maps will be returned.

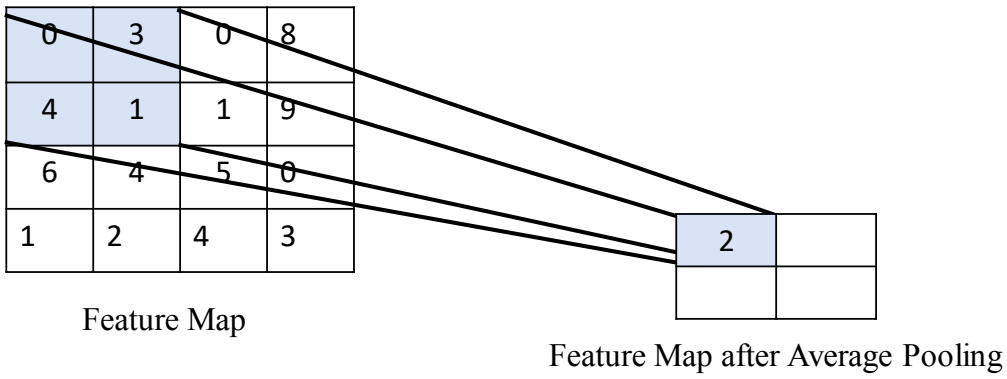


Figure 2.2: An example of average pooling on an input. Here the upper 2x2 block of the input is reduced to the average of its entries.

A simple convolutional model has a series of alternating convolutional and pooling layers. It is common that CNNs have a variety of other layers included in their architecture as well, but convolutional and pooling layers always appear in these models. The feature maps in the intermediate, or hidden layers of a CNN will be referred to as hidden representations in this work.

2.2 The Connection between Machine Learning and the Brain

The conception and advancement of machine learning models are often heavily attributed to neuroscience. The mechanisms of convolutional and pooling layers were shaped from a study done to further understand the physiological processes of the visual pathway. This particular neuroscience experiment is often credited for the origins of machine learning vision models. Performed in 1956 by two neuroscientists, Hubel and Weisel [13], this experiment included recording brain activity in the primary visual cortex of an anesthetized cat while its eyes responded to visual stimuli. They found that in brain areas that responded first to stimuli, certain cells would only respond to specific orientations and locations of lines. Further, every orientation and placement had a group of these cells responding to it. The researchers concluded that these cells, later named simple cells, had a preferred orientation and location of stimuli. Moreover, as the stimuli shifted further away from the preferred positioning, the simple cell would respond more weakly until it stopped responding entirely.

Interestingly, the researchers noticed a different phenomenon occur in later areas of the visual pathway. The cells in these later areas would respond best to lines that kept the same orientation but would traverse locations. These cells were later termed complex cells. This discovery helped the researchers conclude that the complex cells were not actually reacting to the stimuli itself but instead reacting to information supplied to them from a set of simple cells. These simple cells would later be the inspiration for convolutional layers in machine learning models, and the complex cells

would inspire pooling layers.

In 1979 a predecessor of modern CNNs, the Neocognitron model, was developed by Fukushima [10]. This model consisted of multiple feature-extraction layers followed by pooling layers which were directly inspired by Hubel and Weisel's work [13]. The Neocognitron was able to learn different patterns in Japanese hand written characters. Also, the model was robust to shifts in the character's position or distortions to the character's size. Further, the Neocognitron was able to learn these patterns on its own through repeated exposure.

However, the Neocognitron could be improved by making it a supervised model. LeCun et al. [18] devised a way to accomplish supervision with the LeNet model which is considered the first CNN. The LeNet allowed for backpropagation and for these feature-extraction layers to get updated depending on the model's performance. The feature-extraction layers from the Neocognitron were upgraded in the LeNet with convolutional layers through the introduction of filters.

Since the conception of the CNN family, these models have been further developed and studied to include more complex layers and mechanisms to address more complex image recognition tasks. Such progress has led to some CNNs having near-perfect performances in some tasks [20, 21, 23]. Not only have CNNs been directly inspired by neuroscience, but Yamins et al. [30] showed that the hidden representations of these models could also predict brain activation patterns. Yamins et al. developed CNNs that are biologically plausible and looked at the hidden representations from the model's different layers. Even though the researchers did not constrain the network with brain data, they found that the CNN's hidden representations were highly correlated to the neural activations in different areas of the mammalian visual pathway. Furthermore, the researchers found these results most significant when the CNN performed well in its recognition tasks. This high correlation leads to predictability of the brain's behaviour based solely on the network's attributes.

CNNs were designed to mimic the roles and mechanisms of simple and complex cells

identified in a neuroscience experiment. CNNs have continued to develop into their own separate field, adding more complexity to their architectures and tackling more challenging tasks, evolving past their neuroscience origins. Furthermore, these newer and more developed models are highly correlated to neural activations in the visual pathway, especially when they achieve high accuracy. This begs these questions:

- Can we push CNNs to become more brain-like?
- And what effect would that have on the performance of the model?

2.3 Constraining Networks to be More Brain-Like

Here we explore the possibility of constraining a network to have more brain-like representations. We also investigate what effect having brain-like representations would have on different metrics of the model's performance. Two approaches to achieve this have been used so far by machine learning researchers.

The first approach is to gather neural data from the visual pathway of subjects looking at images and then use that data to regularize the model. Two examples of this method are Federer et al. [9] and Safarani et al. [24]. This method uses the model's image recognition performance and similarity of the model's hidden representations to the neural data to adjust the model parameters through its loss function. This regularization can make the model's representations more brain-like without sacrificing its main-task capabilities. Safarni et al.'s [24] regularizing approach was to train a readout of the intermediate layer activations in the CNN to predict neural firing rates when a monkey views the same images as the model. Federer et al.'s approach was to optimize the match between the representational similarity matrices [15] of an intermediate CNN layer and the neural recordings.

The second approach used by Nassar et al. [22] was to identify spectral properties of neural representations. Namely, identifying the 1/n eigenspectrum of the covariance matrix of neural firing rates as formulated in Stringer et al. [28]. After

identifying the spectral properties, the CNN is regularized to match those spectral properties while being trained for object recognition [22]. This approach led to CNNs that demonstrated improved robustness to adversarial attacks compared to CNNs that were not regularized to match the spectral properties of neural representations. Reminder, robustness refers to a model's ability to perform its intended task on unfamiliar datasets. An adversarial attack is one where the dataset consists of images curated to deceive the model.

These two approaches both use neural data; in the first method, the data is directly used, and the second uses the data's found properties to constrain the model. Promising results emerged from these experiments. Compared to the unregularized models, the researchers found brain-regularized models:

- Improved robustness to image distortions was found in Safarani et al. [24].
- Modestly improved classification accuracy and super-class accuracy was found in Federer et al. [9].
- Improved robustness to label corruption during training was found in Federer et al. [9].
- Improved robustness to adversarial attacks was found in Nassar et al. [22].

Although these are optimistic results, there is room for improvement. For one, we would like to produce improved performance on datasets more complex and rich than the grey-scaled, 10-class MNIST dataset [7] used by Nassar. Second, Sarfani et al.'s and Nassar et al.'s experiments focused on improving only Classification Accuracy or Robustness to Adversarial Examples in the model instead of multiple aspects simultaneously. Finally, we aspire to find a single technique that would significantly improve performance in more than one task. That is, a technique that can be applied to a base model which will result in a more-than-modest improvement in classification accuracy and robustness.

2.4 An Examination of Federer et al.'s RSMs

Federer et al.'s [9] model uses a simple metric to determine how brain-like the model's hidden representations are. In our work, we want to find a metric that will more accurately express how similar the hidden representations of the model are to the neural data. However, before amending Federer et al.'s model, we first need to examine its setup, properties, and limitations.

Federer et al. used representational similarity matrices (RSMs) [15] to look at the similarities between the image representations in the brain data and the hidden representations in the CNN for the same image. These RSMs consist of the cosine similarities of image pair responses, which come from neural firing rates when subjects were shown the images. There is a vector v of neuron responses from the brain data for each image. For image i, j:

$$RSM_{i,j} = \frac{v_i \cdot v_j}{\|v_i\| \times \|v_j\|}$$

where $\cdot$ is the dot product, and $\|v\|$ is the length of the vector.

Similarly, an $\widehat{RSM}$ is computed from the hidden representations of one of the model's layers. In other words, the output of a layer in the model will have activations associated with each image; these activations are treated the same way as the neuron responses (see Figure 2.3).

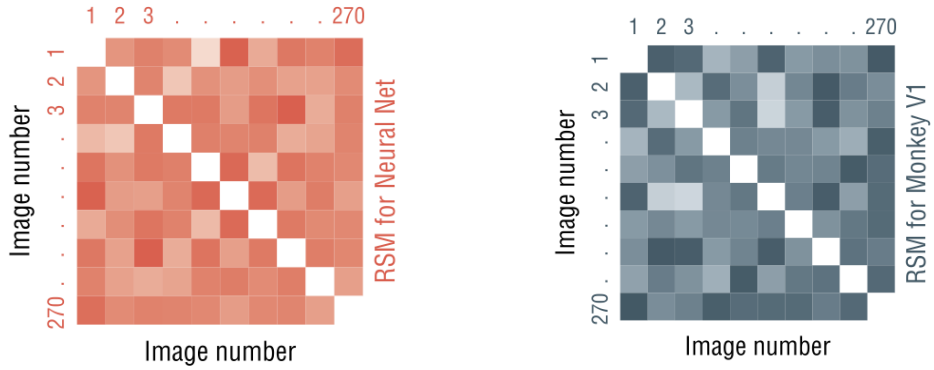


Figure 2.3: Federer et al. used a set of 270 images simultaneously to compute the RSMs for the model's hidden representation and the neural data. Image adapted from Federer et al. 2020.

The objective is that the model's hidden representation would start to mimic the brain's representations. In other words, the hidden representation of the image will start to resemble the neural data from the same image. This gives the cost function:

$$cost_{RSM} = \sum_{i,j}(RSM_{i,j} - \widehat{RSM}_{i,j})^2 \tag{2.1}$$

This formulation imposes a small cost when the entries of the two RSM matrices are similar and a high cost if there is a large difference. Federer et al. created a composite cost function where this cost for the RSM-similarity task and the cost for the classification task are both included. Note that the classification cost can be computed on a different image dataset than the RSM-similarity task; therefore, the costs use different indices. The composite cost function is:

$$\lambda \sum_{i,j}(RSM_{i,j} - \widehat{RSM}_{i,j})^2 - \sum_{k} \hat{y_k} log(y_k) \tag{2.2}$$

where y_k is the indicator variable for the true class of the image and $\hat{y}_k$ is the probability the model assigns to the true class.

With:

$$r = \lambda \frac{[\sum_{i,j}(RSM_{i,j} - \widehat{RSM}_{i,j})^2]}{\sum_k \hat{y_k} log(y_k)} \tag{2.3}$$

the model adjusts the parameter λ to keep r constant. r is a hyperparameter set by the researcher. This adjustment controls the contribution of each half of the composite cost function.

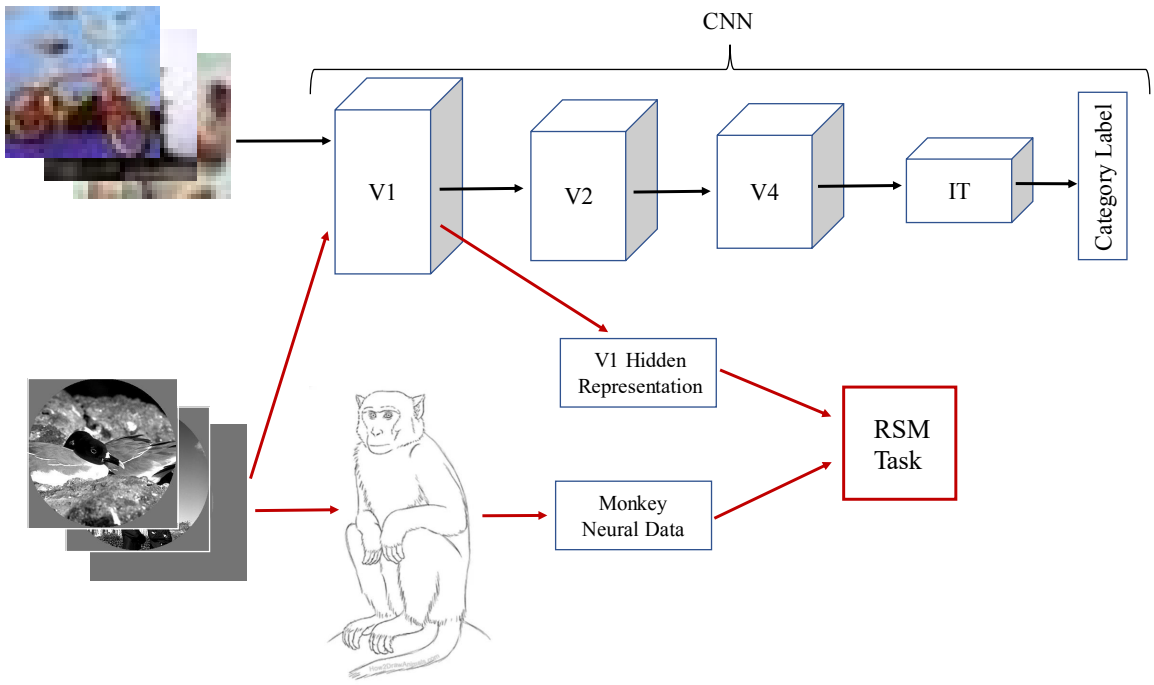
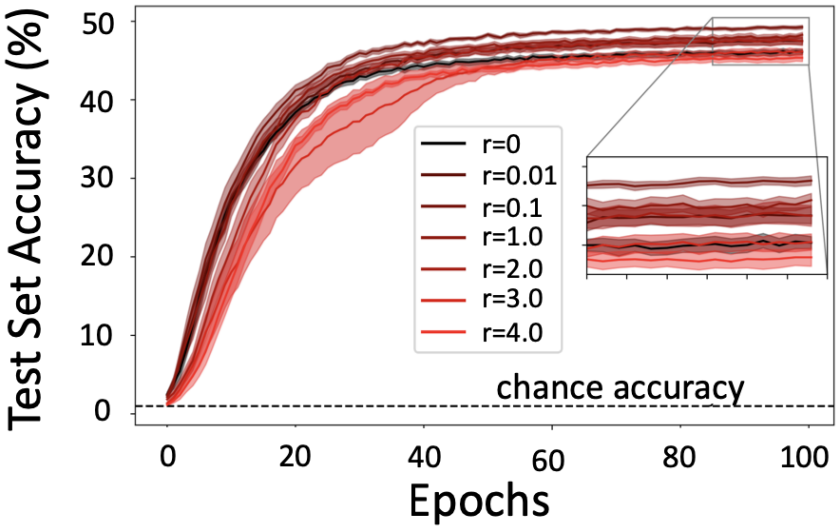
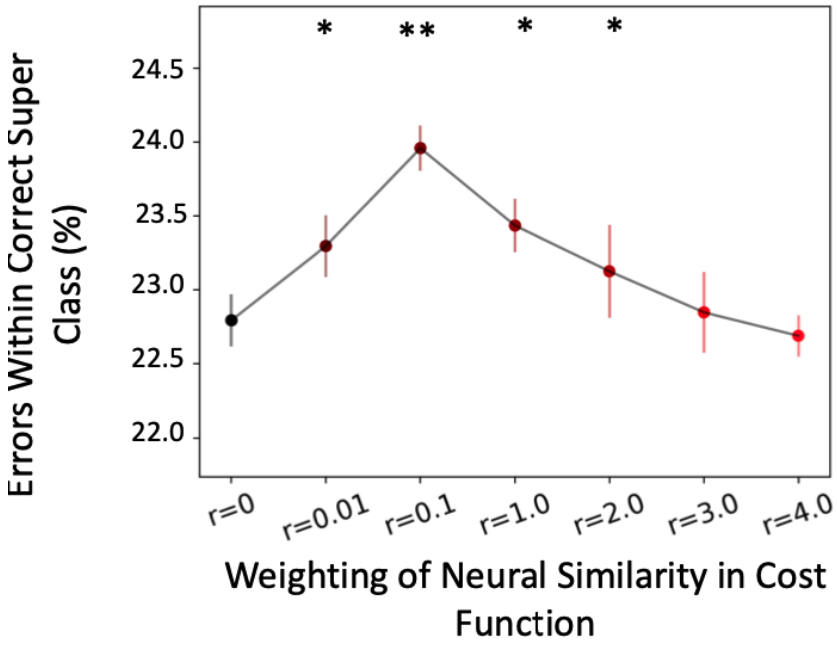


Figure 2.4: The two tasks the regularized network performs. The top row is a classification task, where the inputted images have associated labels and the CNN tries to assign those labels to the images. The bottom row shows the RSM-similarity task where the network tries to optimize the resemblance between the model's hidden representations and the neural data.

The cost function in Equation 2.2 allowed the researchers to achieve promising results using the representations from the V1 layer of the CORnetZ [17] model (for more information, see: Section 3.2) with the CIFAR-100 [16] dataset (see: Section 3.3.1) for classification. Not only did Federer et al. observe that the classification accuracy improves with this regularizer compared to the unregularized model (Figure 2.4(a)), but they also observed improved Super-Class accuracy (refer to Section 1.1 for more details on Super-Class accuracy). Figure 2.4(b) shows that when there was a misclassification in the regularized model, it was more likely that it was misclassified within the correct super-class.



(a) Accuracy using the RSM regularizer.



(b) Super-class accuracy using the RSM regularizer.

Figure 2.5: Federer et al.'s results. Using their regularizer, the model is able to outperform the unregularized model ($r = 0$) in both accuracy and super-class accuracy. The best results use $r = 0.1$. Images from Federer et al. 2020 [9].

Although the RSM constraint yields successful results in evaluating the hidden representation similarities between the brain and the CNN, there are weaknesses. Firstly, the **pairwise** limitation- the RSM method was limited to using pairs of images to build the representations. With this limitation, the model is prevented from constructing the representation using more than two images at a time. Therefore, there may be missing connections that could be found in groups of images. Secondly, the **linear** limitation- these representations are restricted to linear information; the model cannot create more complex relationships beyond simple linear combinations. Finally, the RSM can not **reweight** certain units of the hidden representations more than other unuseful units. That is, no matter how similar a unit is to the neuron responses it will be weighed the same as a unit that is not similar to the neuron responses.

With these limitations considered, and because the regularization has already been shown to improve the model, we hope that using a more flexible metric than RSM will yield more remarkable results.

2.5 Chapter Summary

In this chapter, we introduced CNNs and their important features. Then, we discussed the interconnectedness between neuroscience and machine learning algorithms, especially CNNs. We reviewed previous works that attempted to use this information to create better and more brain-like models. We then examined Federer et al.'s work and determined the validity of their experiments as well as weaknesses that could be addressed with a better metric than RSM. In Chapter 4 we will introduce the tools or pieces needed to construct our improved model.

Chapter 3

Tools

Here we introduce the tools needed for our new method. First, we explore Canonical Correlation Analysis (CCA), a method used to reduce data and uncover the causes of correlation between different sets variables. We come to the conclusion that CCA does not address all of the concerns from the previous RSM method and is not as computationally efficient as other methods. We introduce Deep Canonical Correlation Analysis (DCCA), a possible metric to improve and remedy Federer et al.'s [9] previous RSM method. Then, we present the base model we use for classification, CORnetZ. Lastly, we will familiarize ourselves with the datasets we use, as well as the preprocessing procedure required for one of the datasets.

3.1 DCCA - A New Metric

Here we explore possible metrics that will alleviate the **pairwise**, **linear**, and **reweighting** limitations of the past RSM method in Section 2.4. CCA is introduced as a possible candidate, but we find that a deep network version of CCA would be best suited for our model.

3.1.1 CCA

Canonical Correlation Analysis (CCA) is a statistical method used to reveal relationships between two **views** (sets of variables that represent the same data). CCA is

also used as a tool for data reduction. Here we use CCA for its first property, but also find utility in reducing the size of the data.

Mathematically the two views can be represented as random vectors: $\mathbf{x}_{P\times 1}$ and $\mathbf{y}_{Q\times 1}$.

So we can represent the covariance structure as:

$$cov(\begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix}) = \begin{bmatrix}\Sigma_{xx} & \Sigma_{xy}\\ \Sigma_{yx} & \Sigma_{yy}\end{bmatrix} \tag{3.1}$$

Where:

$cov(\mathbf{x}) = \Sigma_{xx(P\times P)}, ,$ the covariance matrix of random vector $\mathbf{x}$

$cov(\mathbf{y}) = \Sigma_{yy(Q\times Q)}, ,$ the covariance matrix of random vector $\mathbf{y}$

$cov(\mathbf{x}, \mathbf{y}) = \Sigma_{xy(P\times Q)} = \Sigma^T_{yx(Q\times P)}, ,$ the covariance matrix between random vectors $\mathbf{x}$, $\mathbf{y}$

We want to find two coefficient vectors, $\mathbf{a}_{1(P\times 1)}$ and $\mathbf{b}_{1(Q\times 1)}$, which maximize the correlation between the $\mathbf{x}$ and $\mathbf{y}$:

$$\rho = corr(\mathbf{a}_1^T\mathbf{x}, \mathbf{b}_1^T\mathbf{y}) = \frac{\mathbf{a}_1^T\Sigma_{xy}\mathbf{b}_1}{\sqrt{(}\mathbf{a}_1^T\Sigma_{xx}\mathbf{a}_1)\sqrt{(}\mathbf{b}_1^T\Sigma_{yy}\mathbf{b}_1)} \tag{3.2}$$

$(U_{1(1\times 1)}, T_{1(1\times 1)}) = (\mathbf{a}_1^T\mathbf{x}, \mathbf{b}_1^T\mathbf{y})$ are the first canonical pair. We see that U_1 is a linear combination of the elements in the $\mathbf{x}$ vector. Similarly, T_1 is a linear combination of the elements in the $\mathbf{y}$ vector.

To get the second canonical variate pair (U_2, T_2), we again find $\mathbf{a}_2$ and $\mathbf{b}_2$ such that ρ in Equation (3.2) is maximized. $\mathbf{a}_2$ and $\mathbf{b}_2$ are now also subject to the constraint:

$$\mathbf{a}_1^T\Sigma_{xx}\mathbf{a}_2 = \mathbf{b}_1^T\Sigma_{yy}\mathbf{b}_2 = 0 \tag{3.3}$$

Equation (3.3) ensures there is no correlation between U_1 and U_2. Similarly, the constraint ensures there is no correlation between T_1 and T_2. This continues for all min{P,Q} pairs so that:

$$\mathbf{a}_i^T \Sigma_{xx} \mathbf{a}_{i+1} = \mathbf{b}_i^T \Sigma_{yy} \mathbf{b}_{i+1} = 0, \text{ for all } i \in \{1, 2, ..., min\{P, Q\}\} \tag{3.4}$$

CCA can reduce the dimension of the data by storing the covariance information in linear combinations of the vectors. Furthermore, the CCA algorithm finds the most correlated linear combinations that have not already been found. Thus, the covariance of each pair will be less than the covariance of the pairs before it. For example, (U_1, T_1) will account for the most covariance, and $(U_{min\{P,Q\}}, T_{min\{P,Q\}})$ will have the least.

3.1.1.1 Solution of CCA

We can find the correlations of each canonical pair without explicitly finding the pairs themselves. This is done using singular value decomposition and is detailed in Kanti V. Mardia et al. [14]. Here we provide a brief overview of this method.

For $\mathbf{x}$ and $\mathbf{y}$ the canonical pair coefficients can be found from the eigenvectors, and the correlations from the eigenvalues of the following:

$$C = \Sigma_{xx}^{-1/2} \Sigma_{xy} \Sigma_{yy}^{-1} \Sigma_{yx} \Sigma_{xx}^{-1/2} \tag{3.5}$$

More simply, the orthonormal eigenvectors of C, $\mathbf{h_i}$, and the orthonormal eigenvectors of C^T, $\mathbf{g_i}$, are the canonical pairs of standardized $\mathbf{x}$ and $\mathbf{y}$ in order. The non-zero eigenvalues of C, ρ_i, are the square of the corresponding correlation between the elements in a pair.

$$C\mathbf{g_i} = \rho_i \mathbf{g_i}, \; C^T \mathbf{h_i} = \rho_i \mathbf{h_i}$$

So that:

$$(U_i, T_i) = (g_i^T \Sigma_{xx}^{-1/2} \mathbf{x}, h_i^T \Sigma_{yy}^{-1/2} \mathbf{y}) \implies \mathbf{a}_i = g_i^T \Sigma_{xx}^{-1/2}, \; \mathbf{b}_i = h_i^T \Sigma_{yy}^{-1/2}$$

Further,

$$corr(U_i, T_i) = \sqrt{\rho_i}$$

CCA always has a closed-form solution provided that the covariance matrices are nonsingular; this is because the solution to CCA requires the inversion of the covariance matrices. However, it is computationally expensive to find the solution when the number of variables is large. We plan to use this metric with the hidden representations of a deep neural network which has massive dimensions. To invert the covariance matrix of a neural network's hidden representation would be expensive and unreasonable to perform multiple times.

3.1.2 Deep CCA

To remedy the computational burden caused by finding the closed-form of CCA, we consider a method that will further reduce the data with minimal loss of information. Principal Component Analysis (PCA) [8], a data reduction technique, is a possible option to preprocess the data before performing CCA. However, PCA can also become computationally expensive on large datasets, and it has been observed to discard important information in more detailed classification tasks such as fine-grain classification. PCA also does poorly when the underlying structure in the data is nonlinear [19].

A gradient descent version of CCA is also a possibility. However, it would be expensive as we would still need to reduce the data with PCA or another data reduction method after each gradient update. Also, it is still less desirable than finding a closed-form solution or a solution that considers nonlinearity.

Deep CCA (DCCA) addresses the shortcomings of these methods and provides other benefits. DCCA is a deep network version of CCA in which the two views of data each go through their own independent deep network. After going through these sub-networks, the final representations of the views are brought together, and CCA is performed on these final representations (see Figure 3.1). This branched network

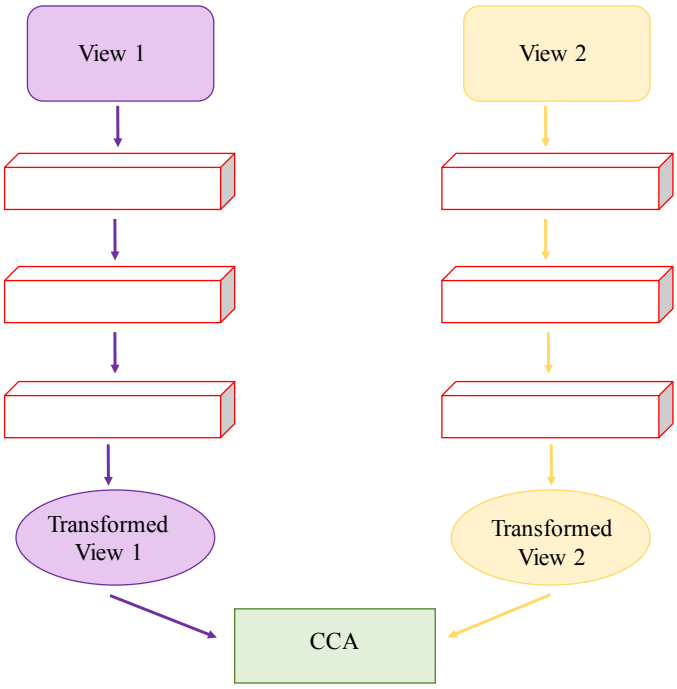


Figure 3.1: A visual aid for Deep CCA. Each view goes through its own network, transforming into their final form. CCA is then performed on these final forms, and the result is backpropagated through the layers of each independent network.

uses CCA as its loss function. CCA will find the pairs that maximize the correlation from the new reduced representations. Finally, with the cost computed, we can use back-propagation to update the weights. Note that the CCA cost can be implemented as seen in Section 3.1.1.1 as we do not need to know the explicit formulation of the canonical pairs.

Deep CCA not only provides a fast and dynamic way to find the relationships or correlations between the two data views, but it also reveals more complex relationships between the two views. Deep CCA can produce nonlinear representations through the use of nonlinear activation functions in its sub-networks' layers. This alleviates the **linear** limitation in Section 2.4 and prevents relationships between two views that may be significant and nonlinear from being neglected. Also, the linear combinations in the CCA cost rectifies the **reweighting** limitation. Further, using batches of images is a solution for the **pairwise** limitation previously seen.

3.1.3 Technical Description of Deep CCA

DCCA uses two independent deep neural networks for each data view, $\mathbf{x}$ and $\mathbf{y}$. The networks can have any type or number of layers. The network of $\mathbf{x}$ has d_x many layers, and $\mathbf{y}$ has d_y many layers. Let $f_{\mathbf{x}}(\mathbf{x})$ be the final representation of $\mathbf{x}$ after being processed through its sub-network, similar for $\mathbf{y}$. Then, the objective is to maximize the correlation between the two final representations $f_{\mathbf{x}}(\mathbf{x})$, $f_{\mathbf{y}}(\mathbf{y})$.

To solve this we look at:

$$h_{\mathbf{x}}^i = s_{x_i}(W_{\mathbf{x}}^i h_{\mathbf{x}}^{i-1} + b_{\mathbf{x}}^i) \tag{3.6}$$

where $W_{\mathbf{x}}^i$ are weight matrices, $b_{\mathbf{x}}^i$ are the biases, s_{x_i} is some nonlinear function, and $h_{\mathbf{x}}^i$ are the hidden representations of the ith layer. Note if i = 0, then $h_{\mathbf{x}}^i$ is the original view of the data $\mathbf{x}$. We can then represent $f_{\mathbf{x}}(\mathbf{x})$ as:

$$f_{\mathbf{x}}(\mathbf{x}) = s_{d_x}(W_{\mathbf{x}}^{d_{\mathbf{x}}} h_{\mathbf{x}}^{d_{\mathbf{x}}-1} + b_{\mathbf{x}}^{d_{\mathbf{x}}}) \tag{3.7}$$

And so we have our optimization function:

$$\Theta_{\mathbf{x}}, \Theta_{\mathbf{y}} = argmax_{\Theta_{\mathbf{x}}, \Theta_{\mathbf{y}}} corr(f_{\mathbf{x}}(\mathbf{x}), f_{\mathbf{y}}(\mathbf{y})) \tag{3.8}$$

Where $\Theta_{\mathbf{x}}, \Theta_{\mathbf{y}}$ contain the weights (W) and biases (b).

After the views are transformed and their dimensions are reduced by their respective networks, their final form is a nonlinear representation of the original view. The two transformed views are used to calculate the DCCA's loss function. This function is the same as the CCA objective we preciously saw in Equation (3.2), but with functions of the views instead of views themselves. In other words, $f_{\mathbf{x}}(\mathbf{x}), f_{\mathbf{y}}(\mathbf{y})$ are now the $\mathbf{x}$ and $\mathbf{y}$ in Section 3.1.1.

The resulting CCA calculation, which we use as our loss function, is found using a series of linear operations as seen above in Section 3.1.1.1, so the result is a linear combination of the entities fed to it. However, because the $f_{\mathbf{x}}(\mathbf{x}), f_{\mathbf{y}}(\mathbf{y})$ are nonlinear

forms of the original views, $\mathbf{x}$ and $\mathbf{y}$, the CCA loss is inputted nonlinear versions of the original features. So the CCA loss will produce a linear combination of the nonlinear combinations of features, which will again be nonlinear with respect to the input features.

The sub-networks for each of these views have their own hyperparameters to be tuned. The activation functions, dropout rates, and learning rates are all to be adjusted for the separate networks depending on the type of data being used and the performance of the overall model. DCCA reduces the dimension of the data creating a more computationally efficient method for a similarity task than other methods we have explored. Further, DCCA allows us to find a more intricate and a more complete idea of the correlation in the data by creating nonlinear combinations of the views. This elevates the **pairwise**, **linear**, and **reweighting** limitations of the previous RSM approach (Section 2.4).

3.2 CORnetZ

Similar to Federer et al. [9], we use the CORnetZ model [17] as our CNN. The CORnet family was created to be structurally and behaviourally similar to the mammalian visual pathway, making the family perfect candidates for an object recognition task. The CORnet models have a V1, V2, V4, and IT layer. These layers map by name and order to the lower visual system of mammals as shown in Figure 3.2. The final layer is a softmax classification layer. This family of models was also created to be as simple as possible while being correlated to the primate ventral visual pathway based on Brain-Score [26], a composite benchmark for comparing models to the brain. In other words, the creators of the CORnet family try to improve object recognition performance while preserving the similarities the model has to brain activation patterns.

We choose to use the simplest member of the family, CORnet-Zero or CORnetZ. The layers of CORnetZ each contain a convolutional layer followed by a pooling layer. Although this model is not a top-performing CNN, it gives us a clear place to amend

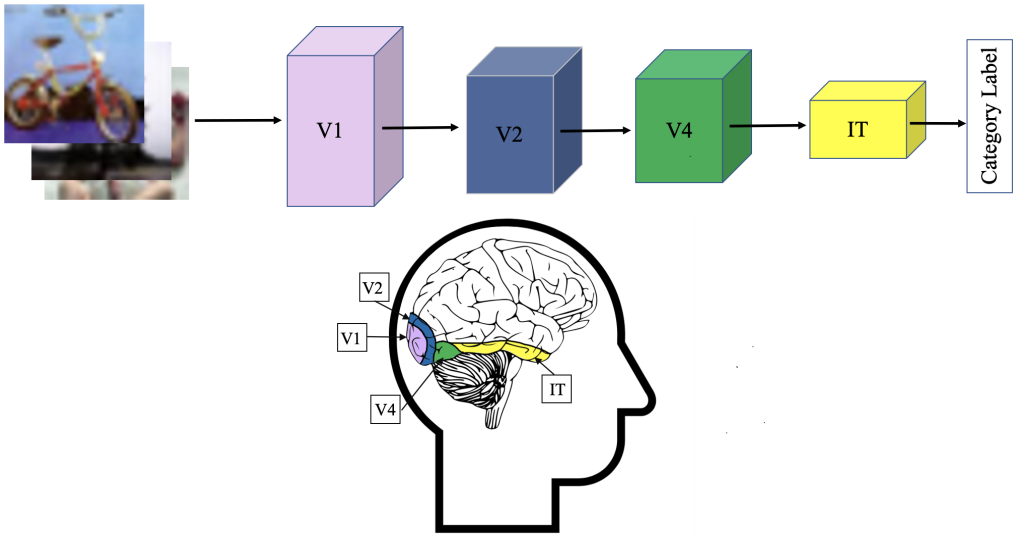


Figure 3.2: Architecture of the CORnet family [17]. Each layer in the model maps in name and order to an area in the brain's visual pathway.

the model with neural data (see Section 3.3.2) depending on where in the brain the data is collected (in our case, the V1 layer). Also, it will give us more room to improve the model, whereas if we use a model that results in near-perfect classification, it will be difficult to see if any meaningful change resulted from our method.

3.3 Data

3.3.1 Classification Data

For our classification task we trained on the CIFAR-100 dataset [16], which consists of coloured 32x32 images. This dataset has 100 classes, each containing 600 images. The authors of this dataset specify a training split of 500 images for each of the 100 classes for training. The remaining 100 images of the 100 classes are used as a test set. Figure 3.3 provides some examples from CIFAR-100.

The CIFAR-100 dataset is aggregated into 20 super-classes, each fully containing 5 of the classes. These super-classes are also intuitive in that similar classes are grouped. For example, the super-class 'Large Carnivores' contains the classes 'Bear,'

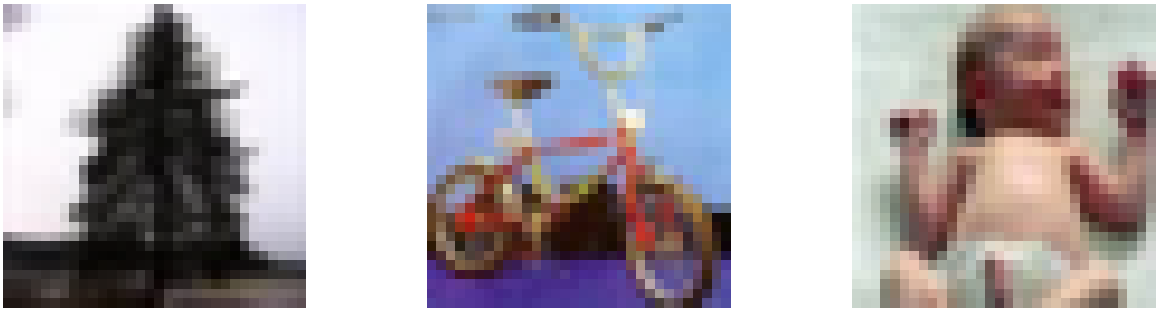


Figure 3.3: Example images from CIFAR-100. From left to right, we have labels: 'Pine tree,' 'Bicycle,' and 'Baby.'

'Leopard,' 'Lion,' 'Tiger,' and 'Wolf' (see Figure 3.4).

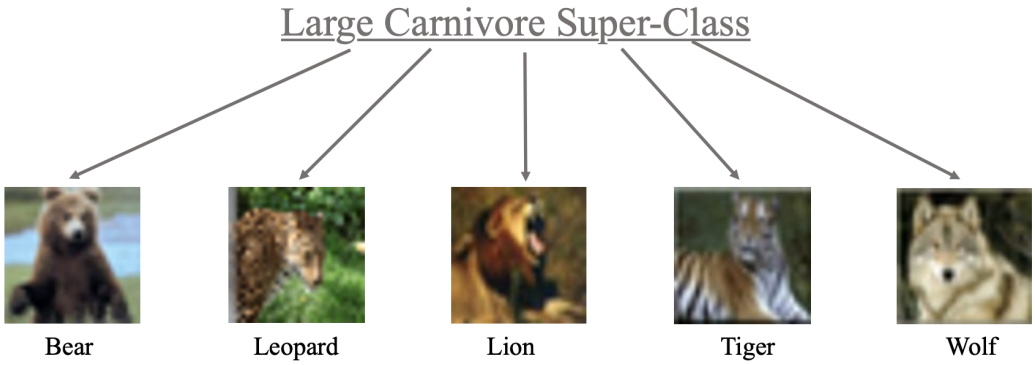


Figure 3.4: Example images from CIFAR-100's [16] Large Carnivore Super-Class.

3.3.2 Brain Data

The data we use as one of the views for our DCCA model is from Coen-Cagli et al. [6]. This data was collected from 3 different fully anesthetized macaque monkeys using the Utah electrode array. The multi-electrode recordings are from the primary visual layer, also termed the V1 layer, of the subjects' visual pathway. Natural images and gratings were presented to the subjects presented in two windowed sizes and various orientations totaling 956 different stimuli (Figure 3.5).

With 3 subjects, 10 sessions were recorded. Subject 1 contributed 1 session, Subject 2 contributed 6 sessions, and Subject 3 contributed 3 sessions. Neural recordings

Figure 3.5: Example images from Coen-Cagli et al.'s [6] experiment. Note that there are no labels associated with this dataset.

from each session are stored in a multidimensional array consisting of the following variables:

- **Neurons** - there are a total of 94-106 (depending on the recording session) instances of this variable, each representing a different neuron. It is important to note that due to the method chosen (electrode implantation), it is unlikely that the equivalent neuron was found in different monkeys or even in different sessions of the same monkey. Electrode implantation is a less sophisticated method than the state-of-the-art methods for brain data collection, such as fMRIs. For example, the neuron could have died from previous implantation of the electrode. However, for our purposes, the exact neuron does not need to be re-recorded, and the neighboring neurons suffice as our preprocessing will take these imprecise measures into account, and we are more interested in the V1 layer as a whole.

- **Images** - 956 different stimuli images are presented. They are all grey-scaled natural images with pixel size 227x227x3. The 3 in the pixel size tuple denotes the image's RGB values, which determines the pixel's colour. Although these are grey-scaled images, we use RGB values because the images need to be in the same form as the classification data, which is coloured. Some images are repeated but presented in a different orientation or zoomed in or out.

- **Time** - neural responses from the subjects were recorded for the duration that the stimuli images were presented (106ms). There is no neural data given for the time between images.

- **Trials** - within each session, there are 20 sub-sessions (trials). In each trial, all 956 stimuli images are shown to the subject.

3.3.2.1 Preprocessing the Monkey Data

We are interested in how different neurons react to different stimuli to build a meaningful representation for the neural network to mimic. Therefore, we look to reduce the data into an (images x neurons) format because this format shows clear similarity to the structure of the network's hidden representation (images x hidden units).

For each session, we consider the 106ms that are recorded and choose to exclude the tail ends to avoid problems such as response latency. The time is then cropped to 50ms-100ms. We then average over the 20 trials leaving us with images x time(ms) x neurons (956 x 50 x 94-106). Because of the unpredictability of neurons, we consider each neuron at each recorded millisecond as its own 'pseudo' neuron, or measurement of interest. This further gives us images x pseudo-neurons (956 x 4700-5300).

At this point the dimension of our data is still large for being auxiliary information for our model, and each session has different dimensions. Ideally, we would like each session to contribute equally. Therefore, because we will further process the data in a linear combination in the DCCA model, we are comfortable reducing the dimension of the data via linear combinations. To do this, we use PCA to find a predetermined amount, in our case, 80, linear combinations of the 'pseudo' neurons against the image dimension. This is done so that each session has the same number of recorded neurons and so the least amount of information (or variance) is lost while doing so. These 80 representations are now our measurements of interest.

After the previous preprocessing, each session now has 956 images x 80 principal components. Again, because equivalent neurons were not captured between monkeys,

we simply concatenate these representations for all 10 sessions for each image. Our final form of the data is 956 images x 800 principal components. Throughout the rest of this work, we will refer to this preprocessed data as our network's **'Brain-View'**.

3.4 Chapter Summary

Here we presented different tools we need to put our model together. First, we explored CCA, a possible solution to the RSM approach's limitations. However, through a detailed discussion, we came to the conclusion that a deep network version of CCA called DCCA would be best suited for our model. We then explored an overview as well as mathematical details of DCCA. Next, we introduced the CORnetZ model, the CNN we will use. We explain why it was created and why it is an appropriate model to use with neural data. Lastly, we introduced the datasets we used. The first dataset is the popular CIFAR-100 dataset that will be used in the classification task by the CORNetZ model. The second dataset is a neural dataset consisting of monkey neuron firings and its associated images. Then, we described the preprocessing of this neural dataset and the result is the **Brain-View** for our DCCA model. In Chapter 4 we will assemble the tools discussed here to form our combined DCCA-CNN model.

Chapter 4

Methods

To improve the performance of classification algorithms, we propose a regularizer that uses neural data. In previous chapters, we discussed the origins of CNNs from neuroscience and how unconstrained neural networks can achieve brain-like representations. We detailed previous works that took advantage of these findings to improve neural networks. In this chapter, we improve upon Federer et al.'s [9] RSM approach in Section 2.4 by using a DCCA-based neural data (ND) regularizer. To do this we first construct a combined DCCA-CNN model from the tools introduced in Chapter 3.

4.1 New Model: CNN with a DCCA Regularizer

To alleviate the **linear**, **pairwise**, and **reweighting** limitations of Federer et al.'s work (Section 2.4) we design a CNN model with a DCCA model as the regularizer. Our CNN, the CORnetZ model, uses cross-entropy (CE) as its loss function ($\mathcal{L}_{\text{CE}}$). We add $\mathcal{L}_{\text{CE}}$ to the DCCA loss ($\mathcal{L}_{\text{DCCA}}$) to create a joint loss function $\mathcal{L}$. To control how much each loss contributes to the joint loss, we use a hyperparameter λ.

$$\mathcal{L}_{\text{CE}} = -\sum_i p_i \log(\hat{p}_i) \tag{4.1}$$

$$\mathcal{L}_{\text{DCCA}} = -corr\big(f_{\mathbf{x}}(\mathbf{x}), f_{\mathbf{y}}(\mathbf{y})\big)/F \tag{4.2}$$

$$= -\frac{1}{F}\sum_{j}^{F} corr(a_j^T \mathbf{x}, b_j^T \mathbf{y}), \text{ as in Equation (3.2)}$$

$$\mathcal{L} = \lambda\ \mathcal{L}_{\text{DCCA}} + (1-\lambda)\ \mathcal{L}_{\text{CE}} \tag{4.3}$$

where p_i is the indicator variable for the true class of the image and $\hat{p}_i$ is the probability the model assigns to the true class. We scale $\mathcal{L}_{\text{DCCA}}$ by F, the number of canonical pairs, so it has the same range as $\mathcal{L}_{\text{CE}}$, (0,1). λ can now be described as the proportion $\mathcal{L}_{\text{DCCA}}$ contributes to $\mathcal{L}$. For example, if $\lambda = 0$ then DCCA would not contribute to the loss function. If $\lambda = 0.5$, then $\mathcal{L}_{\text{DCCA}}$ and $\mathcal{L}_{\text{CE}}$ contribute equally to $\mathcal{L}$.

The stimuli images from Coen-Cagli et al. [6] (described in Section 3.3.2) that we use for our DCCA model, do not have associated labels, so we cannot use them to compute $\mathcal{L}_{\text{CE}}$. Instead, $\mathcal{L}_{\text{CE}}$ is computed with images from CIFAR-100; while $\mathcal{L}_{\text{DCCA}}$ is computed using the stimuli images from the monkey experiment along with the neural recordings. Because the size of the datasets differ, for each epoch of training on CIFAR-100, we cycle through the neural data 20 times. In our figures, the epoch count on the horizontal axes refers to the epochs of training on the CIFAR-100 data.

Next, we create the CORnetZ model originally developed by Kubilius et al. [17] (Section 3.2) and link a DCCA model to the CNN through a shared node. CORnetZ's V1 layer is chosen as the shared node because the neural data (Section 3.3.2) was collected from the subjects' V1 areas in the visual pathway. We also find V1 to be the most appropriate layer to be shared because CORnetZ was created to behave like the visual pathway with labelled layers for each brain area (see Figure 3.2).

The images described in Section 3.3.2 are inputted to CORnetZ's V1 layer. The V1 hidden representations of these images are then inputted as the **CNN-View** of the DCCA model. At the same time, the recorded neural responses to those same images are the **Brain-View** for the DCCA model. Batches of image/neural-recording pairs

are passed through the two DCCA sub-models. The final representations of the two views are then used in the CCA loss function, which is completely differentiable [4]. Finally, backpropagation proceeds through both sub-models, including the shared V1 node of CORnetZ.

At the same time, a classification task using the CIFAR-100 dataset and the CORnetZ architecture, including the V1 layer, also occurs. This results in a CNN model with DCCA attached acting as a regularizer, as depicted in Figure 4.1.

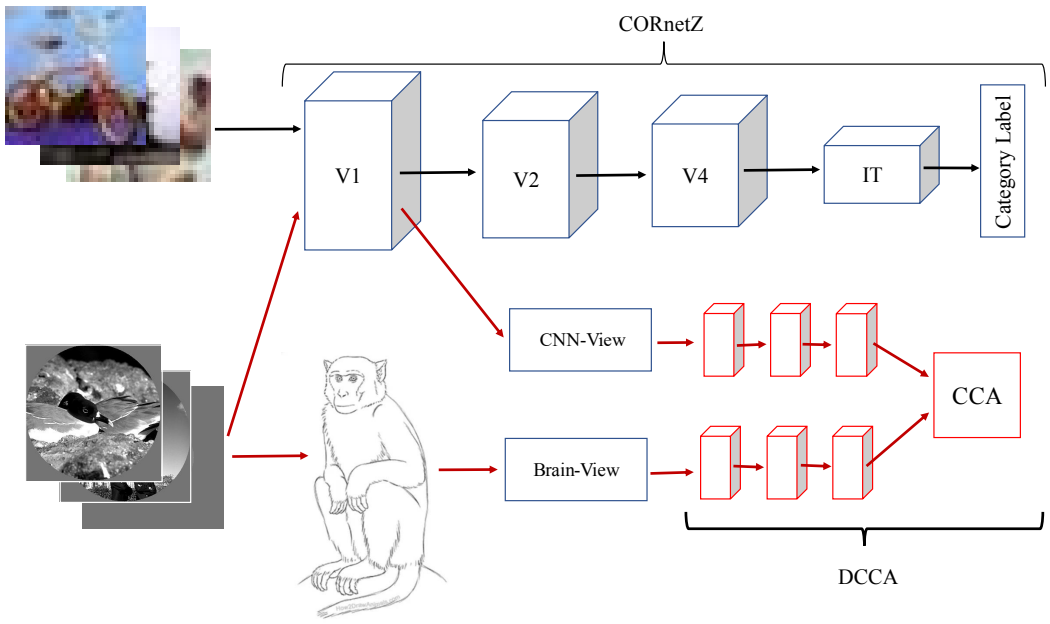


Figure 4.1: Architecture of the combined CNN-DCCA model. The V1 layer is shared between the CORnetZ model and the DCCA model. The DCCA's pipeline is in red, while the CORnetZ's pipeline is in black. Notice the similar structure to Federer et al.'s [9] model in Figure 2.4.

4.2 Experiment Details

4.2.1 Experimental Set-up

The platform Tensorflow [1] and its various libraries were used for this project. One such library, Keras [5], allowed us to work with the user-friendly Keras paradigm without sacrificing the computational power of Tensorflow. The architectures for the CORnetZ [17] model and the DCCA model were independently created. Then, we connect these two models together by having the DCCA model use the hidden representations for the brain images from CORnetZ's V1 layer as an input.This allows V1 to process batches of CIFAR-100 data for the CORnetZ to classify while also processing batches of the monkey images for the DCCA model. Even sharing a node, the DCCA branch and the CNN branch can both independently find their individual loss functions. These loss functions, $\mathcal{L}_{\text{DCCA}}$ and $\mathcal{L}_{\text{CE}}$, are then weighted by λ (and 1-λ) to calculate $\mathcal{L}$ before the back-propagation is performed.

4.2.2 Experimental Tuning

To determine the effects of the ND regularizer on CNN performance, we trained CORnetZ with different regularization strengths λ. For easy comparison, we used the same CORnetZ settings as Federer et al. [9]: learning rate of 0.01, CIFAR-100 batch size of 128, and a dropout rate of 0.5.

Each DCCA sub-model consists of 3 dense layers of width 1024, followed by a 0.0001 dropout layer, and finally, a dense layer with a width of 10. The DCCA branch of the network required independent hyperparameter tuning, which was accomplished by fitting the DCCA branch alone, without the CNN branch. We optimized for a minimal loss function, which is the negated sum of the correlations from the F identified canonical pairs. Because the amount of images in the brain experiment is small (956), we chose to use 10 canonical pairs to avoid exhausting the samples too quickly. To ensure that DCCA has sufficient information to extract for the pairs, we

use a batch size of 50. With these hyperparameters set, we found the optimal settings to yield high DCCA correlation were: a 0.00001 L2 weight decay, a Random Normal kernel initializer with a standard deviation of 0.01, and using the ReLU activation function.

The combined CNN-DCCA model was trained for 100 epochs and 5 randomly-seeded initializations were used for each tested regularization strength λ. We experimented with pretraining the DCCA before incorporating it into the CNN and found only a modest improvement. Because pretraining the DCCA adds computational overhead, and the effect on accuracy is negligible, we did not pursue this approach.

4.2.3 Experimental Controls

As a control, to determine if any of the performance gains obtained with the ND regularizer depended on the specific response of the brain to an image, we repeated our CNN regularization experiments with randomized neural data. This was accomplished in three ways:

1. **Shuffled Labels-** The images associated with each measured neural response were randomly permuted. This ensured that the shuffled dataset matched each neuron's distribution of firing rates but removed any of the neuron's sensitivity to specific image features, as the images were randomly assigned.

2. **V1 Statistics-** The mean and standard error was extracted from each of the 10 monkey recording sessions, and random data was produced using a normal distribution with the same statistics. This random data preserved the size of the real recordings, meaning the number of neurons observed in each session remained the same. These sessions were then preprocessed as in Section 3.3.2.1.

3. **N(0,1) Statistics-** Lastly, using the same sizes as the monkey sessions, images x neurons, random data is generated from a standard normal distribution (N(0,1)). Again, these sessions are then preprocessed as in Section 3.3.2.1.

These newly shuffled or generated datasets were then used in place of the real neural data in the model. We compared the difference in performance between the original neural data and this permuted or randomly generated data to reveal important information about the capabilities and limitations of the neural data and of the DCCA regularizer. More precisely, these experiments measure the extent to which the brain's image representations, as opposed to its other statistical properties, are helpful in the DCCA-regularizer for CNNs.

4.3 Chapter Summary

This chapter introduced our new model that attaches DCCA to a CNN model. We looked at the resulting composite cost function from the two models sharing COR-NetZ's V1 layer. Next, a comprehensive report of the experimental set-up including the libraries and the tuning procedure for both the DCCA and CNN models were discussed. Lastly, we introduced three generated datasets all with different degrees of likeness to the original **Brain-View** data. We will use these datasets as controls for the experiments presented in Chapter 5.

Chapter 5

Experiments and Results

In this chapter, we present experiments to determine the performance of our combined DCCA-CNN model. We determine the effect our model has on: Accuracy, Super-Class Accuracy, Robustness to Adversarial Examples. Next, visual aids, as well as thorough discussion of the results will be provided. We also offer possible interpretations and insights to the patterns that emerge with each experiment.

5.1 Accuracy

Accuracy is defined as the proportion of images that the model classifies correctly.

$$\frac{\#\text{correctly classified images}}{\#\text{images}}$$

5.1.1 Accuracy Results

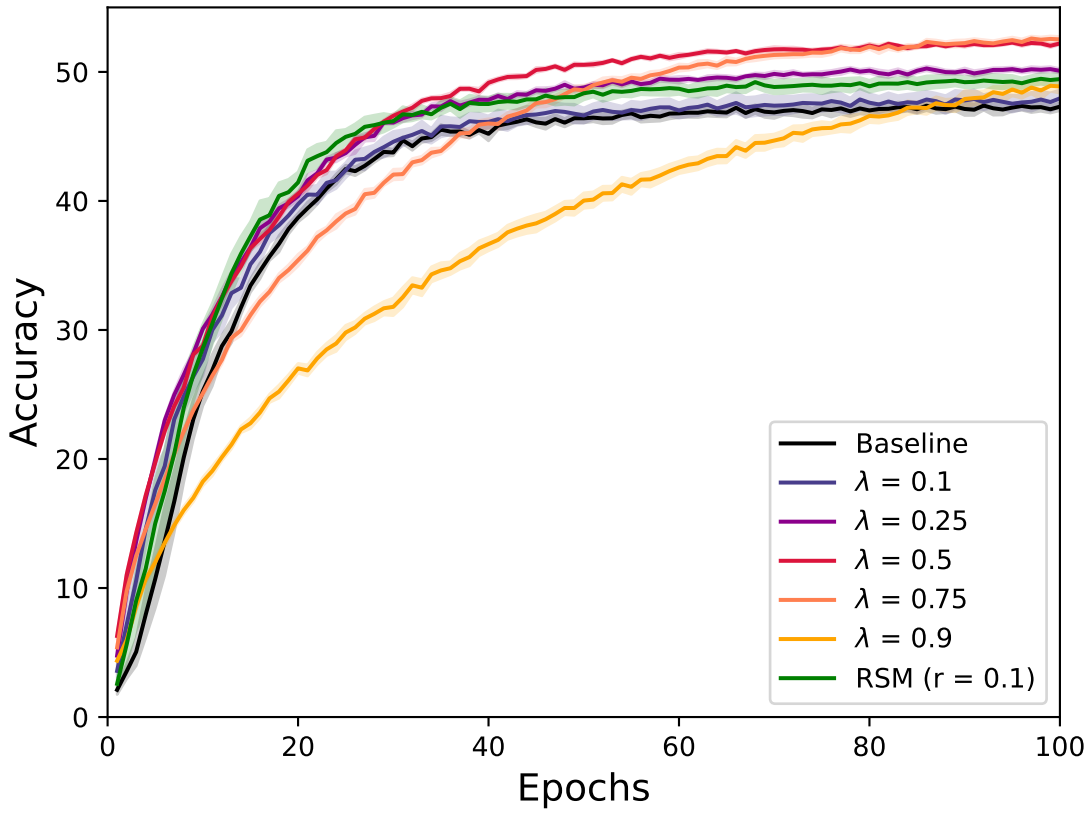
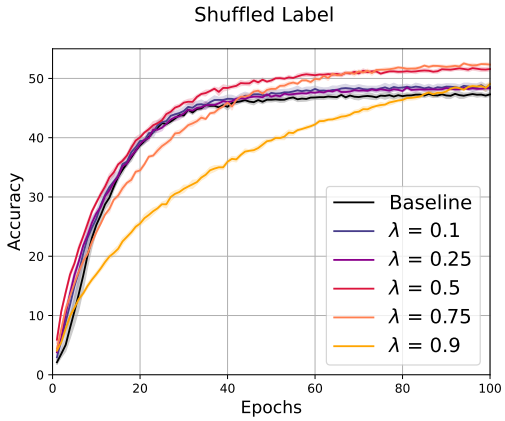
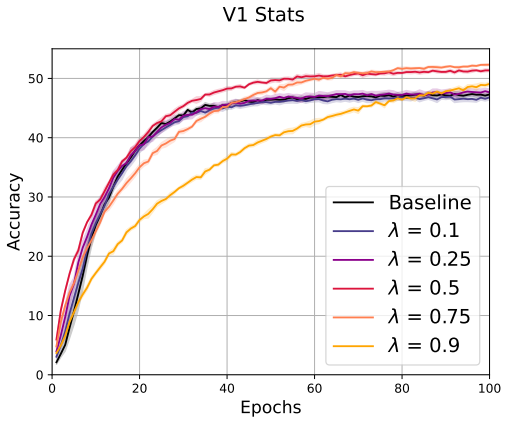


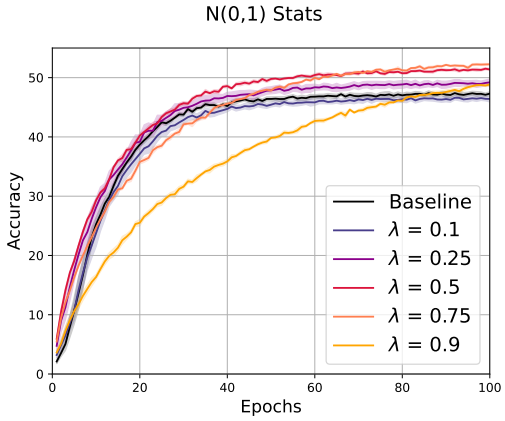
Figure 5.1: A comparison of accuracies achieved with different λ values in the ND regularizer. The best performing RSM model's results are also included. Note that these results are the mean of 5 repetitions, and the lighter colours around the bold lines are the standard error. We see that the mid-magnitude lambda values yield better performance compared to the baseline ($\lambda = 0$) and the RSM models. The accuracy achieved by the baseline is 47.38%. Whereas, the best accuracy with the regularizer is 52.61% with $\lambda = 0.75$ followed by 51.99% with $\lambda = 0.5$



(a) Accuracy using the ND regularizer and **Shuffled Labels** data



(b) Accuracy using the ND regularizer and **V1 Statistics** data



(c) Accuracy using the ND regularizer and **N(0,1) Statistics** data

Figure 5.2: A comparison of λ values for the ND regularized network using different transformed neural data described in Section 4.2.3. **(a)** Using Shuffled Labels neural data, similar to the original neural data, the best performance at 100 epochs is obtained from $\lambda = 0.75$ with 52.34%. **(b)** $\lambda = 0.75$ surpasses $\lambda = 0.5$ at around 70 epochs and is the best performing at 52.31% for the V1 Statistics data. **(c)** Again, for the N(0,1) Statistics data $\lambda = 0.75$ is the best performing λ value with 52.26% accuracy. Figure 5.3 compares the best results across all generated and original data.

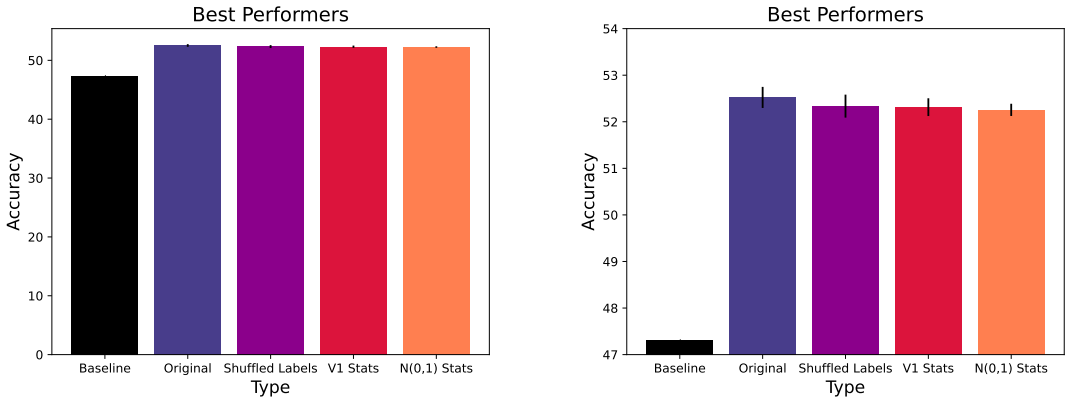


Figure 5.3: Comparison of the best performing setting for all of the generated or original neural datasets and the baseline model. Note that the vertical black lines are the standard error between runs at the 100th epoch for each λ value. $\lambda = 0.75$ is the best performer for each. We see that the original data does slightly better than the generated data. Note that **(b)** is the same data displayed but zoomed in, and the y axis starts at 47.

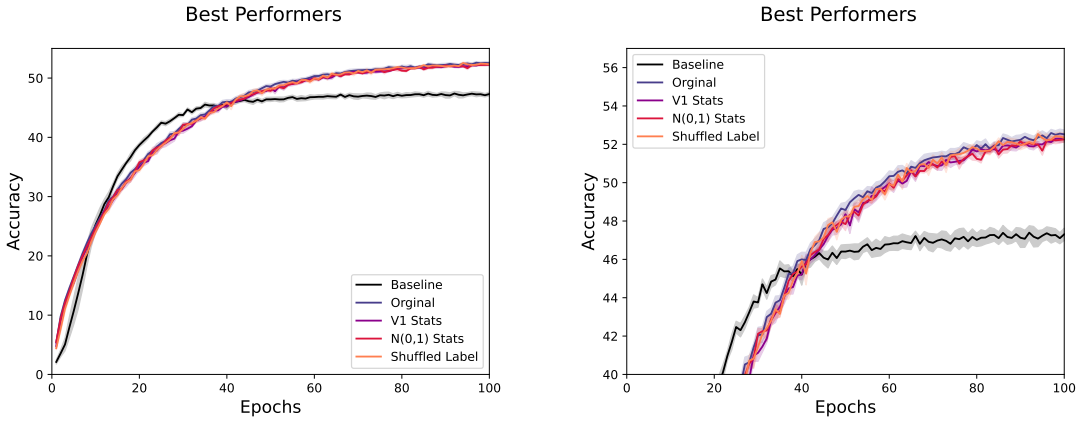


Figure 5.4: A comparison of the best performing settings for accuracy for each type of data. $\lambda = 0.75$ is the best performer for each at 100 epochs. We see they all follow a similar pattern. Note that **(b)** is the same data displayed but zoomed in, and the y axis starts at 40.

5.1.2 Accuracy Discussion

The accuracy experiments in Section 5.1.1 tested the effectiveness of the ND regularizer on accuracy. This is done with the CORnetZ model classifying the CIFAR-100 dataset. With the monkey neural recordings as our **Brain-View** of the DCCA and the monkey stimuli images as our **CNN-View** of the DCCA, we trained the model for 100 epochs. We found a large improvement in accuracy from the baseline model, which is the CORnetZ model without a regularizer. A reminder that the λ parameter controls the contribution of each half of the loss function: $\mathcal{L} = \lambda\mathcal{L}_{\text{DCCA}} + (1-\lambda)\mathcal{L}_{\text{CE}}$. Our implementation normalizes $\mathcal{L}_{\text{DCCA}}$ giving it the same range as the $\mathcal{L}_{\text{CE}}$ (0,1), which gives us a more intuitive understanding of λ. We can interpret λ as the percentage $\mathcal{L}_{\text{DCCA}}$ contributes to $\mathcal{L}$.

$\lambda = 0.75$ was optimal for the ND regularizer and achieved validation accuracy of 52.61% which is over 5% higher than the previous baseline achieved of 47.38%. Following closely behind, $\lambda = 0.5$ reached a validation accuracy of 51.99%. We observed that when λ is too small, it often still does better than the baseline, but not a significant amount. However, when λ is too large, it does not reach the baseline accuracy within 100 epochs. A pattern emerges with further inspection of the λ values. The smaller the λ, the quicker the model finds its plateau, but the larger values may achieve higher accuracy after more than 100 epochs. For example, the baseline can be considered as $\lambda = 0$, and it plateaus first, followed by $\lambda = .1, .25, .5$. Looking at Figure 5.1, we see that $\lambda = .75$ and .9 are still following an upward trend at 100 epochs.

To determine how much improvement is owed to the neural data itself, we rerun our analysis on the **Shuffled Labels**, **V1 Statistics**, and **N(0,1) Statistics** datasets described in Section 4.2.3. The **Shuffled Labels** dataset preserves the neural recording's latent structures and features, such as the distribution the data follows, in response to a visual processing task. At the same time, this data separates

the recording from the specific image that the subjects viewed. The **V1 Statistics** would preserve some lower-level statistics such as the mean and standard deviation but would not capture the same higher order statistics as the recorded data. Finally, the **N(0,1) Statistics** dataset is not influenced at all by the neural data and is simply generated from a standard normal distribution. We find that the more similar the data is to the actual neural data, the better the performance. That is, we find the original dataset to perform the best, followed by the **Shuffled Labels** dataset, then the **V1 Statistics** dataset, and finally the **N(0,1) Statistics** dataset. The significant improvement from the baseline may suggest that the neural data may not be as important as we previously thought as all of the datasets used in the ND regularizer significantly improved the baseline model. Furthermore, we observe the same trend or pattern in the λ values for all 3 of the generated datasets as the original data (see Figure 5.2).

Further experimented with turning the regularizer on and off at different epochs; essentially only using the DCCA ND regularizer for a subset of epochs. In doing so, we found that using the DCCA branch of the network for all 100 epochs continued to improve the model's accuracy, unlike previous work by Federer et al. [9] that found the most increase in accuracy was realized in the first 10 epochs. We also experimented with delaying the onset of the DCCA branch, contributing to various epochs but found this approach less beneficial.

5.2 Super-Class Accuracy

Next, we look at the performance of the ND regularized models for super-class accuracy. Although it is most desirable to correctly classify all samples, when errors in classification do occur, we prefer a model that makes more human-like errors than unnatural errors. For example, if an image of a shrew gets misclassified as a mouse, we would consider that a more reasonable mistake than if the shrew image was misclassified as a tractor image. The CIFAR-100 creators [16] have provided super-classes

to group concepts. For instance, the shrew class is contained in the same super-class as the mouse, while the tractor is not (see Figure 5.5). In humans, the ability to recognize these similarities or relationships between objects is innate. Because we are creating this model to make more brain-like representations, we expect the regularized model to achieve higher super-class accuracy than the unregularized model. We also explore how the generated and shuffled datasets from Section 4.2.3 perform at this task. This helps determine how much the ND regularizer or the neural data contribute to the improvement.

Figure 5.5: An example to demonstrate super-class accuracy. The image on the left is a shrew. It is more reasonable to classify it as a mouse (center-most) than a tractor (right).

To test the super-class accuracy of our models, we load the saved model weights at the 100th epoch from the accuracy experiments and run the test set through the model. Then the evaluated classification labels are mapped to the super-class it belongs to. The super-class accuracy is the proportion of examples mapped to their proper super-class, even if the example image is classified incorrectly.

5.2.1 Super-Class Accuracy Results

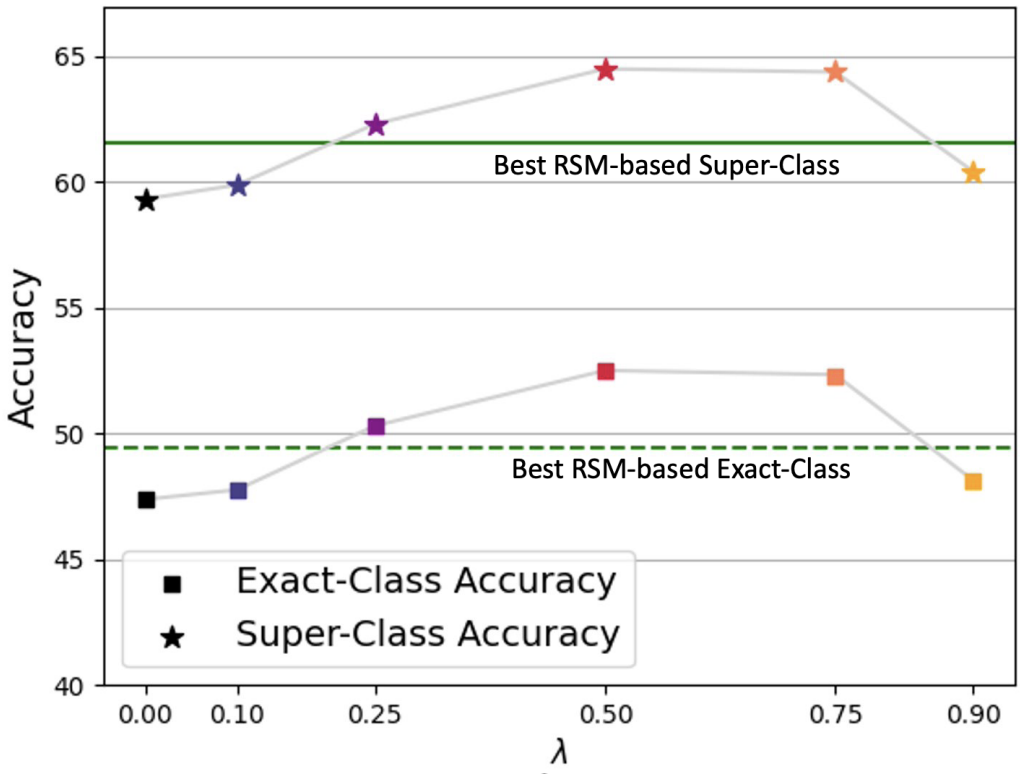
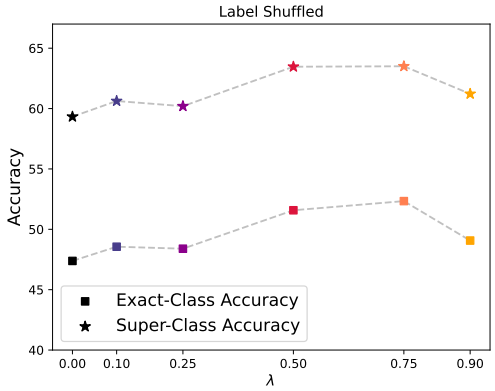
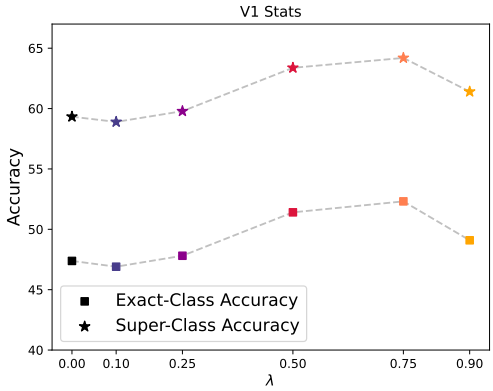


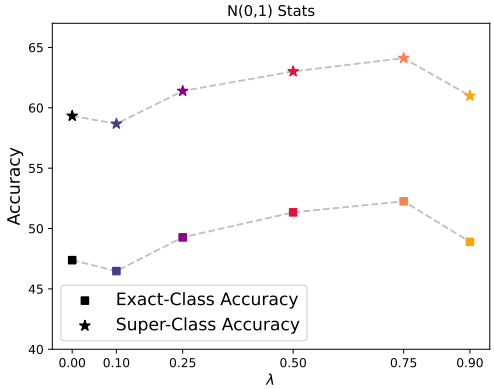
Figure 5.6: A comparison of super-class accuracies achieved with different λ values with the ND regularizer. The best performing RSM model's results are also included. The super-class accuracies follow a similar pattern to what is seen with the (true) accuracies in Section 5.1.1. $\lambda = 0.75$ performs the best with 64.61% accuracy followed by $\lambda = 0.5$ at 63.92%. Most of the tried λ values outperform both the baseline and the RSM models.



(a) Super-class accuracy using the ND regularizer and **Shuffled Labels** data



(b) Super-class accuracy using the ND regularizer and **V1 Statistics** data



(c) Super-class accuracy using the ND regularizer and **N(0,1) Statistics** data

Figure 5.7: A comparison of super-class accuracy for λ values with the ND regularized network using different transformed neural data. $\lambda = 0.75$ achieved the highest super-class accuracy for all transformations of the data. **(a)** Shuffled Label data achieved 63.50%. **(b)** V1 Statistics data achieved 64.19%. **(c)** N(0,1) Statistics data achieved 64.11%. A comparison of the best performances from each dataset are presented in Figure 5.9.

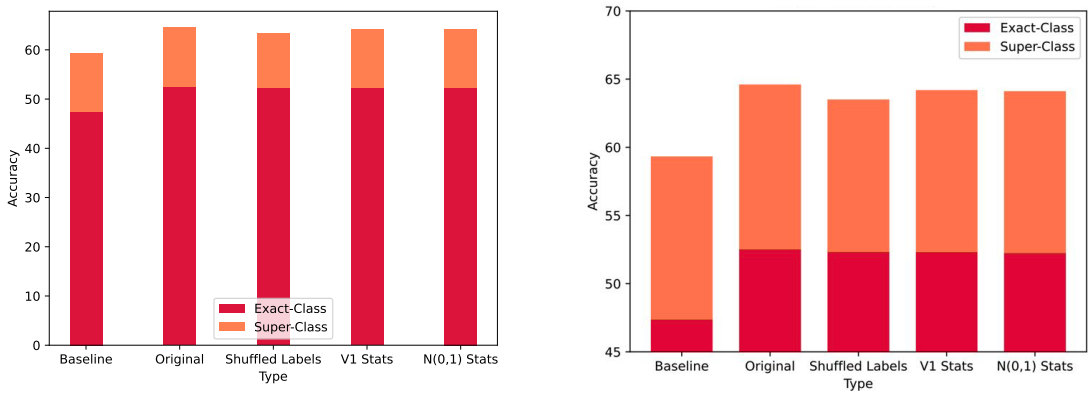


Figure 5.8: Stacked bar graph of the accuracies and super-class accuracies of the data variations from $\lambda = 0.75$ in the ND regularizer. (b) Is the same data displayed but zoomed in, and the yaxis starts at 45.

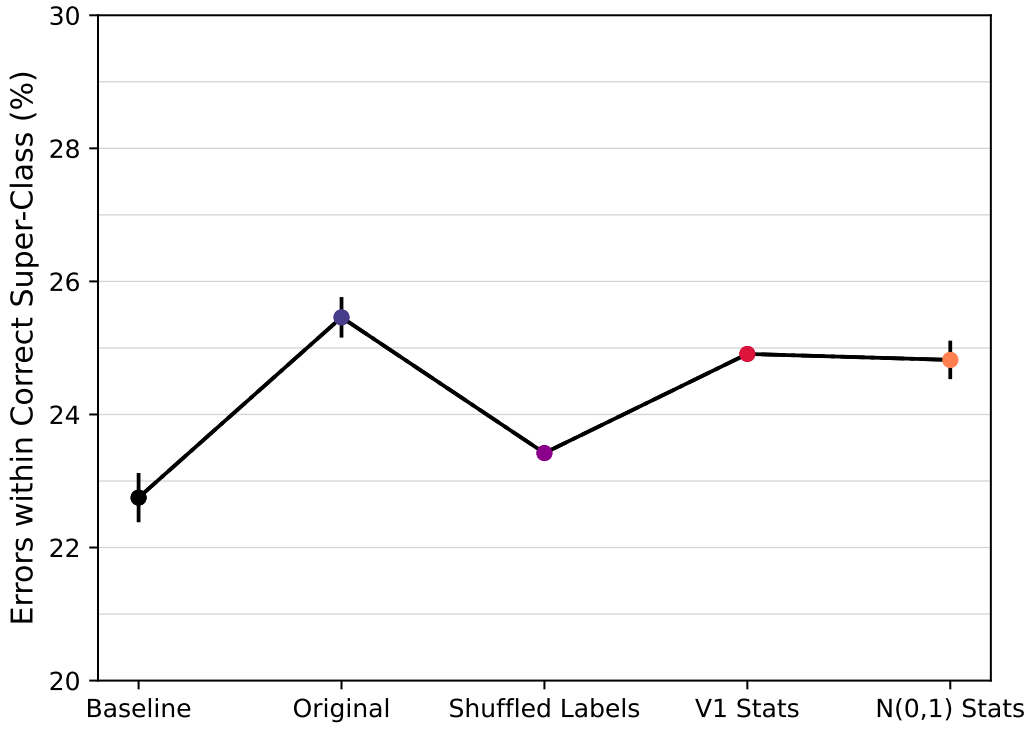


Figure 5.9: The proportion of errors within the correct Super-Class for the different datasets, including error bars. We see that the original dataset has the highest proportion of errors within the correct Super-Class.

5.2.2 Super-Class Discussion

We saw impressive results with the ND regularization on the model's accuracy. However, we are still interested in how reasonable of mistakes the model makes when it misclassifies an image. In other words, we wish to explore whether the model makes reasonable or intuitive mistakes or if it makes mistakes that are unhuman-like. The super-class experiments in Section 5.2.1 tested this.

Again we find that $\lambda = 0.75$ is the optimal λ value for all the datasets. Furthermore, for each individual dataset, the pattern of the super-class accuracy closely follows the same curve seen in the accuracy experiments for ordered λ as seen in Figure 5.6 and Figure 5.7. Like the accuracy results, we find that the original data does the best at correctly classifying the images in their proper super-class with 64.61% accuracy compared to the baseline's 59.33% super-class accuracy. However, unlike the accuracy results, we find that the second best performing dataset is the **V1 Statistics** dataset with a super-class accuracy of 64.19%, followed by **N(0,1) Statistics** dataset reaching 64.11% accuracy, and lastly, the **Shuffled Labels** dataset reaching 63.50% super-class accuracy.

Although we find the neural data most beneficial, these results may suggest that if we are not properly aligning the neural responses to the correct stimuli then it confuses the model. Based on Yamins et al.'s work [30] we know that CNNs produce brain-like representations . So perhaps by using a neural data regularizer with data that is not matched to the stimuli, both $\mathcal{L}_{DCCA}$ and $\mathcal{L}_{CE}$ are trying to form brain-like representations in the V1 hidden representation. However, these two representations may contradict each other for a given image, causing an overall lower gain of brain-likeness. Whereas, the other two datasets, **N(0,1) Statistics** and **V1 Statistics** are more similar to noise. That is, these datasets are adding information without disrupting the representations the model is developing on its own.

To view these super-class results another way, recall that all images classified cor-

rectly are also super-class classified correctly. We examine the percentage of images classified incorrectly but mapped to the correct super-class. Using the results from the accuracy experiments (Section 5.1.1) and the super-class experiments (Section 5.2.1), we can calculate the percent of examples incorrectly classified but properly super-classified. The percentages are as follows: original data 25.46%, **V1 Statistics** 24.91%, **N(0,1) Statistics** 24.82%, and **Shuffled Labels** 23.42%. The original neural data still has the most improvement even though it also has the highest accuracy. This means that the ND regularizer with original data had a lower proportion of images that were misclassified, but could be correctly super-classified. Yet, the original dataset still achieved a higher proportion of incorrectly-classified but correctly-super-classified images than the other datasets (Section 4.2.3).

5.3 Adversarial Robustness

We are not only interested in the accuracies of the models but also in how robust our new model is. Humans can identify objects in images properly even when slight modifications are made to the image (See Figure 5.10 for an example). Here we look at a specific robustness task where targeted noise is added to the image and fed to the model to classify it. If the model is able to classify these modified images correctly, then it is robust. The specific type of robustness we look at here is called Robustness to Adversarial Examples.

Adversarial examples are images that are altered to deceive the model into misclassifying the images. The fast gradient sign method (FGSM) [11] is a well-known technique for generating such derivative images. This method involves the typical forward propagation, like when training neural networks. Then, instead of adjusting the model weights in the direction of the gradients that will minimize the loss, the image pixels are adjusted to maximize the loss.

The actual image is altered by: $P_{new} = P_{old} + \epsilon * \Delta$, where P are the pixel values, ϵ is the attack strength and Δ is the gradients. Note that the resulting images are

Figure 5.10: An demonstration of an adversarial example. On the left is the original image of a stop sign, on the left is the same image but altered. The alterations produce an image that is still recognizable as stop sign to humans, but will often trick a model into classifying it incorrectly.

often unmistakable to humans, so an improvement in adversarial robustness would indicate a more brain-like model.

Examining the above equation, the ϵ value determines the strength of the attack. $\epsilon = 0$ would mean that the attack has no effect and produces images identical to the input images. There is no true upper bound for the ϵ value, but if it gets too large, it distorts the image so much that even humans have difficulty classifying it. Goodfellow et al. [11] used $\epsilon = 0.1$ using CIFAR-10 dataset [16], similar to CIFAR-100, so increments of 0.1 seem reasonable.

FGSM is a white box attack, meaning the attack has all of the architecture information and weights of the model it is trying to deceive. In this method, it does not matter which class the images gets misclassified as, the attack's objective is simply to cause the model to misclassify the images.

To accomplish this, we use an FGSM framework that uses the final trained model from the accuracy experiments (Section 5.1) and runs the FGSM algorithm on that model. Multiple attack strengths were tested on all the variations of models, including

the baseline and shuffled models.

5.3.1 Adversarial Robustness Results

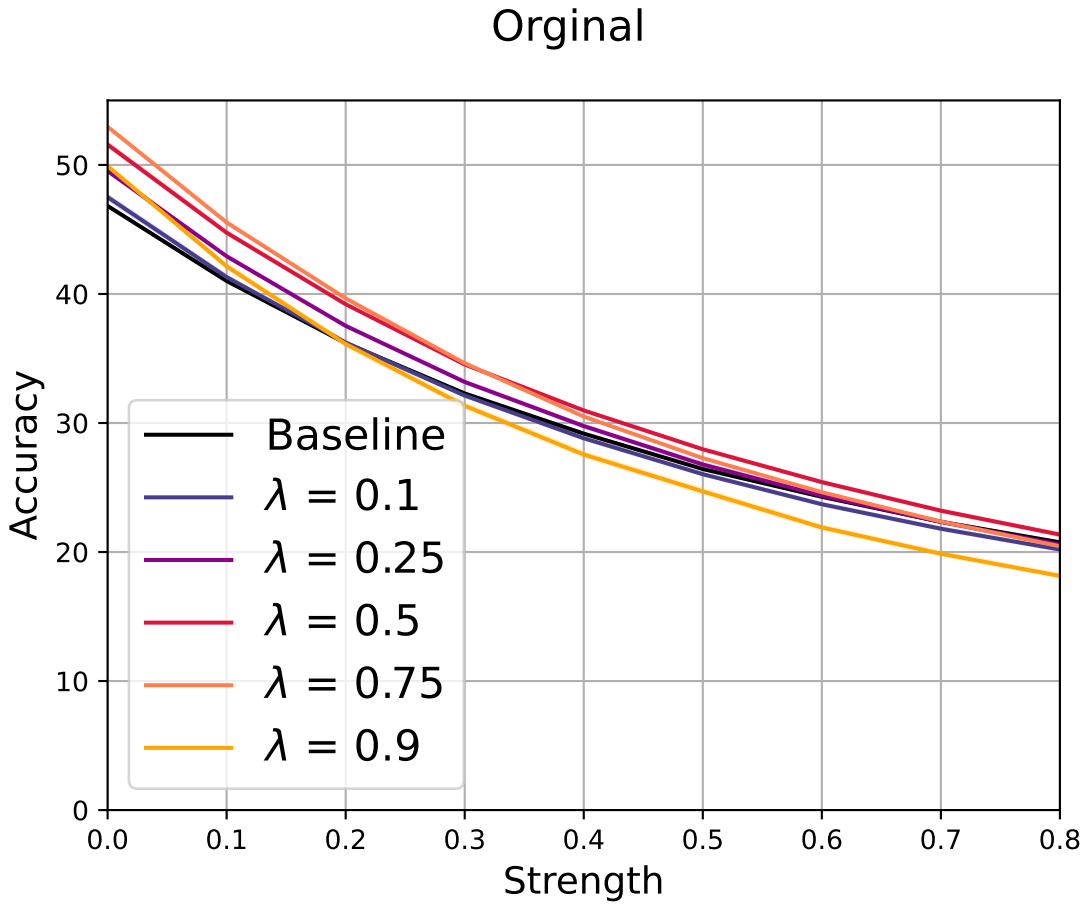
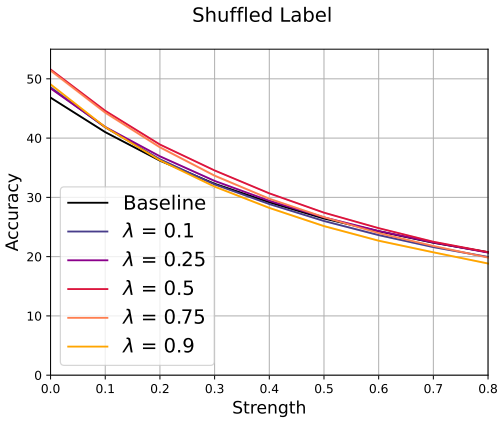
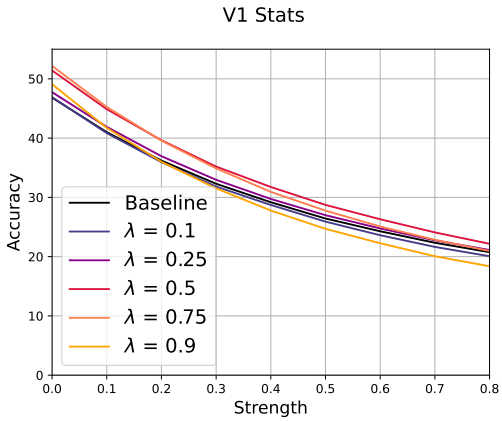


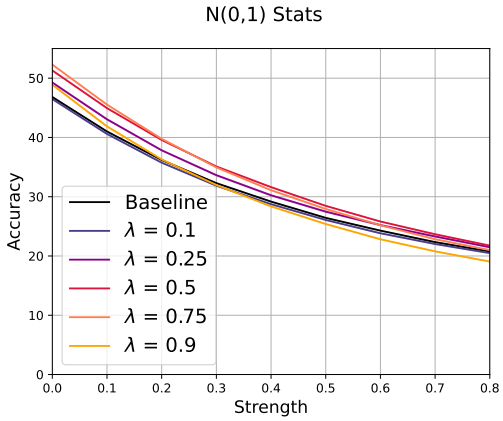
Figure 5.11: A comparison of adversarial robustness for the ND regularized network using different λ values. $\lambda = 0.75$ performs the best until around a strength of .3 when $\lambda = 0.5$ takes over.



(a) Robustness to Adversarial Examples using the ND regularizer and **Shuffled Labels** data.



(b) Robustness to Adversarial Examples using the ND regularizer and **V1 Statistics** data.



(c) Robustness to Adversarial Examples using the ND regularizer and **N(0,1) Statistics** data.

Figure 5.12: A comparison of the λ values for the ND regularized network using different transformed neural data for adversarial robustness. Similar to the original data (Figure 5.11), for all the generated datasets we see that $\lambda = 0.75$ is the most robust to weaker attack strengths. $\lambda = 0.5$ is the most robust for stronger attacks compared to other λ values.

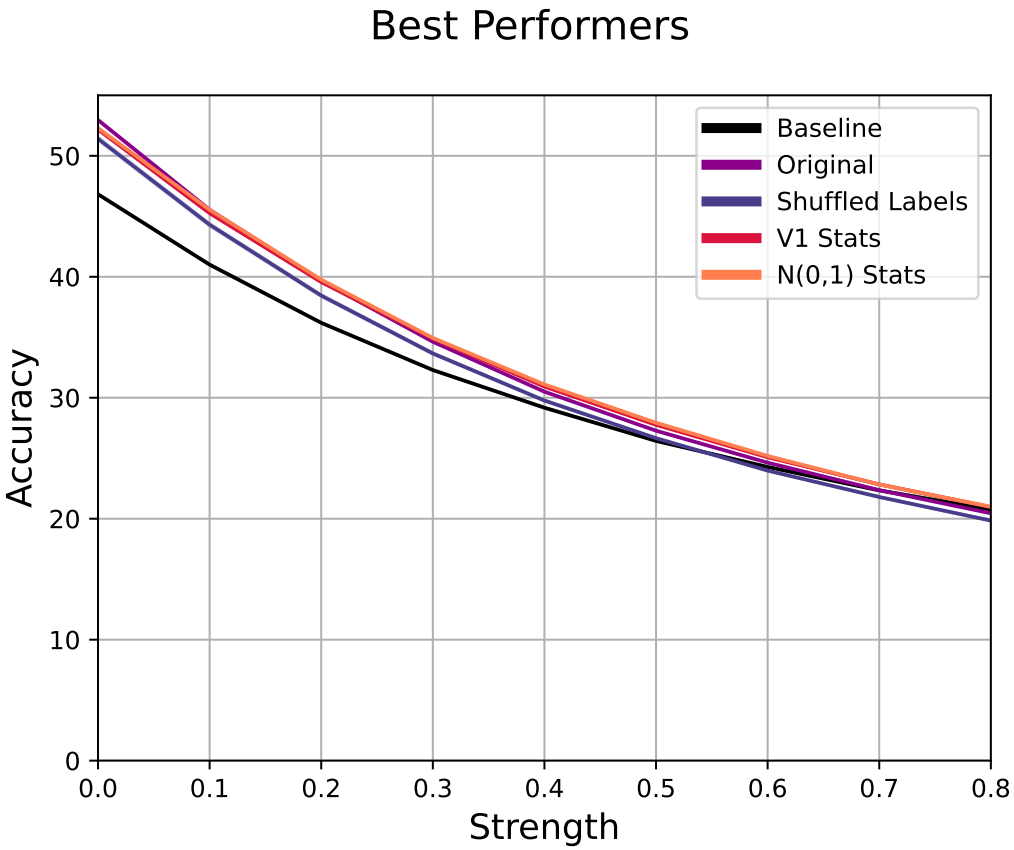


Figure 5.13: A comparison of the different data types at $\lambda = 0.75$. The original data produced the most robust model for low adversarial strengths, but around strength of .4 the **N(0,1) Statistics** data produces the most robust model.

5.3.2 Adversarial Robustness Discussion

To analyze the adversarial robustness experiments in Section 5.3.1, recall that the FGSM [11] adds perturbations that are nearly imperceptible to a human. These attacks use information from the targeted model to add noise to images to deceive the model. From Figure 5.11, compared to the baseline model, moderate strength attacks seem to be better protected against by the ND regularizer using the original data. Interestingly, we find that the larger the λ value, the lower the strength of the attack is when the accuracy preservation declines. $\lambda = .75$ provides the most protection for what we would consider reasonable attack strengths or before the images are unrecognizable to any network.

This pattern is seen again with the other datasets (Figure 5.12). When comparing all 4 datasets at $\lambda = .75$, we find that they all perform very similarly until a strength of .2 when they diverge. Surprisingly after a strength of .2, the **N(0,1) Statistics** dataset and the **V1 Statistics** dataset result in more protection than the original data. However, the accuracy of all 4 datasets' best performing λ values stay well above the baseline's accuracy throughout most of the strengths tested.

5.4 Chapter Conclusion

In this chapter, we established the motivation, techniques, and metrics of performance for our experiments. For each of these experiments, multiple graphs were shown illustrating the results for different λ and dataset settings. A detailed analysis of the graphs and possible interpretations of the results were also given. In the next chapter we will summarize these findings, as well as present our contributions, and discuss the limitations of this work.

Chapter 6

Conclusion and Future Work

Here we will conclude our work by reminding the reader the contributions achieved in this thesis. We will give a brief overview of the content of each chapter as a reminder. Lastly, we discuss the limitations this work has, as well as ideas that could be used to improve it in future works.

6.1 Conclusion

While CNNs are exceptional at categorizing images [25], the usefulness of these algorithms depend on whether humans would agree with their categorizations. That is, the accuracy and the labels of images are in accordance with human judgement. It is clear then, that CNNs should try to achieve outcomes that are similar to human behaviour. However, there is a lack of research that produces algorithms mimicking human behaviour in CNNs. In this thesis, we described a technique that causes a CNN to simulate brain behaviours through the use of a regularizer.

6.1.1 Summary

First in Chapter 2, we discussed the established connection machine learning models have to the brain. We found the conception of CNNs to be heavily influenced by neuroscience findings, and noted that CNNs produce representations that are capable of predicting neural patterns of the brain. We then took special interest in Federer

et al. [9], a study that used RSMs to evaluate the similarity between a network's hidden representations and neural recordings. The similarity measurements were used in the network's cost function to influence the representations to mimic the neural recordings. We also detailed Federer et al.'s successes and shortcomings with this approach.

Second in Chapter 3, we described and examined potential metrics to use in place of Federer et al.'s RSM. CCA [14] was a metric of interest, but we ultimately found more utility in a deep network version of CCA, DCCA [4]. This method would relieve the pairwise, linear, and reweighting limitations in the RSM method, while also being computationally efficient. We then defined CORnetZ [17], our base CNN. Additionally we described the datasets used and outlined the preprocessing procedure.

Thirdly, we introduce our new model in Chapter 4. This model was CORnetZ that shared a node with DCCA. Through the use of a composite cost function $\mathcal{L}$, the model would become more brain-like. Each of the components of the loss function came from either CORnetZ, which classified images, or DCCA, which found the correlation between the shared node's hidden representations and neural recording data. With this, these hidden representations would start to resemble neural recordings.

Next in Chapter 5, we evaluated our model on three criteria. 1) Accuracy - how well it classified images correctly. 2) Super-Class Accuracy - how intuitive the mistakes model made were. 3) Adversarial Robustness - how well the model performed with a dataset designed to trick it. We provided visuals of our results as well as offered possible interpretations of the results for each experiment.

Lastly, here we summarized the thesis and will remind the reader of the major contributions found. Finally, we discuss the limitations of this method as well as possible remedies for these limitations that will be left as future work.

6.1.2 Contributions

Our findings can be summarized as the following,

- The DCCA ND regularizer was able to improve the accuracy of the CORnetZ model on CIFAR-100 data.
- The regularizer produced higher super-class accuracy than the baseline model. Further, the improvement from the accuracy to the super-class accuracy is larger for the regularized model than the baseline model.
- For reasonable strengths of FGSM-produced adversarial examples, the regularized model was more robust.
- The regularized model also outperformed the previous RSM method in accuracy and super-class accuracy.
- Using generated neural data in the DCCA also produced improved results from the baseline and the RSM method in all three experiments (accuracy, super-class accuracy, and adversarial robustness).
- Lastly, we tuned the hyperparameter λ to control the contribution of the DCCA result in the composite cost function. We found that in general, for a weak model like CORnetZ, a λ of 0.5 or 0.75 worked best.

6.2 Limitations and Future Works

Although we found promising results using DCCA with neural data as a regularizer for a CNN, there are some future steps we would like to take.

We would like to try this method on a different CNN. CORnetZ was used mainly because of its structural similarity to the mammalian visual pathway. This structure gave us an obvious place to insert our ND regularizer- the same visual area the data was recorded from (V1). It also allowed us to run proof-of-concept experiments as it was not a top performing model and we would be able to see evident improvement if there was any from our method. However, we would like to see if we could improve a CNN that is already performing remarkably well on an image classification task with

our method. We ran preliminary experiments on ResNet-50 (see Appendix A) and saw promising results. However, more experimentation is needed with a variety of models to really understand the capabilities and limitations of our regularizer.

Because of the availability of brain recording data, we had to use a dataset that was not ideal. The data from Coen-Cagli et al. [6] had a few issues. Firstly, the images were recorded from monkey subjects. Although mammalian brains all have very similar visual pathway structures, we desire our model to mimic human brain behaviour. Also, the images shown to the monkeys were often of objects they would not have encountered or experienced themselves. For example, images of ships or European landmarks were shown to the monkeys. It is difficult to determine whether something so foreign would provoke a meaningful representation in their visual pathway. Secondly, the subjects were anesthetized. Being anesthetized could possibly mean that the neural responses to the stimuli were purely physiological. That is, the monkey's semantic understanding of what is being seen may not have been conveyed in the recordings. This could lead to images of ships and bears that have large white backgrounds to have very similar responses. Thirdly, the dataset is small in two ways. 1) The number of neurons captured is $\mathcal{O}(100)$, which is only a fraction of the range of activity in V1. 2) The number and variety of images are very limited. We would have liked the subjects to see more natural images akin to the classified CIFAR100 dataset. We also believe that 956 images is too small of a dataset to run a machine learning algorithm on. This is evident from having to cycle through the data multiple times for one epoch of the CNN.

6.3 Final Thoughts

In this thesis we introduced a method to bridge the gap between the knowledge that CNNs were inspired by the brain and the knowledge that CNNs develop brain-like representations on their own. Previous works have attempted to capture the connectedness of the brain and CNNs to improve models with great success (Section 2.2).

However, our model achieved better results than these previous works in multiple tasks and with more impressive architectures and/or datasets. With autonomous technology becoming more common in people's daily lives, it is important that this technology use appropriate algorithms. Algorithms that not only achieve high performance (for example in accuracy), but also make more human-like or logical decisions (for example high super-class accuracy or robustness) must be prioritized. As we enable people to use these algorithms to aid in important tasks, it is a researcher's duty to provide models that make more rational decisions, hence making the models safer. The model we created is one step in that direction. Further work can be done to apply this concept to larger and more sophisticated models and with human neural data. Further, more metrics to determine how human-like a model is can be explored, such as other robustness tasks. We hope our work will be sufficient motivation and a solid building block for new research in this area.

Bibliography

[1] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015.

[2] Moustafa Alzantot et al. "Genattack". In: *Proceedings of the Genetic and Evolutionary Computation Conference* (2019).

[3] Laith Alzubaidi et al. "Review of Deep Learning: Concepts, CNN Architectures, challenges, applications, Future Directions". In: *Journal of Big Data* 8.1 (2021).

[4] Galen Andrew et al. "Deep Canonical Correlation Analysis". In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, 2013, pp. 1247–1255.

[5] François Chollet et al. *Keras*. https://keras.io. 2015.

[6] Ruben Coen-Cagli, Adam Kohn, and Odelia Schwartz. "Flexible gating of contextual influences in natural vision". In: *Nature neuroscience* 18.11 (2015), pp. 1648–1655.

[7] Li Deng. "The mnist database of handwritten digit images for machine learning research". In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142.

[8] Karl Pearson F.R.S. "LIII. On lines and planes of closest fit to systems of points in space". In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11 (1901), pp. 559–572.

[9] Callie Federer et al. "Improved object recognition using neural networks trained to mimic the brain's statistical properties". In: *Neural Networks* 131 (2020), pp. 103–114. arXiv: 1905.10679.

[10] Kunihiko Fukushima. "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position". In: *Biological Cybernetics* 36.4 (1980), 193–202.

[11] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and harnessing adversarial examples". In: *arXiv preprint arXiv:1412.6572* (2014).

[12] Kaiming He et al. *Deep residual learning for image recognition*. 2015.

[13] David H Hubel and Torsten N Wiesel. "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex". In: *The Journal of physiology* 160.1 (1962), p. 106.

[14] J. T. Kent Kanti V. Mardia and J. M. Bibby. "Multivariate Analysis". In: (1979).

[15] Nikolaus Kriegeskorte, Marieke Mur, and Peter A Bandettini. "Representational similarity analysis-connecting the branches of systems neuroscience". In: *Frontiers in systems neuroscience* 2 (2008), p. 4.

[16] Alex Krizhevsky. *Learning multiple layers of features from tiny images*. Tech. rep. 2009.

[17] Jonas Kubilius et al. "CORnet: modeling the neural mechanisms of core object recognition". In: *BioRxiv* (2018), p. 408385.

[18] Y. LeCun et al. "Backpropagation applied to handwritten zip code recognition". In: *Neural Computation* 1.4 (1989), 541–551.

[19] Jake Lever, Martin Krzywinski, and Naomi Altman. "Principal component analysis". In: *Nature Methods* 14.7 (2017), 641–642.

[20] José Naranjo Torres et al. "A Review of Convolutional Neural Network Applied to Fruit Image Processing". In: *Applied Sciences* 10 (May 2020), p. 3443.

[21] Iftikhar Naseer et al. "Performance analysis of state-of-the-art CNN architectures for luna16". In: *Sensors* 22.12 (2022), p. 4426.

[22] Josue Nassar et al. "On 1/n neural representation and robustness". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 6211–6222.

[23] Romén Neris et al. "Performance evaluation of state-of-the-art CNN architectures for the on-board processing of remotely sensed images". In: *2021 XXXVI Conference on Design of Circuits and Integrated Systems (DCIS)*. 2021, pp. 1–6.

[24] Shahd Safarani et al. "Towards robust vision by multi-task learning on monkey visual cortex". In: *Advances in Neural Information Processing Systems* 34 (2021).

[25] Jürgen Schmidhuber. "Deep learning in neural networks: An overview". In: *Neural Networks* 61 (2015), 85–117.

[26] Martin Schrimpf et al. "Brain-Score: Which Artificial Neural Network for object recognition is most brain-like?" In: *bioRxiv* (2020).

[27] Martin Schrimpf et al. "Brain-Score: Which Artificial Neural Network for object recognition is most brain-like?" In: *bioRxiv* (2020).

[28] Carsen Stringer et al. "High-dimensional geometry of population responses in visual cortex". In: *Nature* 571.7765 (2019), pp. 361–365.

[29] Ross Wightman, Hoga Touvron, and Hervé Jégou. "Papers with code - resnet strikes back: An improved training procedure in Timm". In: *ResNet strikes back: An improved training procedure in timm — Papers With Code* (2021).

[30] Daniel L. Yamins et al. "Performance-optimized hierarchical models predict neural responses in higher visual cortex". In: *Proceedings of the National Academy of Sciences* 111.23 (2014), 8619–8624.

Appendix A: ResNet-50 Experiments

A.1 Experimental Details

To determine whether our ND regularizer works on a more powerful model than CORnetZ we used ResNet-50 as our CNN.

ResNet-50 has a more elaborate architecture than the previously used CORnetZ. Namely, ResNet-50 contains skip connections. An input $\mathbf{x}$ processed through some layers of a model will give the transformed $\mathbf{h(x)}$. A skip connection is when the input to an intermediate layer is $\mathbf{h(x)} + \mathbf{x}$. Skip connections allows the use of a deeper architecture without degrading the accuracy [12]. The ResNet-50 model has a 50 total layers, often they are grouped in stages (see Figure A.1). In each stage there is 1 convolutional block, and several identity blocks. And in each of these blocks there are a series of layers (for example convolutional layers).

According to Brain Score [27], a composite benchmark for comparing models to the brain, the third block (second identity block) of the second stage forms representations most similar to the V1 area of the brain. For this reason we chose to connect the DCCA to this node. The combined architecture is depicted in Figure A.1.

We use the same hyperparmeter tuning for the DCCA model as in Section 4. We used the architecture described in He et al. [12] along with an image size of 32, batch normalization momentum of 0.9, and an orthogonal kernel initializer. Image augmentation of random horizontal flips and rotations was also used in training. The ResNet-50 experiments were ran for 200 epochs with a learning rate schedule as follows: learning rate = 0.1 for epochs 1-60, learning rate = 0.02 for epochs 61-120, learning rate = 0.004 for epochs 121-180, and learning rate = 0.0008 for epochs 181-200.

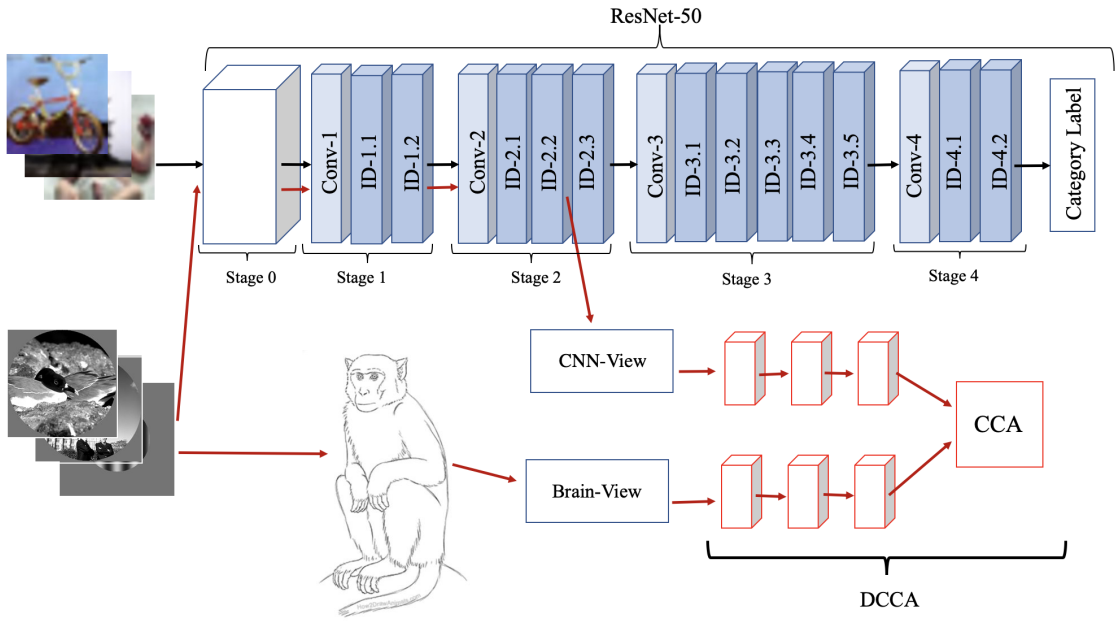


Figure A.1: Architecture of ResNet-50 with the DCCA ND regularizer attached.

Smaller λ values were experimented with as we suspected the ResNet-50 model was already achieving sophisticated and brain-like representations (Yamins et al. [30]) on its own. Three random seeds were run for each $\lambda = (0.01, 0.1, 0.25, 0.5)$.

A.2 Results

We found that the highest accuracy was achieved from using the ND regularizer for the first 10 epochs only, similar to Federer et al. [9]. Note that using our configurations we were unable to achieve state of the art results with the ResNet-50 baseline (Wightman et al. [29]) which we contribute to not using transfer learning and using smaller image sizes.

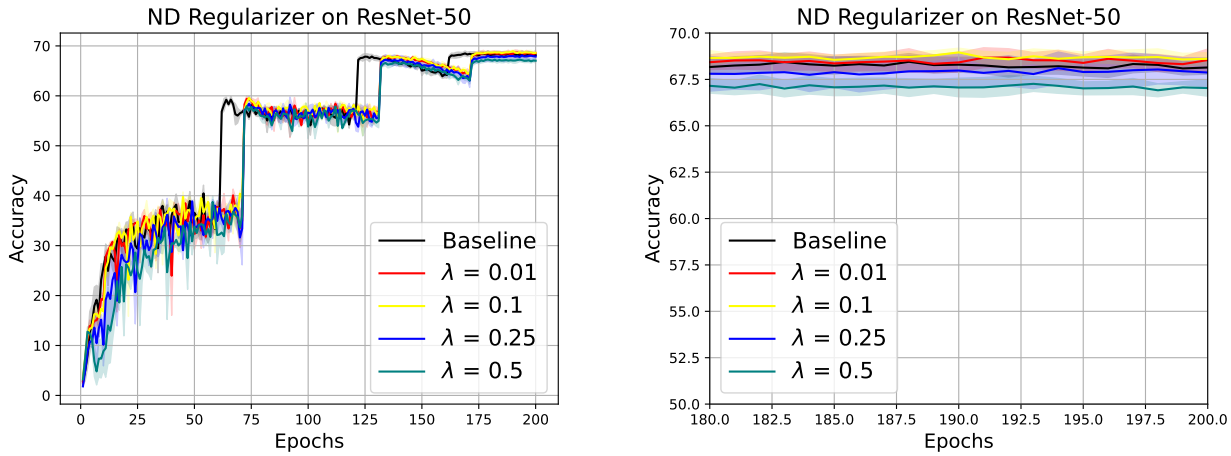


Figure A.2: Accuracy experiment on ResNet-50 with CIFAR-100 data. Both **[a]** and **[b]** are of the same data with axes starting at different points. It can be seen in figure **[b]** that $\lambda = 0.01, 0.1$ consistently achieve higher accuracy than the unregularized model (68.14%). $\lambda = 0.01$ has accuracy of 68.54% on the 200th epoch, $= 0.1$ has accuracy of 68.61% on the 200th epoch. The larger λ values (0.25, 0.5) achieve lower accuracy than the unregularized model.