HOMOGENEOUS VECTOR CAPSULES AND THEIR APPLICATION TO SUFFICIENT AND COMPLETE DATA

A thesis submitted for the degree of Doctor of Philosophy by

Adam D. Byerly

Department of Electrical and Electronic Engineering, Brunel University London

25th May, 2022

Abstract

Capsules (vector-valued neurons) have recently become a more active area of research in neural networks. However, existing formulations have several drawbacks including the large number of trainable parameters that they require as well as the reliance on routing mechanisms between layers of capsules.

The primary aim of this project is to demonstrate the benefits of a new formulation of capsules called Homogeneous Vector Capsules (HVCs) that overcome these drawbacks.

Using HVCs, new state-of-the-art accuracies for the MNIST dataset are established for multiple individual models as well as multiple ensembles.

This work additionally presents a dataset consisting of high-resolution images of 13 micro-PCBs captured in various rotations and perspectives relative to the camera, with each sample labeled for PCB type, rotation category, and perspective categories. Experiments performed and elucidated in this work examine classification accuracy of rotations and perspectives that were not trained on as well as the ability to artificially generate missing rotations and perspectives during training. The results of these experiments include showing that using HVCs is superior to using fully connected layers.

This work also showed that certain training samples are more informative of class membership than others. These samples can be identified prior to training by analyzing their position in reduced dimensional space relative to the classes' centroids in that space. And a definition and calculation both for class density and dataset completeness based on the distribution of data in the reduced dimensional space has been put forth. Experimentation using the dataset completeness calculation shows that those datasets that meet a certain completeness threshold can be trained on a subset of the total dataset, based on each class's density, while improving upon or maintaining validation accuracy.

Acknowledgments

It’s been a long (nearly) five years.

First and foremost, I’d like to thank Dr. Tatiana Kalganova. She took a bit of a risk on me, and I’m hoping she feels that it was worth it. She pushed me hard, but now at the end, I am very thankful that she did. She took a clumsy tinkerer and molded me into a systematic researcher. I especially enjoyed and will always remember the time I realized I had started picking up on some of her speech patterns!

Additionally, I’d like to thank Dr. Maysam Abbod, Dr. Takebumi Itagaki, and Dr. Hongying Meng for giving me their time and effort during my formal reviews during my progression as a doctoral researcher. Their constructive criticism and encouragement were invaluable.

I’d like to thank all of the staff at Brunel University who have helped me along the way, including the College of Engineering, Design and Physical Sciences’s PGR Programmes Administrator, Vicky Maladeni.

At Bradley University, I have many people to thank for many reasons, the subset of those whom I list here by name all had specific and definite contributions, however, there are many more members of the faculty and staff who have been generally supportive and become friends—you know who you are. An appropriate order could not possibly exist for these individuals, so being a computer scientist, I used a random number generator to set the following order: Dr. Steven Dolins, Mr. Scott Williams, Ms. Linda Pizzuti, Dr. Vladimir Uskov, Dr. Yun Wang, Dr. Christopher Jones, Dr. Tachun Lin, Dr. Kelly McConnaughay, Dr. Jason Zaborowski, Dr. Walter Zakahi, and Mr. David Brennan.

Additionally, the reviewers of my work these past (nearly) 5 years deserve credit for helping me to see the ways in which I could improve my work.

I can’t not thank Anthony Grichnik, who knows I don’t dislike not-logic.

And finally, I’d like to the thank the whole of my family and family-in-law. But most especially, my wife Mindy and my kids for making my studies possible, cutting me some slack at times, and picking up so many “extra shifts”, as it were.

With fond memories of Dr. Alexander Uskov

Contents

List of Tables

List of Figures

List of Abbreviations

Abbreviation	Meaning
AGC	Adaptive Gradient Clipping
APE	Absolute Positional Encoding
BN	Batch Normalization
CAD	Computer Aided Design
CFL	Compact Fluorescent Lamp
CIFAR	Canadian Institute for Advanced Research
CNN	Convolutional Neural Network
COIL	Columbia Object Image Library
EMNIST	Extended Modified National Institute of Standards and Technology database
FLOPs	Floating Point Operations per Second
GAN	Generative Adversarial Network
GELU	Gaussian Error Linear Unit
GN	Group Normalization
HVC	Homogeneous Vector Capsule
IC	Integrated Circuit
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
MLP	Multi-Layer Perceptron
MNIST	Modified National Institute of Standards and Technology database
MOE	Mixture of Experts
NLP	Natural Language Processing
NORB	NYU Object Recognition Benchmark
PCB	Printed Circuit Board
PIE	Pose, Illumination, and Expression
PPMCC	Pearson Product-Moment Correlation Coefficient
ReLU	Rectified Linear Unit
RGB	Red, Green, Blue
RPE	Relative Positional Encoding

continued on next page

Abbreviation	Meaning
SAM	Sharpness-Aware Minimization
SGD	Stochastic Gradient Descent
SIFT	Scale Invariant Feature Transform
SVHN	Street View House Number
t-SNE	t-Distributed Stochastic Neighbor Embedding
UMAP	Uniform Manifold Projection and Approximation
USB	Universal Serial Bus
WS	Weight Standardization

List of Datasets

Datasets Used in this Work

Dataset Name	Classes	Image Size	Training Samples	Test Samples
CIFAR-10	10	$32\times32\times3$	50,000	10,000
CIFAR-100	100	$32\times32\times3$	50,000	10,000
MNIST	10	$28\times28\times1$	60,000	10,000
EMNIST-Digits	10	$28\times28\times1$	240,000	40,000
Fashion-MNIST	10	$28\times28\times1$	60,000	10,000
Food-101	101	Varies×Varies×3	75,750	25,250
Imagenette	10	Varies×Varies×3	9,469	3,925
micro-PCB	13	Varies×Varies×3	6,500	1,625

Datasets Referenced in this Work

Dataset Name	Classes	Image Size	Training Samples	Test Samples
ILSVRC 2012	1,000	Varies×Varies×3	1,281,167	100,000
Imagenet-22k	21,841	Varies×Varies×3	14,197,122	N/A*
JFT-3B	30,000	Varies×Varies×3	3,000,000,000	N/A*
JFT-300M	18,291	Varies×Varies×3	300,000,000	N/A*
NORB	5/50**	$96\times96\times1$	48,600	48,600
smallNORB	5/50**	$96\times96\times1$	24,300	24,300
SVHN	10	$32\times32\times3$	73,257	26,032
COIL-20	20	$128\times128\times1$	1,440	N/A*
Multi-PIE	337	Varies×Varies×3	755,370	N/A*

* These datasets have no predefined test set.
** NORB and smallNORB have 5 broad categories of toys, each of which is made up of 10 similar toys.

Chapter 1

Introduction

The general trend in recent neural network research is towards larger and larger model capacities as measured in the number of trainable parameters. Along with this trend towards increased model capacities there exists the trend toward using more and more additional training data. While such research is not without merit, additional research needs to continue to be carried out on smaller scale networks that require less data. There are several reasons for this. First, giga-scale networks are well beyond current hardware capabilities for performing on-device inference with real-time classification requirements. Second, the high cost of resources in terms of time and energy expenditure for the both the data collection for, and training of, giga-scale networks is unnecessary and wasteful for many real world network tasks, which may not require as many classes to be classified or have high accuracy requirements.

In this work, a series of studies will be performed on smaller scale networks and datasets with two main goals. The first goal will be to achieve improved accuracy while simultaneously easing the burden of hyper-parameter tuning for smaller networks. The second goal will be to arrive at an analytical method of measuring the sufficiency of a dataset in terms of the samples required to achieve a target accuracy *prior to training any network on that data*. Special attention will be paid towards achieving these goals using a specific class of neural networks referred to as capsule neural networks.

1.1 Background

In [1], the authors argued that standard convolutional neural networks are "misguided" in their usage of neurons that are composed of singular scalars to summarize their activation. The authors proposed (a) the concept of a "capsule", which is comprised of multiple scalar values and (b) posited that these capsules would be capable of recognizing a "visual entity over a limited domain of viewing conditions and deformations" [1] and that the capsule's members would include both the probability that the entity is present as well as a set of “instantiation parameters” that “may include the precise pose, lighting and deformation relative to the canonical version of that entity” [1]. In their work, they (c) demonstrated that capsules could learn the x and y coordinates of a visual entity and (d) made a convincing case that capsules could learn to identify “any property of an image that we can manipulate in a known way” [1].

Research into capsules did not progress much until a pair of papers were pre-published on arXiv in late 2017. The first of these two papers ([2]) received an especially significant amount of attention, due to the fact that it published results on par with the state-of-the-art for both the standard MNIST [3] and smallNORB [4] datasets using a relatively shallow network in combination with capsules. Additionally, the network described in the first paper was shown to be highly effective at segmenting highly overlapped digits from the MNIST data. Both papers utilized an iterative routing mechanism between layers of capsules. They referred to the method in the first paper as “Dynamic Routing” and used a different method in the second paper based on the Expectation-Maximization algorithm [5]. The architecture described in the second paper ([6]) improved upon the state-of-the-art classification accuracy for smallNORB by 45%.

The architectures described in both papers used two layers of capsules in order to make the final classification and used matrix multiplication between them. In both papers, in addition to learning the weights used in the matrix multiplications using backpropagation, a routing algorithm was employed to iteratively “refine” the weights of the matrices. The authors interpret the first set of capsules as “parts” and the second set as “wholes” and the routing algorithm as a method for finding agreement about which whole is best described by the particular set of parts [1][2].

Both papers published results on relatively small data sets. In both cases this was due to the high computational cost associated with using a routing algorithm. Additionally, the architecture from the first paper requires a large number of parameters per output class (147,456) just for the weights between capsule layers, making datasets with a large number of output classes (like the 1,000 classes in ImageNet) intractable.

Another important thread of neural network research is choosing the best optimization algorithm and its hyperparameters. Stochastic Gradient Descent (SGD) with momentum is simple and effective but requires careful tuning of both the learning rate η and the schedule for decaying that learning rate as training progresses. Though guidance has emerged in the form of rules-of-thumb [7], it is none-the-less true that the choice of the learning rate and rate decay scheme remain a matter of trial-and-error and heavily dependent on the data being trained on. As such, alleviating the need to carefully tune a single learning rate has emerged as an important research area.

The most successful strategy for alleviating the need to carefully tune the learning rate has been to maintain separate learning rates for every trainable parameter and to learn each of these learning rates based on the magnitude of previous gradient updates to those parameters. This method in general is referred to as adaptive gradient descent. Research into this began in earnest with AdaGrad [8] and has continued to be an active area of research up to the present, with the most popular adaptive method currently being Adam [9]. Adaptive methods of gradient descent are popular for several reasons. First, because they adapt a learning rate for every parameter, they are able to learn sparse, yet highly informative features differently than more dense information that may be less predictive. Second, they reduce the need for careful tuning of the learning rate and learning rate decay by allowing the learning rate to be “learned” from the data. And third, they tend to approach a convergence much earlier in the training scheme compared to non-adaptive methods for the same data and network.

Unfortunately, adaptive gradient descent methods have some weaknesses. First, sparsely occurring features that are not highly informative have overweight influence relative to less sparsely occurring features. And second, empirically, they are prone to overfitting and creating a generalization gap between the in-sample and out-of-sample predictions. This has led some researchers to state that the generalization gap of adaptive gradient descent methods is an open problem [10] and has led other researchers to recommend not using adaptive methods at all [7]. Indeed, the best performing convolutional neural networks (CNNs) of the past few years have all used non-adaptive gradient descent methods and hand-tuned learning rate decay schemes [11][12][13][14][15][16].

1.2 Justification

In this work, the prevalent paradigm of using full matrix multiplication between capsule layers and using routing mechanisms to refine the weights between those layers will be challenged. A systematic study of how to form the first layer of capsules is carried out along with proposing and studying using the Hadamard product, rather than matrix multiplication, between the capsule layers. In addition it will be shown that superior classification accuracy can be achieved in the absence of any routing mechanism.

Additionally in this work, it will be shown that this capsule method is superior when using the popular Adam adaptive gradient descent method than when using hand-tuned, non-adaptive gradient descent methods.

Finally, an investigation into an analytical definition of sufficient data is undertaken. In that investigation, a means of identifying *a priori* which training samples improve test accuracy, and which do not, is established. Additionally proposed is a means of allowing the training procedure itself to determine which samples to train with *during the training process*, shortening training time and focusing the training on only those samples that are informative to the classification.

1.3 Contributions

The contribution to the sciences presented in this work are as follows:

1. An analysis of the technologies behind current state of the art in neural networks for image classification, specifically analyzed by both network size, in terms of trainable parameters, and by the amount of training data used. Presented in chapter 3 and the contents of which have been accepted for publication in the *Proceedings of Computing Conference 2022* to be held in London, UK 14–15 July, 2022 [17].

2. A new capsule formulation, which has been named Homogeneous Vector Capsules (HVCs) and demonstration of its effectiveness using two different network architectures and three different datasets of increasing difficulty. Additionally demonstrated is the ability for HVCs to achieve superior results using the Adam adaptive gradient descent optimizer, solving the open problem of the generalization gap of adaptive gradient descent methods [10]. Presented in chapter 4 and the contents of which have been published in *IEEE Access*, vol. 9, pp. 48519–48530, 2021 [18].

3. A neural network design that achieves superior accuracy to the previously best performing capsule network in terms of MNIST validation accuracy, setting a new state of the art for the MNIST dataset, while using dramatically fewer trainable parameters and without using a computationally expensive routing mechanism as compared to the previously best performing capsule network. This network is trained and evaluated specifically on the MNIST dataset to make it directly comparable to the previously best performing capsule network. Presented in chapter 5 and the contents of which have been published in *Neurocomputing*, vol. 463, pp. 545–553, 2021 [19].

4. A study of the ability for both a convolutional neural network using fully connected layers and a convolutional neural network using HVCs to accurately classify a newly created dataset of micro-PCBs[1] in a diversity of rotations and perspectives relative to the presence (or absence) of those rotations and perspectives in the training data. Presented in chapter 6 and the contents of which have been published in *Intelligent Decision Technologies*, vol. 238, pp. 209–219, 2021 [20].

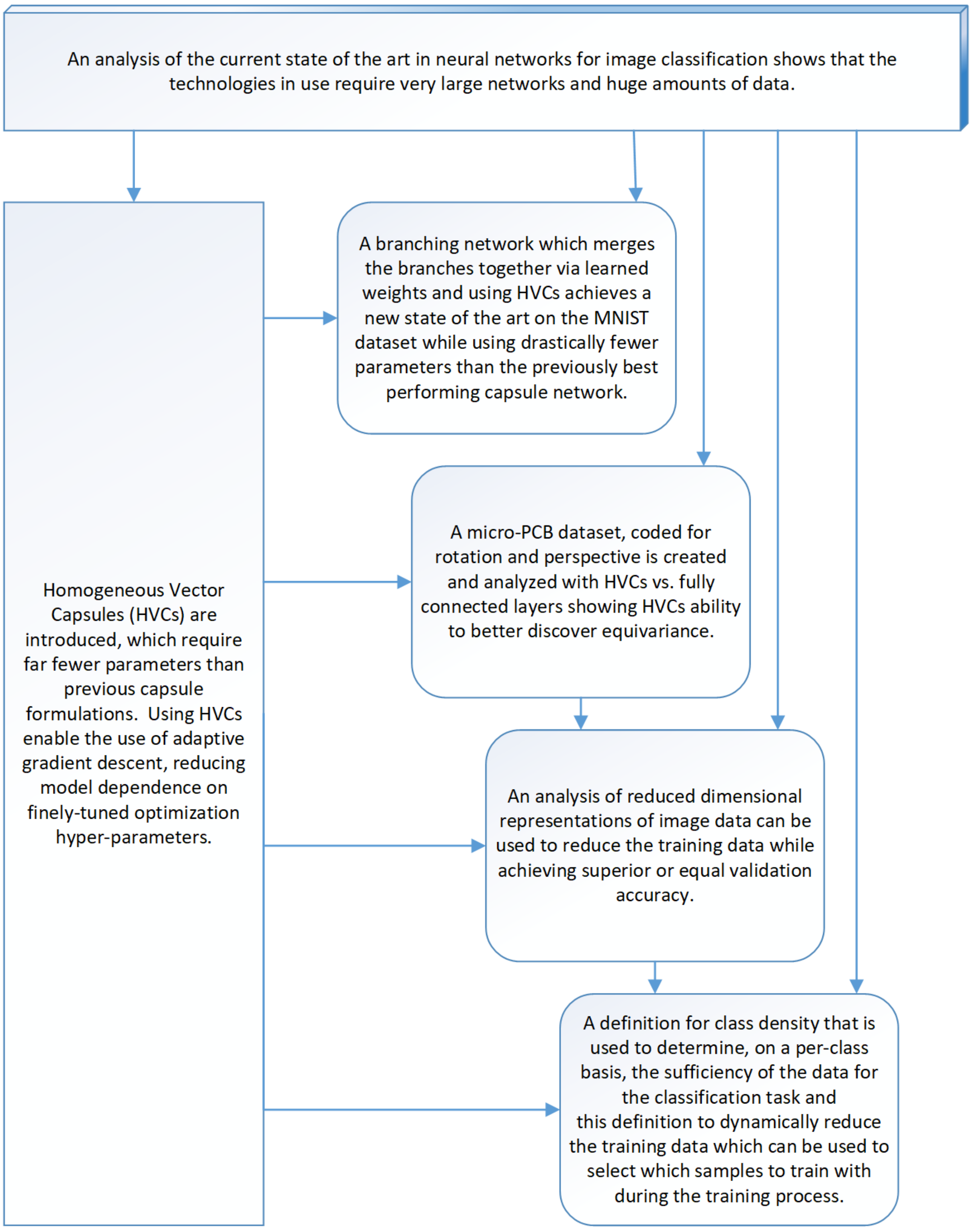
5. A study of the ability to use the distance for training samples from their classes centroids in a reduced dimensional space to indicate their impact on validation accuracy. The results of this study enable researchers to identify *a priori* which

[1]Publicly available at: https://www.kaggle.com/frettapper/micropcb-images

training samples improve test accuracy and which do not. Presented in chapter 7 and the contents of which have been submitted for publication.

6. A definition for class density that is used to determine, on a per-class basis, the sufficiency of the data for the classification task, as well as a study of using this definition to dynamically reduce the training data in any class such that the definition indicates excess data is present. The results of this study enable researchers to allow the training procedure to select which samples to train with *during the training process*, shortening training time and focusing the training on only those samples that are informative to the classification. Additionally presented is a method for calculating an unstructured, high-dimensional dataset's completeness such that when a dataset demonstrates a certain level of completeness (experimentally shown to be > 10), it can be reduced using the density definition. Presented in chapter 8 and the contents of which have been submitted for publication.

Figure 1.1: Visualization of the links between the studies in this work forming the overall research direction of this work.


An analysis of the current state of the art in neural networks for image classification shows that the technologies in use require very large networks and huge amounts of data.
Homogeneous Vector Capsules (HVCs) are introduced, which require far fewer parameters than previous capsule formulations. Using HVCs enable the use of adaptive gradient descent, reducing model dependence on finely-tuned optimization hyper-parameters.
A branching network which merges the branches together via learned weights and using HVCs achieves a new state of the art on the MNIST dataset while using drastically fewer parameters than the previously best performing capsule network.
A micro-PCB dataset, coded for rotation and perspective is created and analyzed with HVCs vs. fully connected layers showing HVCs ability to better discover equivariance.
An analysis of reduced dimensional representations of image data can be used to reduce the training data while achieving superior or equal validation accuracy.
A definition for class density that is used to determine, on a per-class basis, the sufficiency of the data for the classification task and this definition to dynamically reduce the training data which can be used to select which samples to train with during the training process.

Chapter 2

Literature Review

This work will study a combination of factors that, when taken together, provide the basis for achieving high accuracy when using neural networks for classification. In order to achieve a high level of accuracy, a network must be constructed such that it can be easily trained to identify the features that combine to form the classes *equivariantly* and then trained with a sufficient and representative dataset. In terms of structure, this work will study capsule networks and multi-path networks and their ability to be trained with adaptive gradient descent. In terms of sufficient and representative datasets, this work will study data augmentation in general, the ability for data augmentation to supply known missing affine and non-affine transformations as well as perform studies on what samples constitute informative training samples with the goal of arriving at a metric for calculating a dataset's completeness.

2.1 Equivariance

Prior to the relatively recent successes of deep learning, computer vision research emphasized the encoding of knowledge of geometric models of the objects to be recognized. In the absence of existing CAD models, a computer vision engineer of this era would need to devise some method of intuiting objects' geometric properties from the data available and manually inputting the information into the target system [21][22]. Obviously, this is not a scalable solution. In the mid-1990s researchers turned their attention to the mechanisms employed by biological vision systems and began to focus on the appearance of objects in the 3-D world as projected onto 2 dimensions in order to both classify and estimate the pose of objects. In 1995, researchers put forth a method for capturing objects of interest from a multitude of positions relative to the camera and, importantly, publicly released their dataset that included 20 objects captured in 90 different poses with 5 positions for the lighting source for a total of 450 images for each of the 20 objects [23]. This dataset would come to be known as the COIL-20 dataset.

Early methods for attempting to classify objects in varying 2-D projections of the 3-D world used classic computer vision methods for identifying edges and contours. For example, in [24], the authors measured the similarity of objects after identifying correspondences between images and then using those correspondences to construct an estimated aligning transform. They were able to achieve an impressive 2.4% error rate on the COIL-20 dataset when trained from an average of just 4 of the 2-D projections of each 3-D object.

Several years later, researchers generated a larger more difficult dataset, the NORB dataset (NYU Object Recognition Benchmark) [4]. This dataset included higher resolution images than COIL-20 and contained not just variations in the objects' rotations, but also their perspectives relative to the camera.

Also, worthy of note is the Multi-PIE dataset [25], which is composed, not of objects, but of faces, from 337 different subjects from 15 different viewpoints and having 6 different expressions, for the purposes of facial recognition. Recognizing a specific face among a group of faces, as opposed to recognizing a four-legged animal among cars and airplanes, is a significant challenge because, assuming a lack of deformity, all faces are made up of the same sub-parts in similar locations (e.g., two eyes above a nose, which is above a mouth).

In the past several years, the vocabulary in the field has matured and turned to describing the relationships between features of 2-D projections of 3-D objects as being *equivariant*, a term borrowed from the representation theory of finite groups. *Translation equivariance* in convolutional neural networks (CNNs), for example, is demonstrated when shifting an image that is fed into a network produces the same result as shifting the output feature maps of the original image in the same direction. For computer vision applications, the ideal would be for all possible 2-D projections of a 3-D object to possess a calculable or learnable representation of their equivariant properties. One principal area of research involved in achieving better equivariance involves novel neural network architectural elements designed to facilitate it [26][27][28][29].

Invariance is the special case of equivariance wherein the transformation is a null transformation. CNNs are especially adept at successfully identifying features in a translationally invariant manner. In [1], the authors critiqued this as being an undesirable property. They noted that individual features of a larger image are not translationally invariant to one another, but rather possess a specific non-null translational equivariance relative to one another (such as in a person's face, the position of the eyes relative to the nose and the nose relative to the mouth). They also noted that the quite successful computer vision system SIFT [30], used hand-engineered feature detectors that were 128-dimensional vectors called keypoint descriptors. In that work, they put forth a novel neural network architectural element they called capsules that would act much like the scalar-valued neuron, but be vector valued like the keypoint descriptors of SIFT. Using backpropagation, the capsules could then, if exposed to the same objects from different 2-D projections, implicitly learn a vector of values for the equivariance of the feature that capsule detected. In [6], the authors applied capsules to the smallNORB dataset (which is a subset of NORB) and achieved a new state-of-the-art test error rate.

Chapter 6 presents a case study of equivariance using a dataset specifically crafted for this work. The subject matter of the dataset is small, printed circuit boards (micro-PCBs), which like human faces, have features that exist equivariantly to one another. The experiments in chapter 6 include CNNs with fully connected layers and CNNs with homogeneous vector capsules.

2.2 Capsule Networks

Inspired by the work of [31], [32] put forth a neural network architecture similar to that used by [2] in that it utilized vector neurons, rather than scalar neurons, which shared common inputs and outputs. As their work was inspired by the physiology of primate brains, they characterized the structure as minicolumns, the term used for the analogous structure in primate brains. It is noteworthy that their architecture did not use any analog to the routing mechanism employed by [2] and [6]. While performing comparably with traditional CNNs on the MNIST dataset, it performed worse than the architecture employed by [2].

A comparison the effects of various forms of image degradation (additive white gaussian noise, salt and pepper noise, etc.) on MobileNet [33], VGG16 & VGG19 [11], Inception v3 [13], and CapsNet [2] was performed by [34] and the authors found that CapsNet was far more robust against the degradation methods they tested than any of the others. They hypothesize that this is not only due to the presence of the capsule neurons and/or dynamic routing, but also due to the shallower nature of CapsNet, having gone through fewer layers of convolutions.

The first application of capsule networks in the bioinformatics domain applied a capsule network to the task of protein gamma-turn prediction [35]. Novel to their experiments is that they prepended the capsules portion of the network with an inception block ala [13] rather than a simple convolution. They achieved a new state-of-the-art performance on the GT320 benchmark [36] for gamma-turn prediction with an MCC (Matthew Correlation Coefficients—the metric used for this task) of 0.45, beating the previous state-of-the-art of 0.38.

Shortly after the CapsNet architecture was pre-published, [37] ventured to apply it to more complex datasets than MNIST—Fashion MNIST [38], SVHN [39], and CIFAR-10 [40]. Additionally, they experimented with a greater range of affine deformations than the small amount of translation used in the original experiments. Their conclusion was that the CapsNet architecture is “unlikely to work on other classification tasks, let alone machine learning tasks in general”. They also concluded that the design was “not making full use of routing to encode” the spatial relationships between the components of the objects the network was classifying. They hypothesized that a neural network, as opposed to a routing algorithm, would better accomplish the goal of reweighting the coefficients used to determine the agreement between capsule layers. This method was experimented with by [41], though they were unable to produce any significant results. Additionally, they hypothesized that for data more complex than MNIST, deeper networks may be required.

2.3 Multi-Path Networks

Many of the best performing convolutional neural networks (CNNs) of the past several years have explored multiple paths from input to classification [12][13][14][42][43][44]. The idea behind multiple path designs is to enable one or more of the following to contribute to the final classification: (a) different levels of abstraction, (b) different effective receptive fields, and (c) valuable information learned early to flow more easily to the classification stage.

In [14] (and subsequent extensions [45][46][47][48]) the authors added extra paths through the network with *residual blocks* which are meta-layers that contained one or more convolutional operations as well as a “skip connection” that allowed information learned earlier in the network to skip over the convolutional operations. Similarly, in [12] and [13], the authors presented a network architecture that made heavy use of *inception blocks*, which are meta-layers that branch from a previous layer into anywhere from 3 to 6 branches of varying layers of convolutions. Then the branches were merged back together by concatenating the filters of those branches. Let n be the average number of branches of different length (in terms of successive convolutions) and m be the number of successive inception blocks. Then $n \times m$ effective receptive fields and levels of abstraction are present at the output of the final inception block. Additionally, the designs presented in both of these papers included two output stems (one branching out before going through additional inception blocks and the other after all inception blocks) each producing classification predictions. These classifications were combined via static weighting to produce the final prediction.

In contrast to the aforementioned work, chapter 5 presents a network design that produces 3 output stems, each coming after a different number of convolutions, and *thus representing different effective receptive fields and levels of abstraction*. Chapter 5 details experiments that include statically weighted combinations as in [12] and [13] as well as investigating learning the branch weights *simultaneously with all of the other* *network parameters via backpropagation*. Again, in contrast to the aforementioned work, in these experiments, each of the separate classifications were performed with capsules rather than simple fully connected layers.

2.4 Adaptive Gradient Descent

One of the best performing CNNs to be published in the past several years is Inception v3 [13]. To train their architecture, they used the RMSProp[1] optimizer, which is indeed designed to be an adaptive gradient descent method. RMSProp adapts each parameter in the model using Equation 2.1:

$$\frac{1}{\sqrt{E[g^2] + \epsilon}} \tag{2.1}$$

$E[g^2]$ is the exponential moving average of the past squared gradients for the parameter and the intended purpose of the ϵ parameter is to provide numeric stability by mitigating the danger of division by zero, and thus implementations default this value to 1×10^{-10}, which would create a range of possible values for the per-parameter adaptive term of 0 to 1×10^{6}. By using a value of 1.0 when training Inception v3, they limit this range to 0 to 1, thus setting an upper bound five orders of magnitude less than intended for this term. While still technically adapting each parameter, the range of adaptation is so dampened that RMSProp with a 1.0 ϵ should be considered as quasi-adaptive at best. As such, this work agrees with Chen and Gu [10] that effectively utilizing (truly) adaptive gradient descent methods with convolutional neural networks remains an open problem relative to Inception v3.

The Adam optimizer [9] has an analogous per-parameter adaptive term for each of the past squared gradients shown in Equation 2.2 (in addition to another term not relevant to this discussion for past gradients that gives Adam a momentum-like behavior):

$$\frac{1}{\sqrt{\hat{v}_t} + \epsilon} \tag{2.2}$$

$\hat{v}_t$ is the bias corrected exponential moving average of the past squared gradients for the parameter. Here again, the Adam optimizer employs the use of an ϵ that implementations default to 1×10^{-10}. Since in Adam, the ϵ is moved out from underneath the radical, Adam is able to adapt each parameter by five orders of magnitude more than RMSProp (with a range of 0 to 1×10^{10}).

[1]RMSProp is an unpublished, adaptive learning rate method introduced by Geoffrey Hinton in Lecture 6e of a now no longer available Coursera course. See: http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6pdf

2.5 Data Augmentation

In [3], the authors experimented with the number of training samples used in the training of the architecture of their convolutional neural network design. Not surprisingly, they discovered that a larger number of training samples leads to a better test accuracy which is indicative of the network "generalizing" better. The theoretical basis for this has been understood in the context of neural networks since [49]. In [3], the authors saw that while only having 60,000 training samples available, that the trajectory of the accuracy their network achieved was trending towards better accuracy in the presence of additional training samples. The authors sought to verify this hypothesis and thus, "artificially generated more training examples by randomly distorting the original training images" [3]. The authors found that by applying combinations of planar affine transformations to the original training samples (including translation, scaling, and shearing) to generate new samples, that the accuracy of their network did indeed increase. This method for generating new samples has proven to be so effective that it is considered standard practice to apply it whenever training a convolutional neural network for more than an exceedingly small number of epochs and has come to be referred to generally as data augmentation. Standard data augmentation transformations for image data have since been expanded to include generally any affine transformation, including rotation and reflection. Additionally, non-affine transformations are sometimes considered, such as the elastic deformation introduced by [50].

Recent research into data augmentation strategies has been focused in two areas: dynamic determination of the augmentations to be performed and where the augmentation should be performed, i.e., in data-space or feature-space. Data-space augmentation involves transforming the actual input data before feeding it into the neural network. This includes all of the techniques thus far discussed. Feature-space augmentation, in contrast, involves analyzing the distribution of features detected inside the network for the classes the network classifies and performing linear interpolation between the k nearest neighbors in that space and adding "synthetic" data points at those locations for that class [51]. Experiments performed by [52] comparing both data-space and feature-space augmentation using the MNIST dataset demonstrated that while both methods achieved marginally higher test set accuracy, data-space augmentation did more so. The conclusion they drew from this is that in cases where label-preserving transformations in data-space are known, data-space augmentation should be preferred. However, if a given application does not have a well understood relationship between a class's label and what data-space augmentation methods would preserve that label, then feature-space data augmentation can still be used, as it is application independent.

Given this, it seems natural to search for methods of dynamically discovering label-preserving methods of data-space augmentation. Research into this has been especially active in recent years. [53] approach this by formulating this as a search problem over the multi-set of common transformation operations. The augmentation methods they consider include the affine transformations of shearing, translation, rotation as well as a slew of RGB channel manipulations, including adjustments to posterization, contrast, and brightness. They quantized the probability of applying a transformation and the range of the transformations so that the entire search space, including what transformations to apply, is discretized, and thus allowed them to use a discrete search algorithm. By limiting the maximum number of transformations to be applied to 5, they created a search space with a size of approximately 2.9×10^{32}. Clearly, an exhaustive search was not an option, so they employed a search algorithm using Reinforcement Learning. The reward signal for their algorithm was how well the policy generalized to a classifier network that used the policy in question.

Another direction dynamic data augmentation strategy discovery has taken recently involves creating synthetic data points but in data-space, rather than feature-space [51]. This direction differs from the previously mentioned data-space methods in that it does not involve a preselected group of transformations, but rather, seeks to create new data points "in between" existing data points. [54] and [55] approach this by employing the use of generative adversarial networks (GANs) to generate new images from combinations of inter-class images or by applying various external "styles" to the dataset images. [56] also employed the use of an additional network beyond the classifier network. In their approach, a generative network is prepended to the classifier network. The generative network accepts multiple inter-class samples and generates a new sample from them, and then feeds the sample into the classifier network. The novel idea they employed involved allowing the backpropagation from the classifier network to continue into the generative network, causing the generative network to "converge to the best choices to train [the classifier network] for that specific task, and at the same time, it is controlled by [a loss function] in a way that ensures that the outputs are similar to other members of its class."

In [57], the authors employed three forms of data augmentation: random translation, random horizontal flipping, and altering the RGB channel intensities. Given the success of the network they designed, the methods they used have become the standard starting point for most data augmentation strategies since. Their network was designed to work on the ImageNet dataset. In the case of MNIST images, of the three methods they used, only random translation applies. This is because horizontal flipping is not a label-preserving transformation for the Hindu-Arabic digits. For example, horizontally flipping an image of a 5 would result in an image that should be classified as a 2. Additionally, altering RGB channel intensity does not apply because the MNIST digits are greyscale. Thus, leaving only translation from the methods they employed as both label-preserving and applicable to the MNIST dataset.

In [2], the authors state that the data augmentation strategy they used was limited to shifting the training images randomly by up to 2 pixels in either or both directions, but otherwise no other affine transformations or deformations were performed. This is consistent with the data augmentation methods used by [57] when applied to the MNIST dataset. The limit of only 2 pixels for the translation is hypothesized to be chosen to ensure that the translation is label-preserving. As the MNIST training data has varying margins of non-digit space in the available 28×28-pixel canvas, using more than 2 pixels randomly, would be to risk cutting off part of the digit and effectively changing the class of the image. For example, a 7 that was shifted too far left could become more appropriately classed as a 1, or an 8 or 9 shifted far enough down could be more appropriately classed as a zero.

The highly structured nature of the MNIST training data allows for an algorithmic analysis of any image that will provide the translation range available for that specific image that will be guaranteed to be label-preserving. Figure 2.1 shows an example of an MNIST training image that has an asymmetric translation range that, as long as any translations are performed such that the digit part of the image is not moved by more pixels than are present in the margin, will be label-preserving. In other words, the specific training example shown in Figure 2.1 could be shifted by up to 8 pixels to the left or 4 to the right and up to 5 up or 3 down, and after doing so, all of the pixels belonging to the actual digit will still be in the resulting translated image.

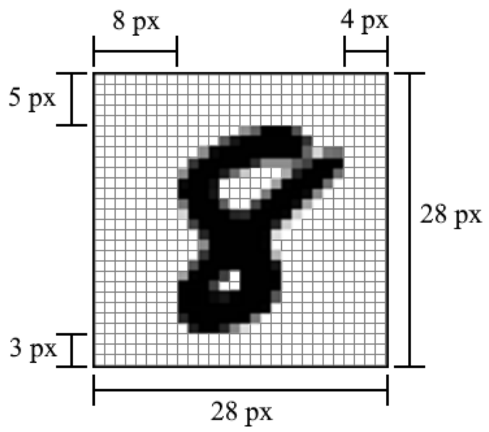


Figure 2.1: Example MNIST digit w/annotated margins.

2.6 PCB Datasets

Printed Circuit Boards (PCBs) are like the human face in that, for a given PCB, the individual elements (such as capacitors, resistors, and integrated circuits (ICs)) are not present invariantly relative to one another, but rather at very specific locations relative to one another. Compared to the human face, even on small PCBs, there are a greater number of features with greater similarity to one another (some modern small surface-mounted capacitors and resistors are nearly indistinguishable from one another, whereas eyes, noses, and mouths are quite distinctive from one another).

In [58], the authors put forth a small, medium-resolution dataset of a variety of PCBs and applied SIFT and other similar methods to the task of locating and categorizing the components on the board (i.e., identifying resistors vs. capacitors vs. ICs). This dataset has a very small number of images and annotates the category and locations of the components on the PCBs. Likewise, work by [59], [60], and [61] introduced similar datasets at higher resolutions. The unifying characteristic of these datasets is that they all captured relatively few images of any one PCB, all from a completely neutral perspective (i.e., the camera directly above the PCB), and that the intended purpose was for classification and categorization of the many sub-components on the PCBs. Other work has further focused on detecting specifically only surface mounted components [62], through-hole components [63], on-board printed text [64], or manufacturing defects [65]. Again, all of these were focused on analysis of the individual sub-components of the PCBs. The motivation in all of these cases is to facilitate automated analysis to optimize recycling practices.

There are indeed a truly massive number of different PCBs that do or could exist. As such, the analysis of sub-components does make sense for the general case—so that knowledge of the specific make and model of a PCB is not a prerequisite to knowing the sub-components used in it. However, if the distribution of PCB makes and models follows a Pareto distribution (a presumed conjecture not demonstrated here), then it stands to reason that a small number of PCB makes and models could be far more common. Assuming this hypothesis, a candidate for the types of PCBs that would be more common are those that are general-purpose, affordable, and small. This work refers to this class of PCBs as micro-PCBs. In those cases, accurately classifying the make and model of the PCB, rather than attempting to locate and categorize every sub-component, could, through a bill of materials, accurately identify *all* of the sub-components on such PCBs. In this case, robustness to environmental factors during image acquisition through a wider range of rotations and perspectives becomes an enabling factor for implementations.

2.7 Dataset Density and Completeness

k-Nearest Neighbor (kNN) [66] and other so called “instance-based” methods are also sometimes called “memory-based” because building the model from the training data involves storing the training instances. During evaluation, the sample being evaluated is compared to each stored instance in order to find the best classification for it. Thus, evaluation is an $O(n)$ operation, where n is the number of stored instances. Because the computation cost during evaluation is so high, these algorithms are also sometimes called “lazy learners”.

Prior to the relatively recent pivot to large, over-parameterized neural networks for classification, these instance-based methods were some of the most widely used for classification. As such, a fair amount of research went into reducing the number of training samples that had to be “memorized” both to ease the memory requirements and to speed up evaluation. Early methods for doing this include Condensed Nearest Neighbor (CNN) [67], Selective Nearest Neighbor (SNN) [68], and Edited Nearest Neighbor (ENN) [69]. More recently a slew of other methods have been investigated [70][71][72][73][74].

However, after the success of AlexNet [57], research into classification methods pivoted to large, over-parameterized neural networks. Shortly after that, some research into reducing training dataset size was conducted [75], however, the focus was on acceptable loss of accuracy, rather than maintaining or improving accuracy.

More recently, the focus has shifted to adding training data beyond datasets’ canonical training set in order to improve accuracy. This trend is elucidated in detail in chapter 3.

Data density is of interest in the classification domain mostly in cases where there exists a class imbalance problem, which can be quite common in many real-world data collection scenarios (for example credit card fraud detection [76][77]). In this context, the interest is in increasing the density of the data in the minority class via synthetic sample creation [51] or in under-sampling the majority class [78]. In [79] and [80], the authors showed that generating synthetic minority class samples from data near the boundaries of the classes produced superior results as compared to completely randomly sampling from the entire minority class. In the clustering domain, data density has been analyzed to produce more intelligent clustering algorithms that do not require the prior selection of the number of clusters (as is required for k-means clustering) [81].

Studies in data density are replete in the literature for low-dimensional data, such as in those cited in the prior paragraph. However, data density is less studied in high-dimensional data. This is most likely due to the fact that distance measurements in high-dimensional space produce near uniform distances [82], a phenomenon colloquially referred to as the *curse of dimensionality*.

As has been put forth and stands to reason, data quality is context dependent [83]. Most of the context in the existing literature is on highly-structured, often relational, data for which there are known domains and constraints for well-defined groups (tuples) of atomic values [84][85]. Further, there is no universally agreed upon definition of data quality, let alone a universal method for measuring it. However, there is widespread agreement on some things, such as having high levels of the following attributes:

timeliness, accuracy, consistency, and completeness [86].

2.8 Summary

Capsule Networks

This work agrees with these hypotheses put forth in [37] and [41] and chapter 4 elucidates experiments that, (1) use a neural network approach, rather than a routing approach, when transforming between capsule layers, and (2) use deeper networks for classifying image data that is much more complex than MNIST.

Multi-Path Networks

An analysis of the existing literature shows that of the many branching methods explored, those that produced multiple final classifications merged those classifications via static weighting, which presupposes the relative importance of each output. Chapter 5 includes and compares the results of both statically weighting the classification branches and learning the weights of the classification branches via backpropagation.

Adaptive Gradient Descent

Chapter 4 details experiments comparing RMSProp with an ϵ set to 1.0 with two different learning rate schedules and the Adam optimizer with and without a learning rate schedule. These experiments are performed on both Inception v3 and a simple monolithic CNN on three different datasets of increasing difficulty.

Data Augmentation

Data augmentation is used in all experiments in this work. For the experiments detailed in chapter 4 the data augmentation is the same as was used in [13]. The experiments detailed in chapter 5 take advantage of the highly structured nature of the MNIST dataset and employ a range of data augmentation techniques informed by that structure. The experiments in chapter 6 are tailored to the micro-PCB dataset detailed in that chapter and take advantage of the fact that the images in that dataset are coded for rotation and perspective, and a detailed set of experiments on the ability of data augmentation to supply missing rotations and perspectives is performed. The experiments in chapter 7 and chapter 8 use the same data augmentation as was used for the experiments in chapter 4 or chapter 5 depending on the dataset being used.

PCB Datasets

Chapter 6 details a set of experiments aimed at classifying micro-PCBs from a variety of rotations and perspectives and performs a study on classification accuracy when the test dataset includes rotations and perspectives not seen during training.

Dataset Density and Completeness

In chapter 7, in order to reclaim meaningful differences when measuring the distances between samples, Uniform Manifold Projection and Approximation (UMAP) [87] is used to reduce high-dimensional image data to 3 dimensions. Then, a study of which samples best inform the classification task, relative to each class's centroid in the reduced dimensional space, is performed.

In chapter 8, the focus on quality will be in the sense of completeness. Only a dataset comprised of low-dimensional data with very limited domains could ever be considered complete in the strictest definition of the term. For the high-dimensional, unstructured data studied in this work, completeness will be demonstrated in chapter 8 by removing data from the dataset, based on a target density threshold, while maintaining classification accuracy.

Chapter 3

Exploding Dataset Sizes and the Current State of the Art in Image Classification

In [88], the authors pose the question: “Are we done with ImageNet?”. They ask if the recent progress on the Imagenet-1K [89] evaluation benchmark is continuing improvement on generalization or the result of the deep learning for image classification community learning some latent properties of the labeling procedure. The latter possibility is interesting, and in their work, they do some good analysis and provide a better set of labels, which should be considered for use going forward. However, for now, the original labels remain the standard benchmark and the means by which comparisons among the best models are made. Papers with Code [90] has become the best-known record of the state-of-the-art methods for all types of deep learning tasks, including image classification. On Papers with Code, in the case of Imagenet, the performance is ranked by top-1 accuracy achieved. This chapter examines the technologies behind the top 40 best ranked accuracies, which are reported in 22 papers (some papers present multiple models which rank multiple times).

Image classification, until very recently, has been the domain of convolutional neural networks. However, since 2020, transformer networks have seen great success for this task, as well. The analysis in this chapter will be organized based on whether the networks being discussed use (a) transformers, (b) convolutions (all of which use specifically EfficientNet [91] networks), (c) a hybrid of the transformers and convolutions, or (d) neither. Additionally, 6 of the papers (accounting for 10 of the top 40 results) attribute their results to innovations related to the training procedure, rather than on any network architecture. All 6 of these papers used convolutional neural networks, 5 of which used EfficientNet networks and 1 of which used ResNet ([92]) networks.

The work presented in this chapter has been accepted for publication in the *Proceedings of Computing Conference 2022* to be held in London, UK 14–15 July, 2022 [17].

3.1 Transformer Networks

Since [93] and later [94], transformer networks have been dominating NLP deep learning tasks. As such, computer vision researchers have been looking into ways to take that success and transfer it to their domain. They have done so with a fair amount of success, with the caveat that such success in most cases has required unprecedentedly large networks with unprecedentedly large sets of additional training data. The fact that this chapter includes non-transformer-based networks trained without additional training data that are competitive with these networks suggest that it remains an open question whether or not transformer-based networks will entirely supplant convolutional neural networks in computer vision tasks. See Table 3.1 for a comparison of the transformer-based models reviewed here.

In [99], the authors introduce ViT. ViT is currently the vision transformer network that most recent transformer networks compare themselves to or use as a basis for their designs. Inspired by the success of transformers applied to the NLP domain, the authors endeavored to create a network for the vision domain out of transformers *sans* convolutions entirely, and in their own words "with the fewest possible modifications" to existing transformer designs. The authors note that applying self-attention naively to entire images means attending every pixel to every other pixel and thus represents a quadratic complexity relative to the image's size, which would not scale well to usable input sizes. The insight they leveraged was that 16×16 patches of an image could be treated much like words are treated in NLP applications. Prior attempts at fully transformer-based networks [107] failed to achieve competitive results on ImageNet-1k evaluation accuracies due to having not attempted to scale up the networks parameters and additional training data. Again, in their own words, the authors discovered that "large scale training trumps inductive bias"—the inductive bias being that which is introduced by convolutions.

In [95], the authors conducted a systematic study of the relationships between data size, compute budget, and achieved accuracy across a spectrum of ViT models [99]. Unsurprisingly, they discovered that bigger models with larger compute budgets result in higher accuracies, with the caveat that there exists sufficient data to train the model. In the largest models they studied, even 300M samples was insufficient to saturate the models' achievable accuracy. Additionally, they found that the larger models were more *sample efficient*, meaning they achieve the same accuracy as smaller models after training for fewer steps. Another important observation that the authors made was that for more than two orders of magnitude, compute budget and accuracy followed a power-law, and at the high end of the compute budget, the largest models were not tending toward perfect accuracy, suggesting that a model with infinite capacity would achieve less than perfect accuracy. The authors noted that similar effects have been

Table 3.1: Transformer Networks

Rank	top-1 Acc.	# of Params.	Addt'l. Training Samples	Paper Title
2	90.45%	1843M	3B	Scaling Vision Transformers [95]
4	90.35%	14700M	3B	Scaling Vision with Sparse Mixture of Experts [96]
8	88.87%	460M	300M	TokenLearner: What Can 8 Learned Tokens Do for Images and Videos? [97]
9	88.64%	480M	1.8B	Scaling Up Visual and Vision-Language Representation Learning With Noisy Text Supervision [98]
11	88.55%	632M	300M	An Image is Worth 16$\times$16 Words: Transformers for Image Recognition at Scale [99]
14	88.36%	7200M	300M	Scaling Vision with Sparse Mixture of Experts [96]
15	88.23%	2700M	300M	Scaling Vision with Sparse Mixture of Experts [96]
16	88.08%	656M	300M	Scaling Vision with Sparse Mixture of Experts [96]
18	87.76%	307M	300M	An Image is Worth 16$\times$16 Words: Transformers for Image Recognition at Scale [99]
20	87.54%	928M	14M	Big Transfer (BiT): General Visual Representation Learning [100]
21	87.5%	173M	14M	CSWin Transformer: A General Vision Transformer Backbone with Cross-Shaped Windows [101]
22	87.41%	3400M	300M	Scaling Vision with Sparse Mixture of Experts [96]
24	87.3%	197M	14M	Swin Transformer: Hierarchical Vision Transformer using Shifted Windows [102]
26	87.1%	296M	0	VOLO: Vision Outlooker for Visual Recognition [103]
27	86.8%	193M	0	VOLO: Vision Outlooker for Visual Recognition [103]
32	86.5%	356M	0	Going deeper with Image Transformers [104]
35	86.4%	150M	0	All Tokens Matter: Token Labeling for Training Better Vision Transformers [105]
37	86.3%	271M	0	Going deeper with Image Transformers [104]
38	86.3%	307M	0	BEiT: BERT Pre-Training of Image Transformers [106]
39	86.3%	86M	0	VOLO: Vision Outlooker for Visual Recognition [103]
40	86.1%	271M	0	Going deeper with Image Transformers [104]

observed in generative models and the authors of that work referred to this phenomenon as the “irreducible entropy” of the task [108]. This further supports the hypothesis that there is a ceiling on the achievable accuracy for the ILSVRC 2012 Imagenet validation set [88]. They observed a similar saturation at the lower end of the compute budget scale, where smaller models achieved better accuracy than the power-law would predict.

Mixture of Experts (MOE) is a method of combining the outputs of multiple sub-models called *experts* using a *router* mechanism. Generally, these have been studied since the early 1990s [109][110][111]. More recently, they have been applied to computer vision tasks [112]. In [96], the authors endeavored to combine MOEs with transformers. They designed a network that, while containing a large number of parameters, not all parameters get used during inference and they demonstrate the network’s ability to achieve competitive results while using as little as half of the computational power available in the network on any sample. Interestingly, the router mechanism they designed doesn’t route entire images, but rather individual patches of the images, so that different transformers in the network operate on different patches, possibly of a single image. Additionally, they created a fixed buffer size per expert in their mixture to encourage load balancing among the experts which encourages the overall model not to end up favoring only a small subset of the experts.

The network designed by the authors of [97] is another design based on transformers. Transformers for visual tasks work by splitting the input into patches. The authors noted that in most cases, of the 200–500 patches produced for images of typical training sizes, about 8 or 16 of them were the most informative. They propose a mechanism that the call “TokenLearner” which, prior to the transformer block, learns which patches are significant and passes only those to the transformer. In so doing, they were able to reduce the total number of FLOPs by half and maintain classification accuracy. In addition to the TokenLearner module that precedes the transformer block, they devised a “TokenFuser” module that follows the transformer block which maps the result of the transformer operation back to the input’s original spatial resolution, which allows the input and output of the set of operations to maintain the same tensor shape, making them easier to fit into a model’s overall architecture.

In [101], the authors grapple with the fact that in transformer-based architectures for vision tasks, global self-attention is an extremely expensive operation (quadratic in complexity) compared to local self-attention, which limits interactions between tokens. Their attempt to find a middle option is to introduce what they term a “Cross-Shaped Window” (CSWin), which is an attention mechanism that involves computing self-attention for vertical and horizontal stripes of the input image in parallel. In addition, they introduce a new positional encoding scheme they call “Locally-enhanced Positional Encoding” (LePE), which they claim, “handles the local positional information better than existing encoding schemes”, “naturally supports arbitrary input resolutions”, and is “especially effective and friendly for downstream tasks”. LePE differs from other positional encoding schemes by, rather than being concatenated into the input before the transformer block as with absolute positional encoding (APE) [93] and conditional positional encoding (CPE) [113], moving the encoding inside the encoding block as with relative positional encoding (RPE) [114][102]. But rather than happening inside the softmax operation that uses the queries, keys, and values, LePE is applied directly to the values only.

[102] precedes and is cited by [101] and the two papers share an author. The approaches are also quite similar, though the leap from a network of transformers like are present in ViT to what the authors propose in this work is a little more apparent. In this work, the authors note that the spatial position of the patches of the images (the tokens) being used by all layers in ViT are the same. The authors argue that it is better to think of how the patches are divided up as being subject to a window that shifts across the image in subsequent layers. This allows for connections between overlapping regions in the image to be learned by combinations of transformers. This network is trained entirely on publicly available data, using the 14M image ImageNet-22k dataset for additional training data.

The authors of [104] start with a network similar to ViT, consisting of a series of transformer blocks with residual connections between them. They then altered this design in two specific ways. They posit that a problem with the ViT architecture is that the class token being passed along with the image patches through every transformer layer is asking the optimizer to optimize two contradictory objectives. Those objectives being learning the self-attention for the patches and learning the information that leads to the correct classification. In order to combat this, they propose using two different processing stages, the first of which is not passed the class token so that the transformers in this stage can focus solely on learning the self-attention, and only in this stage does the self-attention get updated. In the second stage, the class token is added, and the transformers begin learning the classification. Additionally, they added a learnable diagonal matrix they call the “LayerScale” which they multiply the output of a transformer block by before concatenating together with the path that skipped over that transformer block. They refer to this architecture as CaiT (Class-Attention in Image Transformers). This network is trained without using any additional training data.

In [105], the authors propose a method they call “token labelling”. The idea behind it is to have each token coming out of a transformer block learn a K-dimensional vector for the classification for that specific patch, where K is the number of classes, and the vector components represent the probabilities of that patch belonging to each class. And then for the final classification, these are averaged together across the patches and then combined with the overall image class to form a final prediction. A drawback to this method is that before doing this, each patch’s probability for each image must be generated and stored. This network is trained without using any additional training data.

The authors of [106] attempt to take the methods of BERT [94], which are applied to the natural language processing (NLP) domain and apply them to the vision domain. They call their attempt BEiT. To do this requires a pre-pre-training step that creates discrete token values for each patch of each image via an autoencoder. Then, during pre-training, a transformer-based BEiTEncoder is trained to encode image patches into their corresponding tokens, with the caveat that some of the image patches fed into the network are masked out. Then for the final task of image classification, the pre-trained model has an additional classifier network appended. This network is trained without using any additional training data.

The authors of [103] took note of the fact that all of the best performing transformer-based vision models were using large amounts of additional training data to achieve their results. This motivated them to study the use of transformers while training on only the actual Imagenet 1k training data. Their findings were that a major factor in this is the larger patch sizes (typically $16{\times}16$ or $14{\times}14$) that most transformer architectures use due to their quadratic complexity. The authors posit that this fails to encode sufficiently fine-grained information. Their solution, which at first seems counter-intuitive, is to increase the patch size to $28{\times}28$, which for images of size $224{\times}224$ means an $8{\times}8$ embedding. Then, within each of those patches, use a sliding window attention mechanism to relate the fine-grained information within those patches together. A series of these transformer blocks make up the first stage of their design. The second stage of their design is to split each of those embeddings into $2{\times}2$ embeddings of size $14{\times}14$ and again apply the sliding window attention mechanism. This network is trained without using any additional training data and is the highest ranked network to do so.

In their own words, the authors of [100] “aim not to introduce a new component or complexity, but to provide a recipe that uses the minimal number of tricks yet attains excellent performance on many tasks”. They refer to this “recipe” as Big Transfer (BiT). In their work, they show that BiT can be pre-trained once and then fine-tuned quite cheaply on the task it is transferred to using a simple heuristic for choosing the hyperparameters for the fine-tuning training. They call this heuristic the “Bit-HyperRule”. In their study they found that they could limit the hyperparameters that need fine-tuned to the learning rate schedule and whether or not to use MixUp [115] after transferring. The first step in their heuristic is to categorize the size of the dataset they are transferring to. They class datasets with fewer than 20k labeled examples as *small*, datasets with more than that, but less than 500k labeled examples as *medium*, and everything else as *large*. Then after transfer, for small datasets, they train for 500 steps, for medium, 10,000 steps, and for large, 20,000 steps, decaying the learning rate by a factor of 10 after 30%, 60%, and 90% of the training steps. They use MixUp with $\alpha = 0.1$ for medium and large datasets. The network they designed is based on ResNet-v2 [116], but instead of using Batch Normalization (BN), they use Group Normalization (GN) [117] and add Weight Standardization (WS) [118] to all of the convolutions. The authors argue that batch normalization is a poor choice for transfer learning due to the requirement to update running statistics and show that the combination of GN and WS has a significant positive impact on transfer learning tasks. This network is trained entirely on publicly available data, using the 14M image ImageNet-22k dataset for additional training data.

3.2 Convolutional (EfficientNet) Networks

EfficientNet [91] is a model family that consists of progressively larger models which have been optimized for computation and parameter efficiency using Neural Architecture Search [119], which is a reinforcement learning method that learns the best neural network architecture to use for a given task. See Table 3.2 for a comparison of the EfficientNet networks reviewed here.

Table 3.2: Convolutional (EfficientNet) Networks

Rank	top-1 Acc.	# of Params.	Addt'l. Training Samples	Paper Title
12	88.5%	480M	300M	Fixing the train-test resolution discrepancy: Fix-EfficientNet [120]
23	87.3%	208M	14M	EfficientNetV2: Smaller Models and Faster Training [121]
25	87.1%	66M	300M	Fixing the train-test resolution discrepancy: Fix-EfficientNet [120]
28	86.8%	120M	14M	EfficientNetV2: Smaller Models and Faster Training [121]
30	86.7%	43M	300M	Fixing the train-test resolution discrepancy: Fix-EfficientNet [120]
34	86.4%	30M	300M	Fixing the train-test resolution discrepancy: Fix-EfficientNet [120]

In [121], the authors of the original EfficientNet paper continue their work by introducing EfficientNetV2. In their study, they argue that the scale of regularization needs to be proportional to the original image size of the dataset's images. This includes varying the regularization on a single network design based on the original image size of the dataset it is being trained with. Networks that work with smaller images, should use less regularization, and networks that work with larger images should use more regularization. In their prior work, the authors scaled up the number of layers in every stage of their network by the same factor. In this study, they show that gradually adding additional layers in the later stages is superior. Their prior work achieved the then state-of-the-art top-1 accuracy of 84.4%. This extension to that work achieved 87.3% top-1 accuracy—nearly a 4% absolute improvement.

The work in [120] can appropriately be seen as an extension of their earlier work [122]. In both papers, the authors note that there exists a discrepancy between the prevalent data pre-processing operations during training vs. evaluation. It is common to extract random rectangles from training images and scale them to a certain size each epoch as a form of data augmentation, but during evaluation, the common practice is to choose a central crop of equivalent size. This differing approach during training and evaluation results in varying typical scales of the objects trained on compared to objects of the same class during evaluation, and crucially, unlike with the case of translation, CNNs do not respond to scale differences in a predictable manner. In both works, the authors combat the scale discrepancy by allowing the network to learn how to resize the images during both training and evaluation. The details of the method by which they accomplish this are quite involved and beyond the scope of this work. The interested reader is referred to the original works. In the first paper, the authors applied their method to ResNet networks and trained only with the 1.2M training images that are a part of the standard Imagenet-1k training set. In the second paper, they applied their method to EfficientNet [91] networks and used the standard Imagenet-1k training set with an additional 300M images for training.

3.3 Transformer/Convolution Hybrid Networks

Two of the works reviewed in this chapter, including the top-ranking design, endeavored to use a combination of transformers and convolutions in their designs. See Table 3.3 for a comparison of the transformer/convolution hybrid networks reviewed here.

Table 3.3: Transformer/Convolution Hybrid Networks

Rank	top-1 Acc.	# of Params.	Addt'l. Training Samples	Paper Title
1	90.88%	2440M	3B	CoAtNet: Marrying Convolution and Attention for All Data Sizes [123]
3	90.45%	1470M	3B	CoAtNet: Marrying Convolution and Attention for All Data Sizes [123]
19	87.7%	277M	14M	CvT: Introducing Convolutions to Vision Transformers [124]

The authors of [123] note that convolutional neural networks perform well due to their natural locality bias and tend to generalize well and converge relatively quickly, whereas networks employing transformers perform well because of their ability to find global interactions more easily than CNNs but have been shown to require much more data and many more parameters. In their work, the authors endeavored to combine the benefits of both convolution and attention by summing a global static convolution kernel with the attention matrix prior to the softmax normalization inside the transformer block's attention heads. They refer to this as *relative attention*. Because the global context required for relative attention has a quadratic complexity with respect to the spatial size of the input, the direct application of relative attention to the raw image is not computationally tractable. Thus, their overall network architecture begins with an initial stem of traditional convolutional operations, which they refer to as stage 0, that down-samples the input image to feature maps half of the original image's size. Then, stage 1 and 2 are Squeeze and Excitation [125] blocks that each further reduce the size of the filter maps by half. It is at this point the filter maps have attained a size that relative attention is able to cope with. As such, stages 3 and 4 are made up of a series of relative attention transformer blocks before the network goes on to a final global pooling and fully connected layer that leads to the output classification probabilities. Residual connections are made between each stage and before the feed-forward network of each transformer block. The authors pre-trained their networks on Google's internal JFT-3B dataset [95], which as the name implies, consists of 3 billion images. It is worthy of note that training their best performing network took 20.1K TPUv3-core days.

The authors of [124] start with ViT as a basis for their design and then introduce 3 changes. First, at the beginning of each transformer, they introduce what they call a *convolutional token embedding*, which involves reshaping the token sequence going into the transformer back into their 2D spatial positions and performing an overlapping, striding convolution. Then, they replace the linear projection before each self-attention block with what they call "convolutional projection", which uses depth-wise separable convolutions [126] on the 2D-reshaped token map. This replaces the linear projection used by ViT that is applied to the query, key, and value embeddings. Finally, they remove the positional encoding that is usually present in the first stage of a transformer block. The question regarding the necessity of positional encoding in transformers used for vision tasks had been previously raised and studied [113]. Notably, this is the highest rank achieved using less than 300M additional training samples, as well as being the highest-ranking design to use a public dataset (Imnagenet-22k) for its additional 14M samples of training data.

3.4 Using Neither Transformers nor Convolutions

A single network using neither transformers nor convolutions ranks among the top 40 state-of-the-art networks reviewed in this chapter (see Table 3.4).

Table 3.4: Networks Using Neither Transformers nor Convolutions

Rank	top-1 Acc.	# of Params.	Addt'l. Training Samples	Paper Title
17	87.94%	431M	300M	MLP-Mixer: An all-MLP Architecture for Vision [127]

The authors of [127] begin their introduction with the observation that "As the history of computer vision demonstrates, the availability of larger datasets coupled with increased computational capacity often leads to a paradigm shift". Ironically, their architecture involves avoiding the usage of the canonical paradigm shifting methods of convolutions and transformers and instead is made up entirely of simple multi-layered perceptrons (MLPs). Their architecture uses exclusively matrix multiplication, reshaping and transposition, and scalar nonlinearities. They use two different types of MLP layers. One which works independently on image patches, which "mix" the per-location features, and one which works across patches, which "mix" spatial information. They build their architecture from a series of "Mixer" layers, each of which is made up of each of the two types of "mixer" MLPs, each of which is two fully-connected layers and a GELU [128] nonlinearity. Mixer layers also include residual connections around the mixing sub-layers.

3.5 Innovations Related to Training Procedures

In the remaining works reviewed in this chapter, the authors credit their achievement of state-of-the-art results not on the design of the network they used, but rather on other innovations related to the training of the networks (see Table 3.5).

Pseudo-labeling [134] involves using a teacher network that generates pseudo-labels on unlabeled data that is fed into a student network in tandem with labeled data. Eventually, the student outperforms the teacher. In [129], the authors extended on this idea by allowing the teacher to receive feedback from the student and then to adapt. Specifically, how well the student performs on the labeled data is fed back to the teacher as a reward signal for the quality of the pseudo-labels it generated. This surprisingly simple idea leads to the highest ranked design reviewed in this chapter that does not use transformers.

The work presented in [132] is clearly the prior steppingstone that led to [129] reviewed above, as the methods described are quite similar, and the papers share 3 authors. The first key difference in this work is the attention they pay to the role of noise in the teacher-student training process, thus the name NoisyStudent. They never inject

Table 3.5: Innovations Related to Training Procedures

Rank	top-1 Acc.	# of Params.	Addt'l. Training Samples	Paper Title
5	90.2%	480M	300M	Meta Pseudo Labels [129]
6	90%	390M	300M	Meta Pseudo Labels [129]
7	89.2%	527M	300M	High-Performance Large-Scale Image Recognition Without Normalization [130]
9	88.64%	480M	1.8B	Scaling Up Visual and Vision-Language Representation Learning With Noisy Text Supervision [98]
10	88.61%	480M	300M	Sharpness-Aware Minimization for Efficiently Improving Generalization [131]
13	88.4%	480M	300M	Self-training with Noisy Student improves ImageNet classification [132]
29	86.78%	377M	0	Drawing Multiple Augmentation Samples Per Image During Training Efficiently Decreases Test Error [133]
31	86.5%	438M	0	High-Performance Large-Scale Image Recognition Without Normalization [130]
33	86.45%	255M	0	Drawing Multiple Augmentation Samples Per Image During Training Efficiently Decreases Test Error [133]
36	86.3%	377M	0	High-Performance Large-Scale Image Recognition Without Normalization [130]

noise in the teacher model so that when it generates pseudo labels, those labels are as accurate as possible. However, when training the student, they inject considerable noise using RandAugment [135], dropout [136], and stochastic depth [137]. The second key difference is that rather than having the single student feedback to the single teacher, in this work, the authors follow a self-training framework [138] consisting of three steps. The first step is training the teacher with labeled data. The second step is to generate pseudo labels for unlabeled data with the teacher. The third step is to train the student with a mixture of labeled and pseudo-labeled data. These steps are repeated several times, each time promoting the prior student to be the new teacher and creating a new student model. The authors compare their method to Knowledge Distillation [116] but note that in that work the student was often smaller so that it could infer faster and did not inject noise so aggressively. They say that their method could be thought of as Knowledge Expansion in that the student is larger, with greater capacity and taught in a difficult environment made up of more noise.

In [98] the authors note that there are no publicly available labeled datasets of the order being used by many of the state-of-the-art network designs (e.g., JFT-300M and JFT-3B). This is due in large part to how costly and labor intensive it is to curate such datasets. In their work, they describe a process of downloading 1.8B images accompanied with alt-text from the internet, and rather than doing labor intensive curation, instead opt to only perform a small amount of filtering to the alt-text. Although they don't give a detailed explanation of their filtering process, it would stand to reason that they would filter out words that occurred very infrequently or extremely frequently. After the filtering process, they then have multiple noisy "labels", one per word in the alt-text, per image. Prior to doing training for the image classification task, they trained a different model to embed the image and alt-text pairs of their 1.8B image dataset into a shared embedding space where matched pairs were pushed together, and unmatched pairs were pushed apart. They then used this embedding to give each of the images' associated alt-text words different weights as labels.

The majority of networks, especially very deep networks like ResNets [92], employ Batch Normalization (BN) [139]. BN has the effect of smoothing the loss landscape which allows for larger learning rates and larger batch sizes. However, BN is a costly operation, behaves differently during training than it does evaluation, and breaks the independence among the training examples in each batch. Furthermore, BN results in a tight coupling of batch size to network performance such that when the batch size is too small, the network performs poorly. The authors of [130] believe that in the long term, reliance on BN will impede progress in neural network research. They noted that by suppressing the scale of the activations on residual branches in ResNets, networks can be trained effectively without BN. Specifically, they propose *Adaptive Gradient Clipping* (AGC) which works by clipping the gradients based on the ratio of gradient norms to parameter norms. The authors note that their work is closely related to recent work studying “normalized optimizers” [140][141][142] which ignore gradient scale and instead choose adaptive learning rates inversely proportional to the gradient norms. They state that “AGC can be interpreted as a relaxation of normalized optimizers, which imposes a maximum update size based on the parameter norm but does not simultaneously impose a lower-bound on the update size or ignore the gradient magnitude”.

The authors of [131] point out that with the heavily overparameterized models that are commonly in use, minimizing the training loss, which is the usual goal when training neural networks, can easily result in a suboptimal model. They propose a simple, yet effective approach of not only minimizing the training loss but while doing so, simultaneously minimizing the curvature of the loss landscape in the neighborhood of the loss. Among their other results, notably, they show that when using Sharpness-Aware Minimization (SAM), they achieve robustness to noisy labels “on par with that provided by state-of-the-art procedures that specifically target learning with noisy labels”. In their related work section, they note that similar superior generalization had previously been observed by achieving wider minima, not by explicitly searching for such, but by arriving at it by evaluating on a moving average of the prior training weights [143].

The usual approach to online data augmentation is to draw n samples from the training data, augment each of them with whatever augmentation procedure is being followed and then submit that batch of n augmented images to the training procedure. In [133], similar to earlier work as in [144] and [145], the authors perform a study of the consequences of drawing n samples, augmenting each of them c times and submitting a batch of size cn to be trained. One of their key findings is that for integer values of c greater than 1, higher accuracies were achieved, even in the presence of fixed batch sizes, which means the number of unique images in each batch was fewer. The authors noted that this was especially true in the cases of large batch sizes. The authors state of such models that “despite their superior performance on the test set, large augmentation multiplicities achieve slower convergence on the training set.” Perhaps it is not “despite” this but at least in part *because* of this. The authors also note that prior work has found that observations of the regularizing effect of large learning rates was proportional to the batch size used [146][147][148]. An interesting hypothesis put forward by the authors is that this observation is a specific case where c is held at 1 of the more general principle that the regularizing effect of large learning rates is proportional to the number of unique training samples in the batch.

3.6 Summary

The general trend towards larger and larger model capacities as measured in the number of trainable parameters is readily apparent (see Figure 3.1). One thing that could be overlooked, though, is that along with the trend towards increased model capacities there exists the trend toward using more and more additional training data (see Figure 3.2), the two largest sets of which are not publicly available.

These trends present problems for independent researchers, researchers who are University faculty, and smaller labs. The first such problem is simply the availability of data. The creation of Imagenet represents a turning point in the history of computer vision. Up to that point, dataset sizes were most commonly measured in the 10s of thousands of samples. Imagenet has provided a mega scale dataset and has become the de-facto measure of state of the art as a result. The second problem is the compute power required to train giga scale models on giga scale data. For example, the highest ranked model had to be trained for 20,100 TPUv3-core days. The published price for this much compute is over $300,000 and would take 10 days using the largest TPUv3 pod that exists. On a consumer GPU like an NVIDIA GeForce RTX 3090, it would take approximately 18 years to train this model. As such, state-of-the-art research is now dominated by large corporations like Google, Microsoft, and Facebook.

What can the deep learning computer vision community do to re-democratize the research of state-of-the-art methods? The directions that have been pursued recently should not be abandoned nor should such research be ceded to large corporations. Instead, additional vectors of research should be explored. One such vector would be the study of network architectures that can cope better with less data and less fine tuning. This work explores using Homogeneous Vector Capsules in a variety of architectures to accomplish just that. Another such vector would be the **data** (as opposed to the network design) used to train these models with. Prioritizing the collection of new standard benchmark datasets that fill the gaps between CIFAR-100 and Imagenet and between Imagenet and the internal giga scale datasets of large corporations should be considered. And furthermore, researching and developing analytical methods of measuring sufficiency in data completeness should be investigated, as opposed to simply assuming more data will always be better. Chapter 7 and chapter 8 pursue this direction.

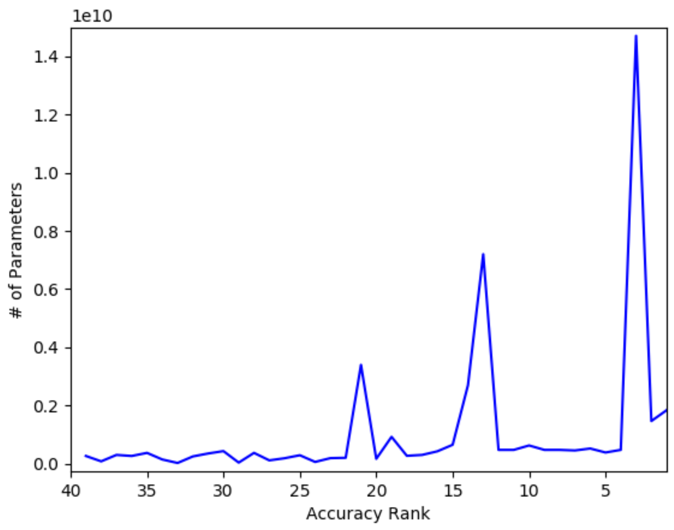


Figure 3.1: The number of model parameters used to achieve the top 40 best accuracies on the ILSVRC 2012 Imagenet validation set.

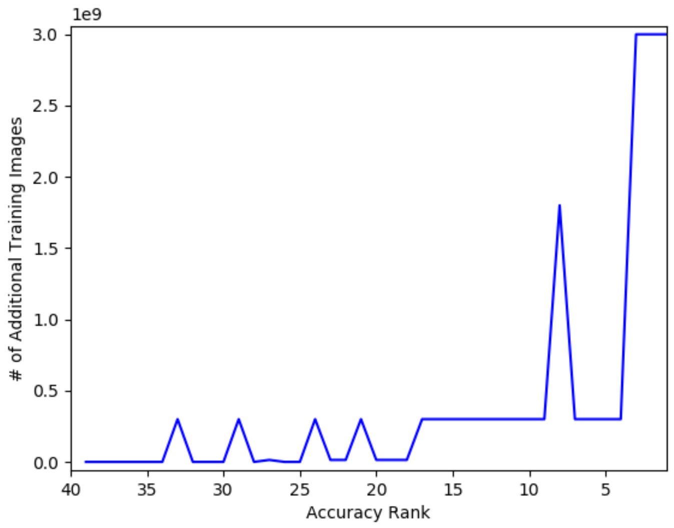


Figure 3.2: The amount of extra training data used to achieve the top 40 best accuracies on the ILSVRC 2012 Imagenet validation set.

Chapter 4

An Investigation of Capsule Dynamics

In this chapter, a new method of learning between capsules layers is presented and compared to previous methods present in the literature. Then a series of experiments across a wide variety of parameters is investigated. These parameters include using:

- 3 different datasets of increasing difficulty
- 4 different optimization strategies
- 3 methods of constructing the first layer of capsules from the preceding filter maps
- 2 different network architectures

These parameters result in 192 combinations for which one experiment of each is conducted.

4.1 Capsule Layers Configuration

In [2] the authors proposed two final layers of capsules. The first of which has 8 dimensions shaped as a vector and the second of which has 16 dimensions, also shaped as a vector. The transformation between the two layers of capsules is a typical matrix multiplication, wherein every pair of capsules has an associated 16×8 matrix of trainable parameters and is multiplied by each of the 8-dimensional vector capsules and summed to form the input into the 16-dimensional capsule. In Equation 4.1, an equivalent transformation simplified to two and four dimensions for clarity is presented.

$$\begin{bmatrix} a & b \\ c & d \\ e & f \\ g & h \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} ax_1 + bx_2 \\ cx_1 + dx_2 \\ ex_1 + fx_2 \\ gx_1 + hx_2 \end{bmatrix} \tag{4.1}$$

The work presented in this chapter has been published in *IEEE Access*, vol. 9, pp. 48519–48530, 2021 [18].

In this equation, as well as in Equation 4.2 and Equation 4.3, the variables a through h represent learned weights that are being applied in the transformation, and the variables x_1 through x_2 (in Equation 4.1) and x_4 (in Equation 4.2 and Equation 4.3) represent the values computed from the previous operation in the network. In each case, the second matrix on the left-hand side of the equation is the first capsule and the matrix on the right hand side is the second capsule.

A problem with this transformation in Equation 4.1 becomes apparent when viewing it as an overdetermined system of linear equations in matrix form: every dimension in the second layer of capsules, beyond the dimensions in the first layer, are at best redundant and more probably, due to the random initialization of the weights, a challenge to the optimization algorithm used during backpropagation to reconcile multiple differing losses derived from each activation in the previous layer.

Also, it should be noted that each dimension of the second layer of capsules is a linear combination of all dimensions of the first layer of capsules. This is a desirable property in a fully connected layer in a neural network. However, with the interpretation and empirical verification in [2] of the dimensions of a capsule as being distinct features of a given sample, it is the hypothesis of this work that this entangling of distinct features from one layer into all features in the next layer is an undesirable property.

In their follow-up work [6], the authors switched to using an equivalent number of dimensions in neighboring capsule layers, though they did not cite their motivation for doing so as to alleviate the problem of an overdetermined system. Additionally, they shaped their capsules as matrices rather than vectors. The authors noted that this reshaping had the effect of reducing the number of trainable parameters (for every pair of capsules) from being the product of the dimensions of the two layers of capsules to being only the number of dimensions of a single layer of capsules. This method of matrix capsules requires that the number of dimensions in neighboring layers be both equivalent and a perfect square. In Equation 4.2, an equivalent transformation simplified to four dimensions is presented:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} x_1 & x_2 \\ x_3 & x_4 \end{bmatrix} = \begin{bmatrix} ax_1 + bx_3 & ax_2 + bx_4 \\ cx_1 + dx_3 & cx_2 + dx_4 \end{bmatrix} \tag{4.2}$$

In addition to alleviating the problem of an overdetermined system and significantly reducing the number of trainable parameters, this formulation results in only the square root of the total number of features in the first layer being entangled with each feature in the second layer.

This work proposes a new method for the transformation from one layer of capsules to the next. Rather than using the typical transformation matrix, the proposed method involves using a transformation vector and rather than using the typical matrix multiplication, the proposed method involves using the Hadamard product (element-wise multiplication). This method is shown in Equation 4.3, simplified to four dimensions for clarity:

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \odot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} ax_1 \\ bx_2 \\ cx_3 \\ dx_4 \end{bmatrix} \tag{4.3}$$

This method goes back to using vectors for the shape of the capsules and requires that the neighboring layers of capsules be of equivalent dimension, thus these capsules are called homogeneous vector capsules. With the constraint of requiring equivalent dimensions in the capsule layers, this method comes with the following benefits:

1. Because this method uses the Hadamard product rather than typical matrix multiplication, the drawback of using the more intuitive vector shape for a capsule is removed, as the number of trainable parameters per pair of capsules stays equal to the number of dimensions in the capsules (as in [6]), rather than being that number of dimensions squared (as in [2]).
2. By the nature of the Hadamard product, this method cannot suffer from the problem of an overdetermined system.
3. This fully disentangles features from the dimensions in the first layer of capsules from differing dimensions in the subsequent layer of capsules. i.e., each dimension in the first layer maps to one and only one dimension in the second layer.
4. This eliminates all of the addition operations used in matrix multiplication for a modest reduction in computational cost.
5. Whereas the number of dimensions in [6] must be a perfect square, HVCs can be composed of any number of dimensions that evenly divides the number of neurons being input into them.

4.2 Experimental Design

The experiments in this chapter are designed to compare (a) baseline neural network architectures that use the standard approach of transforming the final convolutional layer in the network as in Figure 4.1 with (b) reshaping the final set of feature maps into j n-dimensional vector capsules, where $j \cdot n$ is the total number of weights coming out of the final set of feature maps. When doing this, the final classification is done, rather than with scalar output neurons, with y n-dimensional vector capsules as in Figure 4.2, that are reduced to predictions by computing the Euclidian norm of the vectors.

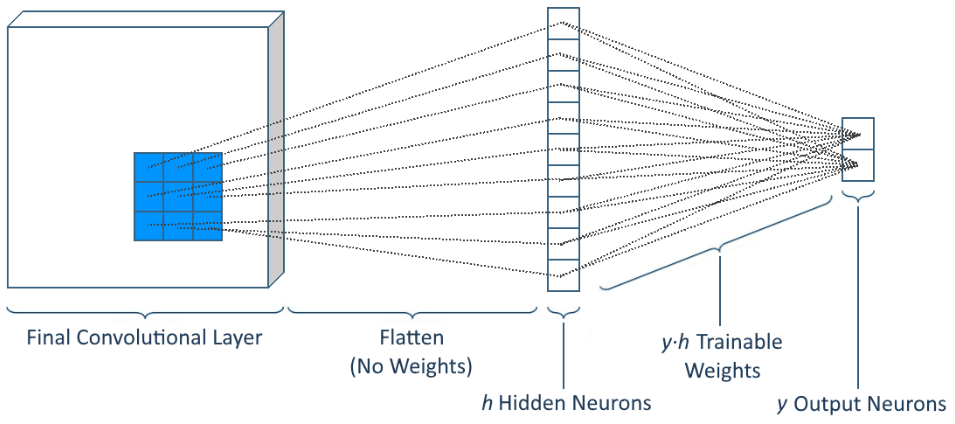


Figure 4.1: The standard approach to transforming the final convolutional layer into class predictions.

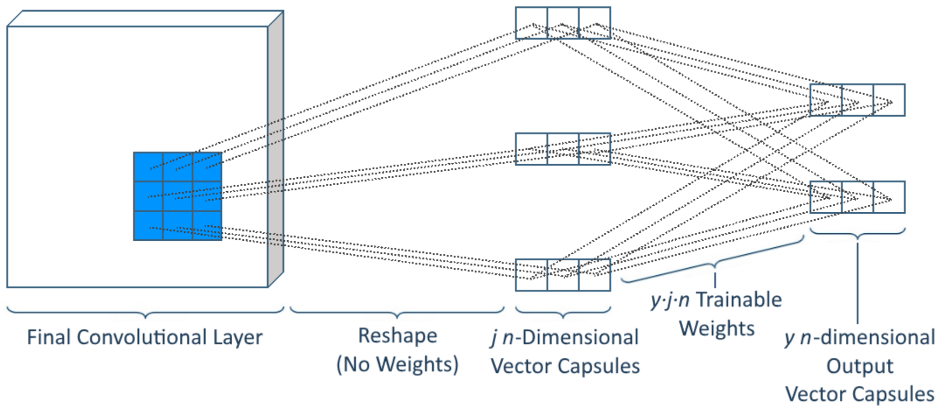


Figure 4.2: Using homogeneous vector capsules to transform the final convolutional layer into class predictions.

4.2.1 Networks

The experiments in this chapter are conducted using two convolutional neural network architectures. The first network is a typical simple monolithic CNN featuring a series of 3$\times$3 convolutions interspersed with max pooling operations (see Table 4.1). The motivation behind this design was to examine the effect of capsules on a simple, widely understood and easily implemented architecture with a low number of parameters (in this case ~ 1.6M to ~ 22.1M, depending on the number of output classes and the capsule configuration). No drop-out or L2 (or any other form) of regularization was used with this architecture. The second network is the popular Inception v3 architecture [13]. This network was chosen due to its good performance given the relatively low number of parameters it uses (~ 23.2M to ~ 156.1M, depending on the number of output classes and the capsule configuration).

4.2.2 Datasets

The experiments in this chapter used three datasets of increasing difficulty:

- The full-sized Imagenette [149], a subset of ImageNet consisting of 10 easily classified classes: tench, English springer, cassette player, chain saw, church, French horn, garbage truck, gas pump, golf ball, and parachute.
- The full-sized Imagewoof [149], a subset of ImageNet consisting of 10 more closely related classes, all of which are dog breeds: Australian terrier, Border terrier, Samoyed, Beagle, Shih-Tzu, English foxhound, Rhodesian ridgeback, Dingo, Golden retriever, and Old English sheepdog.
- Food-101 [150], a challenging and noisy dataset consisting of 101 classes of images retrieved from the now defunct foodspotting.com.

4.2.3 Optimization Strategies

The experiments in this chapter used four different optimization strategies. RMSProp has been a popular choice for optimizing convolutional neural networks since [13]. This strategy which is denoted O1 (see Table 4.2) is the strategy used in [13] whereas the strategy denoted O2 is the strategy employed by the official TensorFlow implementation of Inception v3 published on github.com[1] which results in slightly higher accuracy. In addition, two other optimization strategies were experimented with. O3 is the Adam optimizer with the defaults suggested in [9], and O4 is the Adam optimizer with a slowly decaying base learning rate.

[1]https://github.com/tensorflow/models/blob/master/research/slim/train_image_classifier.py

Table 4.1: The stem of the simple monolithic CNN.

Operation	Feature Maps	Output Shape
3×3 convolution w/stride 2	32	149×149×32
3×3 convolution w/stride 1	32	147×147×32
3×3 convolution w/stride 1	32	145×145×32
2×2 max pool w/stride 1	N/A*	72×72×32
3×3 convolution w/stride 1	64	70×70×64
3×3 convolution w/stride 1	64	68×68×64
3×3 convolution w/stride 1	64	66×66×64
2×2 max pool w/stride 1	N/A*	33×33×64
3×3 convolution w/stride	128	31×31×128
3×3 convolution w/stride 1	128	29×29×128
3×3 convolution w/stride 1	128	27×27×128
2×2 max pool w/stride 1	N/A*	13×13×128
3×3 convolution w/stride 1	256	11×11×256
3×3 convolution w/stride 1	256	9×9×256

The baseline experiments were classified through a fully connected layer after flattening the final set of feature maps as in Figure 4.1. All other experiments used HVCs as in Figure 4.2.
* Max-pooling is a sub-sampling operation that involves no trainable parameters.

Table 4.2: Optimizers used for all experiments.

Optimizer	Description
O1	RMSProp w/epsilon 1 and 0.045 learning rate exponentially decaying every 2 epochs by 0.94
O2	RMSProp w/epsilon 1 and 0.1 learning rate exponentially decaying every 30 epochs by 0.16
O3	Adam w/0.001 learning rate
O4	Adam w/0.001 learning rate exponentially decaying every epoch by 0.96

4.2.4 Capsule Configuration

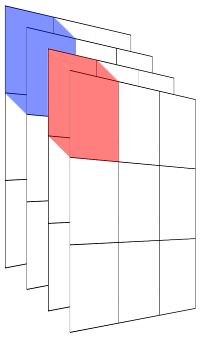
The experiments in this chapter explored 3 different methods of transforming the final set of feature maps into capsules. The first method creates multiple capsules for each distinct x and y coordinate of the feature maps (see Figure 4.3). The second method creates a single capsule for each distinct x and y coordinate of the feature maps (see Figure 4.4). The intuition behind these two methods is that each position in the feature map represents a meaningful feature and that using capsules to "group" these together from multiple filter maps encourages the feature maps to cooperate. The difference being that in the multiple capsule case, multiple disparate groups are allowed, wherein the single capsule case, only one such group is allowed. The third method creates a single capsule for each distinct feature map (see Figure 4.5). The intuition behind this method is the standard interpretation of a feature map (i.e., it represents a single feature per map). However, rather than allowing each dimension of the feature map to learn independently through a fully connected layer, capsules are used to maintain the cohesion among the dimensions.

The experiments in this chapter included eight variations of the simple monolithic CNN architecture (see Table 4.3). The first such variation, denoted S1, is the baseline model that flattens the final set of feature maps and then classifies through a layer of fully connected neurons. Variations S2 through S6 reshape the final set of feature maps as in Figure 4.3. Variation S7 reshapes the final set of feature maps as in Figure 4.4. Variation S8 reshapes the final set of feature maps as in Figure 4.5. In variations S2 through S8, after the first layer of capsules are shaped, they are then classified through the second set of capsules that form the HVC pairs.

The experiments in this chapter included eight variations of the Inception v3 architecture (see Table 4.4). The first such variation, denoted I1, is the baseline model as described in [13]. Variations I2 through I6 reshape the final set of feature maps in both the main and auxiliary branches as in Figure 4.3. Variation I7 reshapes the final set of feature maps in both branches as in Figure 4.4, and Variation I8 reshapes the final set of feature maps in both branches as in Figure 4.5. In variations I2 through I8, after the first layer of capsules are shaped each branch is then classified through the second set of capsules that form the HVC pairs.

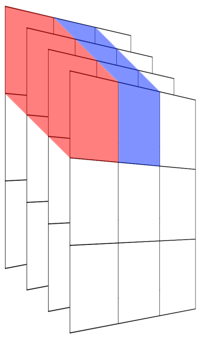
Unique to the Inception v3 architecture relative to the simple monolithic CNN is that, in the baseline model I1 and models I2 through I6, the final operation before the flattening operations in both the main and auxiliary outputs reduce the feature maps to 1×1. In the main branch, this is accomplished via global average pooling [151] and in the auxiliary branch, this is accomplished by performing a 5×5 convolution on a set of 5×5 feature maps. Both of these methods effectively collapse the spatial information present in the preceding operations into a single scalar value per feature map. Despite this, these

Figure 4.3: Segmented Z-Derived Capsules.



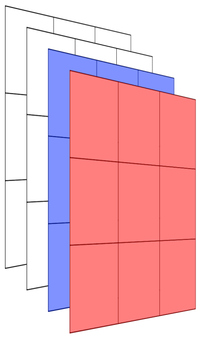
In this example, the filter maps have been converted into two 2-dimensional capsules for each distinct x and y coordinate of the feature maps. The first 2 of 18 such capsules are highlighted in red and blue respectively. Models S2, I2, I3, I4, I5, and the main output of I6 use this method.

Figure 4.4: Unbroken Z-Derived Capsules.



In this example, the filter maps have been converted into a single 4-dimensional capsule for each distinct x and y coordinate of the feature maps. The first 2 of 9 such capsules are highlighted in red and blue respectively. Models S4, I8, and the auxiliary output of I6 use this method.

Figure 4.5: XY-Derived Capsules.



In this example, the filter maps have been converted into four 9-dimensional capsules, each made from an entire feature map. The first 2 of 4 such capsules are highlighted in red and blue respectively. Models S3 and I7 use this method.

Table 4.3: Models used for the experiments conducted using a simple monolithic CNN.

Model	Capsule Configuration	HVC Dimensions	# of HVCs
S1	No capsules — This is the baseline model		
S2	See Figure 4.3	8	2,592
S3	See Figure 4.3	16	1,296
S4	See Figure 4.3	32	648
S5	See Figure 4.3	64	324
S6	See Figure 4.3	128	162
S7	See Figure 4.4	256	81
S8	See Figure 4.5	81	256

Table 4.4: Models used for the experiments conducted using the Inception v3 architecture.

		Main Out HVCs		Aux Out HVCs	
Model	Capsule Configuration	Dimensions	#	Dimensions	#
I1	No capsules — This is the baseline model				
I2	See Figure 4.3	8	256	8	16
I3	See Figure 4.3	16	128	16	8
I4	See Figure 4.3	32	64	32	4
I5	See Figure 4.3	64	32	64	2
I6	See Figure 4.3	128	16	128	1
I7	See Figure 4.4	2,048	64	128	25
I8	See Figure 4.5	64	2,048	25	128

global operations have been empirically shown to be effective in maintaining models' ability to achieve good generalization and accuracy, all while significantly reducing the number of trainable parameters. Generally, these global operations precede a final fully connected layer from which classification is performed. The larger the number of classes being classified, the more pronounced the reduction in trainable parameters is. For two of the Inception v3 experiments, I7 and I8, these global operations were removed, which results in the final set of feature maps in the main branch being 8×8 and the final set in the auxiliary branch being 5×5. This in turn results in an increasing number of parameters in the model as the number of output classes increases (see Table 4.5).

4.2.5 Additional Experimental Parameters

Additional experimental parameters are as follows:

- All activations were ReLU preceded by batch normalization [139].
- Loss for the Inception v3 experiments was computed using the label-smoothing regularization method as in [13], whereas categorical cross-entropy was used for the simple monolithic CNN experiments.
- All experiments ran for 100 epochs.
- Evaluations were performed using the exponential moving average of past weights as in [143], with a decay factor of 0.999.
- A batch size of 32 was used for Imagenette and Imagewoof. A batch size of 96 was used for Food-101 for models S1-S8 and I1-I6 and a batch size of 68 for models I7 and I8 (see Table 4.3 and Table 4.4). These batch sizes were dictated by the constraints of the available hardware.
- An image size of 299×299 for all images in all datasets, in all cases augmented using the strategy employed by the official TensorFlow implementation of Inception v3 published on github.com.[2]

[2]https://github.com/tensorflow/models/blob/master/research/slim/preprocessing/inception_preprocessing.py

4.3 Experimental Results

4.3.1 The Simple Monolithic CNN

As can be seen in Table 4.6 and Figure 4.6, models S2-S8, which used HVCs, wildly outperformed the baseline model S1 with all optimization strategies, on all three datasets tested. The experiment for the baseline model S1 when using optimization strategy O2 was not able to learn to a better accuracy than random guessing on Imagenette and Imagewoof and actually "learned" to achieve an accuracy of 0% on Food-101. The average accuracy of the experiments of the baseline model S1 with all optimization strategies, on all three datasets, excluding those experiments where the model had not learned to an accuracy better than random guessing, is 64.55%. The average accuracy of the experiments for models S2-S8 with all optimization strategies, on all three datasets is 76.92%. This is a relative improvement of 19.16%. For all three datasets, the best performing experiments used optimization strategy O4, which was the Adam optimizer with an appropriately small ϵ resulting in the intended adaptability along with a slowly decaying base learning rate.

6 out of 12 of the combinations of optimization strategy and dataset achieved their highest accuracy with model S7, which used the method that creates a single capsule from each distinct x and y coordinate of the feature maps. 4 out of 12 of the combinations achieved their highest accuracy with model S6, which used the method that creates 2 capsules from each distinct x and y coordinate of the feature maps. This suggests that deriving 1 or 2 capsules for each distinct x and y coordinate of the feature maps is superior to deriving a higher number of capsules from each such x and y coordinate (models S2-S5) or deriving the capsules from entire, individual feature maps (model S8).

4.3.2 Inception v3

Optimization strategies O1 and O2 are the two optimization strategies published and used to train Inception v3 on ImageNet [13]. It would be understandable, yet naïve, to assume that these optimization strategies would be superior choices in general. But as can be seen in Table 4.7 and Figure 4.7, only occasionally did either strategy O1 or O2 outperform O3, and only once did O2 outperform O4. This demonstrates that the hyperparameters are finely tuned, not just to the network architecture, but also to the data.

The only times the baseline model I1 outperformed all capsule models I2-I8 was for the Food-101 dataset when using optimization strategies O1 and O2. The best performing capsule model outperformed the baseline model I1 by an average of 1.32%

across all optimization strategies and datasets.

3 out of 12 of the combinations of optimization strategy and dataset achieved their highest accuracy with model I7. This stands in contrast to the experiments on the simple monolithic CNN where twice as many combinations were superior for the analogous model S7. The two architectures are too dissimilar to draw any firm conclusions, but a reasonable hypothesis is that there are two factors contributing to this. First, creating a single capsule for each distinct x and y coordinate of all feature maps of the main output for Inception v3 results in 2,048 dimensional capsules (as the final set of feature maps is 2,048 in number) compared to only 256 dimensions for the capsules coming out of the final set of feature maps in the simple monolithic CNN. Second is the presence of the auxiliary output stem in Inception v3. 5 out of 12 combinations achieved their highest accuracy with model I6 and 3 out of 12 with I5. These results are less conclusive than those with the simple monolithic CNN and permit fewer firm conclusions. However, these experiments do suggest that a single capsule to a small number of capsules for each distinct x and y coordinate of all feature maps is the superior choice.

4.3.3 Optimization Strategy

For models S1-S8 and for all three datasets tested, optimization strategy O4 achieved the highest accuracy. The second highest accuracy was achieved with strategy O1 twice and with O2 once. For models I1-I8 and for all three datasets, optimization strategy O3 achieved the highest accuracy twice and O4 once.

With the Food-101 dataset, arguably the most difficult of the three datasets tested, baseline model S1 performed better with the quasi-adaptive optimization strategy O1 than with either of the truly adaptive strategies O3 or O4. And yet, strategy O2 achieved a top accuracy of 0% for this model. O1 and O2 are the same optimization algorithm but parameterized differently. Further, these parameterizations were not ad-hoc, but rather parameterizations that are published along with the Inception v3 architecture and perform well on the ImageNet dataset with that architecture. This underscores just how important hyperparameter choice can be and how closely related to both network structure and dataset it truly is. This in turn underscores the relative utility of an adaptive gradient descent method that is less reliant on hyperparameter choice.

With adaptive gradient descent methods, there is a base learning rate η that is the same for all parameters and a separate per-parameter learning rate that is adapted based on previous gradient updates to that parameter. The two are multiplied together to determine each parameter's actual update. With the Adam optimizer, the suggested base learning rate η is 0.001 and the range of possible values for the per-parameter update are 0 to 10^{10}. After being multiplied together, this gives a range of possible per-parameter updates of 0 to 10^{7}. This is exactly what optimization strategy O3 uses for each parameter for the duration of the training. Optimization strategy O4 starts with this range for each parameter, and then gradually decays the base learning rate η over the epochs of training such that the resultant per-parameter updates are eventually constrained to a range of 0 to 10^4. This is similar to, but far less extreme (by four orders of magnitude) than the dampening effect caused by using a large ϵ in the denominator of the per-parameter term of an adaptive gradient descent method (contra its intended purpose), as is the case with optimization strategies O1 and O2. Further, when decaying the *base* learning rate in the manner of optimization strategy O4, the dampening is applied gradually over time as the parameter values descend the loss landscape, rather than statically for the duration of training (as in the case of a large ϵ).

Effectively, by allowing the learning rates of different parameters to change based on what has previously been learned, an adaptive gradient descent method attempts to achieve the goals of exploitation and exploration simultaneously. Exploration is achieved by decoupling each parameter from a single learning rate and exploitation is achieved by the coupling of each parameters' own learning rate to what had previously been learned. Using an adaptive gradient descent method with a large ϵ greatly reduces the amount of per-parameter exploitation possible. This shoulders the machine learning engineer with the task of choosing just the right hyperparameters to balance this small amount of variability in exploitation with the proper amount of exploration—the very thing adaptive gradient descent methods are meant to alleviate. This is why using a large ϵ is characterized as *quasi*-adaptive in this work. By using a truly adaptive gradient descent method (one with an appropriately small ϵ) and then decaying the *base* learning rate during training, the simultaneous explore/exploit nature of the method is preserved early in training and then slowly shifted to be more exploitative on average, but still allowing each parameter to have its own still rather large range of possible explore vs. exploit dispositions.

4.4 Summary

The experimentation detailed in this chapter demonstrates that:

1. Using HVCs on an advanced neural network architecture like Inception v3 increases the achievable accuracy by a small but significant margin.

2. Using HVCs on a simple monolithic CNN increases the achievable accuracy massively.

3. Deriving 1 or 2 capsules from each distinct x and y coordinate of all feature maps outperforms both deriving a larger number of capsules in the same manner and deriving capsules from entire, individual feature maps.

Figure 4.6: Classification accuracy on 4 simple CNN models, 3 different datasets, and 4 different optimization strategies.

Each chart is cross-referenced in Table 4.6

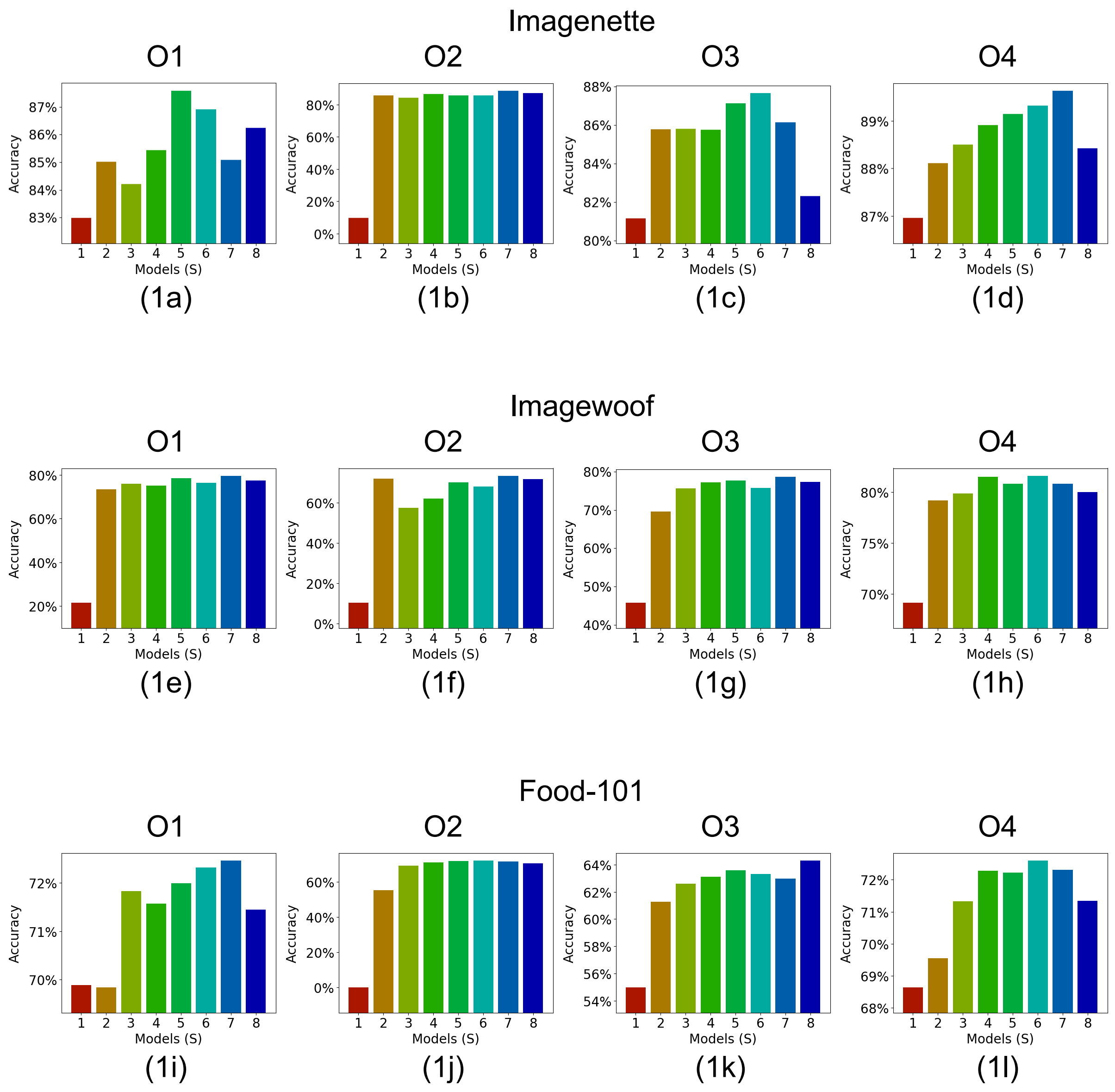
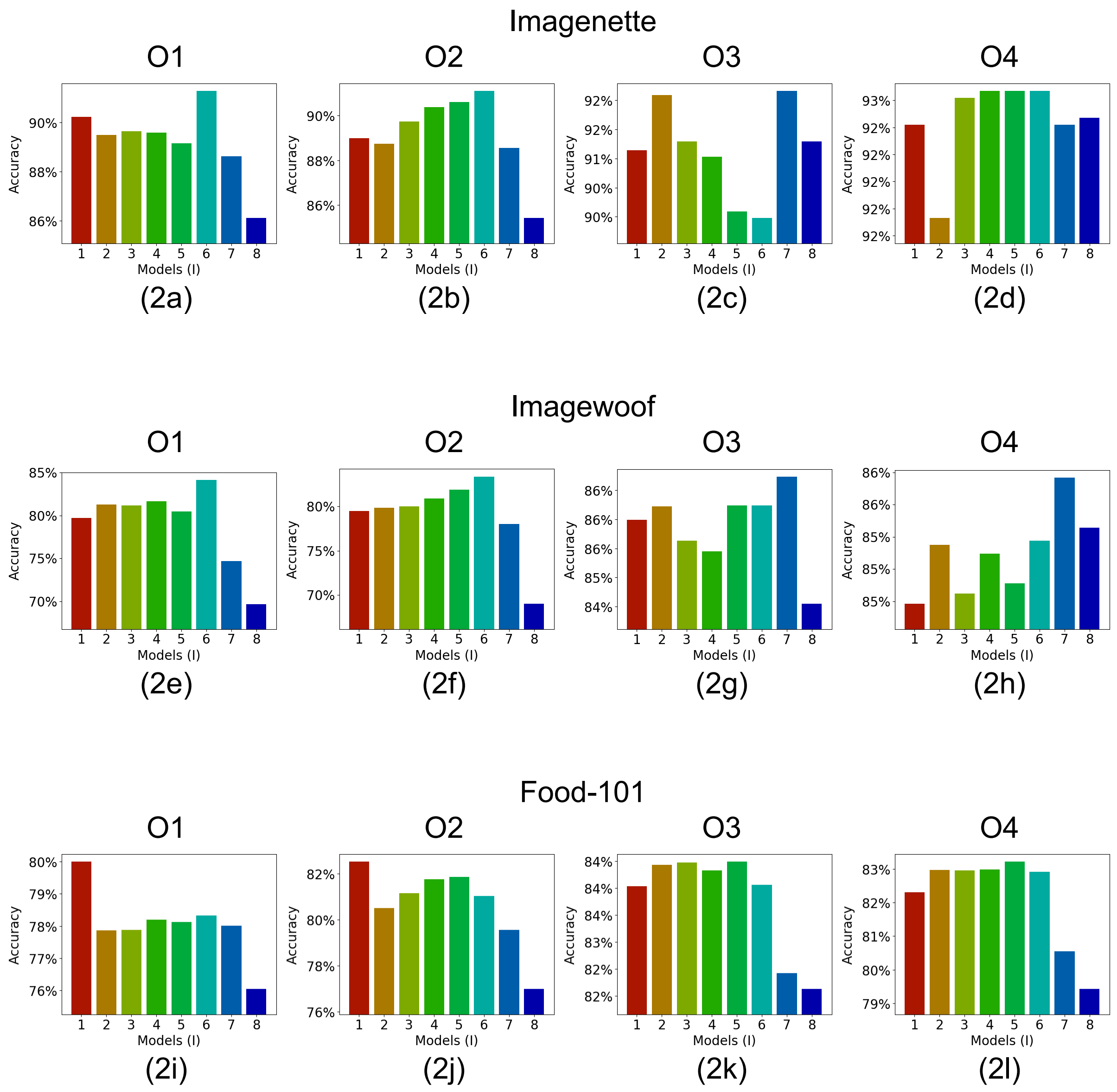


Figure 4.7: Classification accuracy on 8 Inception v3 models, 3 different datasets, and 4 different optimization strategies.

Each chart is cross-referenced in Table 4.7



And thus, that using HVCs enable convolutional neural network researchers to:

1. Use adaptive gradient descent methods when training CNNs without experiencing a generalization gap.
2. Save time and compute cycles searching for the best learning rates and learning rate decay schedules to use to train their network with a non-adaptive gradient descent method and instead use an adaptive gradient descent method that does not require this fine-tuning.

In conclusion, it is worth noting that due to constraints in terms of the available computational budget in conjunction with the wide coverage of parameters experimented with in this chapter resulted in only a single trial of each experiment being executed. In those cases where the accuracies between 2 individual experiments are near each other, firm conclusions cannot be drawn until additional trials are conducted. However, comparing groups of experiments together allows for the experiments in the classes to be treated as individual trials. For example, each capsule configuration of the simple monolithic CNN (S2-S8) was statistically significantly superior to the baseline (S1) when taking all combinations of dataset and optimization strategy as the trials.

Table 4.5: Trainable parameters used by models and number of classes.

Models	Final Feature Map Dimensions	Classes	# of Parameters
S1-S8	9×9	10	1.6M
		101	3.5M
		1000	22.1M
I1-I6	1×1	10	22.3M
		101	22.5M
		1000	24.5M
I7-I8	8×8 5×5	10	23.2M
		101	35.4M
		1000	156.1M

The difference between the number of trainable parameters for otherwise equivalent models using fully connected layers vs. HVCs is negligible. For S1 vs. any of S2 through S8, the former has only 0.16% fewer parameters. As models get larger the difference lessens and eventually inverts. For example, the difference between I1 and any of I2 through I6 when classifying 1000 classes is 0.007% fewer parameters for the HVC models.

Table 4.6: Classification accuracy on 8 simple monolithic CNN models, 3 different datasets, and 4 different optimization strategies.

Dataset	Cross Reference to Figure 4.6	Optimizer	Models (see Table 4.3)			
			S1	S2 (8d)	S3 (16d)	S4 (32d)
Imagenette	1a	O1	82.99%	85.02%	84.22%	85.43%
	1b	O2	9.68%	85.66%	84.38%	86.68%
	1c	O3	81.15%	85.78%	85.81%	85.76%
	1d	O4	86.96%	88.11%	88.50%	88.91%
Imagewoof	1e	O1	21.49%	73.39%	76.00%	75.18%
	1f	O2	10.45%	72.03%	57.45%	62.06%
	1g	O3	45.72%	69.54%	75.64%	77.23%
	1h	O4	69.16%	79.18%	79.84%	81.48%
Food-101	1i	O1	69.89%	69.84%	71.83%	71.57%
	1j	O2	0%	55.41%	69.29%	71.23%
	1k	O3	54.98%	61.27%	62.60%	63.10%
	1l	O4	68.64%	69.55%	71.33%	72.28%

Dataset	Cross Reference to Figure 4.6	Optimizer	Models (see Table 4.3)			
			S5 (64d)	S6 (128d)	S7 (256d)	S8 (81d)
Imagenette	1a	O1	**87.58%**	86.91%	85.09%	86.24%
	1b	O2	85.86%	85.71%	**88.65%**	87.12%
	1c	O3	87.12%	**87.65%**	86.14%	82.30%
	1d	O4	89.14%	89.32%	**89.63%**	88.42%
Imagewoof	1e	O1	78.43%	76.31%	**79.59%**	77.41%
	1f	O2	70.31%	68.21%	**73.44%**	71.85%
	1g	O3	77.72%	75.77%	**78.64%**	77.28%
	1h	O4	80.81%	**81.58%**	80.81%	79.97%
Food-101	1i	O1	71.99%	72.32%	**72.46%**	71.45%
	1j	O2	71.95%	**72.22%**	71.75%	70.73%
	1k	O3	63.59%	63.32%	62.98%	**64.30%**
	1l	O4	72.22%	**72.60%**	72.31%	71.33%

The first column in each row is a cross-reference to the charts in Figure 4.6. The model parentheticals refer to the number of capsule dimensions. For example, S2 (8d) refers to model S2 which uses 8-dimensional capsules.

Table 4.7: Classification accuracy on 8 Inception v3 models, 3 different datasets, and 4 different optimization strategies.

Dataset	Cross Reference to Figure 4.7	Optimizer	Models (see Table 4.4)			
			I1	I2 (8d)	I3 (16d)	I4 (32d)
Imagenette	2a	O1	90.24%	89.50%	89.65%	89.60%
	2b	O2	88.99%	88.73%	89.73%	90.37%
	2c	O3	91.14%	92.09%	91.29%	91.03%
	2d	O4	92.42%	91.73%	92.62%	**92.67%**
Imagewoof	2e	O1	79.71%	81.28%	81.17%	81.63%
	2f	O2	79.48%	79.84%	80.02%	80.89%
	2g	O3	85.99%	86.22%	85.63%	85.45%
	2h	O4	84.73%	85.19%	84.81%	85.12%
Food-101	2i	O1	**80.00%**	77.86%	77.88%	78.20%
	2j	O2	**82.52%**	80.51%	81.15%	81.76%
	2k	O3	84.03%	84.43%	84.47%	84.33%
	2l	O4	82.30%	82.97%	82.95%	82.98%

Dataset	Cross Reference to Figure 4.7	Optimizer	Models (see Table 4.4)			
			I5 (64d)	I6 (128d)	I7 (2,048d/128d)	I8 (64d/25d)
Imagenette	2a	O1	89.16%	**91.29%**	88.63%	86.12%
	2b	O2	90.60%	**91.01%**	88.55%	85.43%
	2c	O3	90.09%	89.98%	**92.16%**	91.29%
	2d	O4	**92.67%**	**92.67%**	92.42%	92.47%
Imagewoof	2e	O1	80.48%	**84.14%**	74.67%	69.65%
	2f	O2	81.89%	**83.38%**	78.02%	69.01%
	2g	O3	86.24%	86.24%	**86.73%**	84.55%
	2h	O4	84.89%	85.22%	**85.71%**	85.32%
Food-101	2i	O1	78.13%	78.32%	78.01%	76.05%
	2j	O2	81.86%	81.03%	79.56%	76.99%
	2k	O3	**84.49%**	84.06%	82.42%	82.13%
	2l	O4	**83.22%**	82.92%	80.55%	79.43%

The first column in each row is a cross-reference to the charts in Figure 4.7. The model parentheticals refer to the number of capsule dimensions. For example, I2 (8d) refers to model I2 which uses 8-dimensional capsules.

Chapter 5

Demonstrating the Irrelevance of Routing Mechanisms

Chapter 4 proposed a capsule design that used element-wise multiplication between capsules in subsequent layers and relied on backpropagation to do the work that prior capsule designs were relying on routing mechanisms for. This capsule design is referred to as homogeneous vector capsules (HVCs). This chapter directly extends the work of [1] and [2] on capsules specifically as applied to MNIST by applying HVCs to MNIST. The intent behind the design in this chapter is to show that the accuracies achieved in these prior works can be surpassed when using HVCs and without requiring the use of expensive routing algorithms.

5.1 Capsule Networks Applied to MNIST

Capsules have become a more active area of research since [2], which demonstrated near state-of-the-art performance on MNIST [152] classification (at 99.75%) by using capsules and a routing algorithm to determine which capsules in a previous layer feed capsules in the subsequent layer. MNIST is a classic image classification dataset of hand-written digits consisting of 60,000 training images and 10,000 validation images. Studying MNIST, due to the more highly structured content as compared to many other image datasets, allows for the use of more informed data augmentation techniques and, when using capsules, the ability to investigate the capsules' interpretability. In [6], the authors extended their work by conducting experiments with an alternate routing algorithm. Research in capsules has since focused mostly on various computationally expensive routing algorithms ([153][154]). By using the design presented in this chapter, the computationally expensive routing mechanisms of prior capsule work is avoided

The work presented in this chapter has been published in *Neurocomputing*, vol. 463, pp. 545–553, 2021 [19].

and the performance of [2] on MNIST is surpassed, all while requiring 5.5$\times$ fewer parameters, 4$\times$ fewer epochs of training, and using no reconstruction sub-network.

5.2 Experimental Design

5.2.1 Network Architecture

The starting point for the network design used for the experiments in this chapter is a conventional convolutional neural network following many widely used practices. These include stacked 3×3 convolutions, each of which with ReLU [155] activation preceded by batch normalization [139]. The common practice of increasing the number of filters in each subsequent convolutional operation relative to the previous one is also followed. Specifically, the first convolution uses 32 filters, and each subsequent convolution uses 16 more filters than the previous one. Additionally, the final operation before classification is to softmax the logits and to use categorical cross entropy for calculating loss.

One common design element found in many convolutional neural networks which was intentionally avoided was the use of any pooling operations. This work agrees with Geoffrey Hinton's assessment [156] of pooling (a method of down-sampling) as an operation to be avoided due to the information it "throws away". With the MNIST data being only 28×28, there is no need to down-sample. In choosing not to down-sample, there is a potential dilemma of how to reduce the dimensionality as operations descend deeper into the network. This dilemma is solved by choosing not to zero-pad the convolution operations and thus each convolution operation by its nature reduces the dimensionality by 2 in both the horizontal and vertical dimensions. This work deems choosing not to zero-pad as preferable in its own right in that zero padding effectively adds information not present in the original sample.

Rather than having a single monolithic design such that each operation in the network feeds into the next operation and only the next operation, the design employed creates multiple branches. After the first two sets of three convolutions, in addition to feeding to the subsequent convolution, the network also branches off the output to be forwarded on to an additional operation (detailed next). Thus, after all convolutions have been performed, the network has three branches.

1. The first of which has been through three 3×3 convolutions and consists of 64 filters each having an effective receptive field of 7×7 of the original image pixels.

2. The second of which has been through six 3×3 convolutions and consists of 112 filters each having an effective receptive field of 11×11 of the original image pixels.

3. The third of which has been through nine 3×3 convolutions and consists of 160 filters each having an effective receptive field of 15×15 of the original image pixels.

For each branch, rather than flattening the outputs of the convolutions into scalar neurons, the filters are instead transformed into vectors to form the first set pf capsules in the pairs of homogeneous vector capsules. This operation is represented by “Caps 1(a)”, “Caps 2(a)” and “Caps 3(a)” in Figure 5.1.

Then element-wise multiplication is performed on each of those capsules with a set of weight vectors (one for each class) of the same length. This results in *n*×*m* weight vectors where *n* is the number of capsules transformed from filter maps and *m* is the number of classes. Then summation is performed, per class (*m*), of each of the *n* vectors to form the second capsule in each pair of homogeneous vector capsules. After this, batch normalization is applied and then ReLU activation. The process elucidated in this paragraph is represented by “Caps 1(b)”, “Caps 2(b)” and “Caps 3(b)” in Figure 5.1.

After the pairs of capsules for each branch, the second capsule vector in each pair is reduced to a single value per class by summing the components of the vector. These values can be thought of as the branch-level logits.

Before classifying, the three branch-level sets of logits need to be reconciled with the fact that each image only belongs to one class. This is accomplished by stacking each class’s branch-level logits into vectors of length 3. Then, each vector is reduced by summation to a single value to form the final set of logits to be classified from. Figure 5.1 shows the high-level view of the entire network.

Chapter 4 detailed experiments with a variety of methods for constructing the first layer of capsules out of the preceding filter maps. This chapter experiments with 2 of these methods. The first method, referred to as *XY-Derived Capsules* (see Figure 4.5), constructs each capsule from each distinct feature map, whereas the second method, referred to as *Unbroken Z-Derived Capsules* (see Figure 4.4) constructs each capsule from each distinct x and y coordinate of the combination of all of the feature maps.

No weight decay regularization [157], a staple regularization method that improves generalization by penalizing the emergence of large weight values, was used. Nor was any form of dropout regularization [158][159], which are regularization methods designed to stop the co-adaptation of weights, used. Nor was a reconstruction sub-network, as in [2], used. These decisions were made in order to investigate the generalization properties of the chosen network design elements in the absence of other techniques associated with good generalization. In addition, no “routing” mechanism, as in [2] and [6], was used, preferring to rely on traditional trainable weights and backpropagation.

5.2.2 Merge Strategies

In [12] and [13], the authors chose to give static, predetermined weights to both output branches and then added them together. In this chapter, for both *XY-Derived Capsules* and *Unbroken Z-Derived Capsules*, three separate experiments of 32 trials each are conducted in order to investigate the effects of predetermined equal weighting of the branch outputs compared to learning the branch weights via backpropagation:

1. **Not learnable**. For this experiment, the three branches are merged together with equal weighting in order to investigate the effect of disallowing any one branch to have more impact than any other.

2. **Learnable with randomly initialized branch weights**. (Abbreviated as **Random Init.** subsequently.) For this experiment, randomly initialized weights are learned via backpropagation.

3. **Learnable with branch weights initialized to one**. (Abbreviated as **Ones Init.** subsequently.) For this experiment, the weights are learned via backpropagation. The difference with the **Random Init.** experiment being that in these experiments, the weights are initialized to 1. These experiments are conducted in addition to the **Random Init.** experiments in order to understand the difference between starting with random weights and starting with equal weights that are subsequently allowed to diverge during training.

5.2.3 Data Augmentation

Most (but not all [160][161]) of the state-of-the-art MNIST results achieved over the past decade have used data augmentation [162][159][44]. In addition to the network design, a major part of the work detailed in this chapter involves applying an effective data augmentation strategy that includes transformations informed specifically by the domain of the data. For example, excess rotation is avoided so that images of one class do not end up being more like a different class (e.g., rotating an image of the digit 2 by 180 degrees to create something that would more closely resemble a malformed 5). Nor was translation allowed such that the image content would move off of the canvas and perhaps cut off the left side of an 8 and thus create a 3. Choosing data augmentation techniques specific to the domain of interest is not without precedent (see for example [44] and [2], both of which used data augmentation techniques specific to MNIST).

By modern standards, in terms of dataset size, MNIST has a relatively low number of training images. As such, judicious use of appropriate data augmentation techniques is important for achieving a high level of generalizability in a given model. In terms of structure, hand-written digits show a wide variety in their rotation relative to some shared true “north”, position within the canvas, width relative to their height, and the connectedness of the strokes used to create them. Throughout training for all trials, every training image in every epoch was subjected to a series of four operations in order to simulate a greater variety of the values for these properties.

1. **Rotation**. First, each training image is randomly rotated by up to 30 degrees in either direction. Whether to actually apply this rotation was chosen by drawing from a Bernoulli distribution with probability p of 0.5 (a fair coin toss).

2. **Translation**. Second, each training image is randomly translated within the available margin present in that image. In [2], the authors limited their augmentation to shifting the training images randomly by up to 2 pixels in either or both directions. The limit of only 2 pixels for the translation ensured that the translation is label-preserving. As the MNIST training data has varying margins of non-digit space in the available 28$\times$28-pixel canvas, using more than 2 pixels randomly, would be to risk cutting off part of the digit and effectively changing the class of the image. For example, a 7 that was shifted too far left could become more appropriately classed as a 1, or an 8 or 9 shifted far enough down could be more appropriately classed as a zero. The highly structured nature of the MNIST training data allows for an algorithmic analysis of each image that will provide the translation range available for that specific image that will be guaranteed to be label-preserving. Figure 2.1 shows an example of an MNIST training image that has an asymmetric translation range that, as long as any translations are performed such that the digit part of the image is not moved by more pixels than are present in the margin, will be label-preserving. In other words, the specific training example shown in Figure 2.1 could be shifted by up to 8 pixels to the left or 4 to the right and up to 5 up or 3 down, and after doing so, all of the pixels belonging to the actual digit will still be in the resulting translated image. The amount within this margin to actually translate a training image was chosen randomly. Whether to translate up or down and whether to translate left or right were drawn independently from a Bernoulli distribution with probability p of 0.5 (a fair coin toss).

3. **Width**. Third, each training image’s width is randomly adjusted. MNIST images are normalized to be within a 20×20 central patch of the 28×28 canvas. This normalization is ratio-preserving, so all images are 20 pixels in the height dimension but vary in the number of pixels in the width dimension. This variance not only occurs across digits, but intra-class as well, as different peoples’ handwriting can be thinner or wider than average. In order to train on a wider variety of these widths, each image’s width is randomly compressed and then equal zero padding

is applied on either side, leaving the digit's center where it was prior. This was inspired by a similar approach adopted in [44]. In this work, each training sample is compressed within a range of 0–25%.

4. **Random Erasure**. Fourth, each training image is subjected to the random erasure (setting to 0) a 4×4 grid of pixels chosen from the central 20×20 grid of pixels in each training image. The X and Y coordinates of the patch were drawn independently from a random uniform distribution. This was inspired by the random erasing data augmentation method in [163]. The intention behind this method is to expose the model to a greater variety of (simulated) connectedness within the strokes that make up the digits. An alternative interpretation would be to see this as a kind of feature-space dropout.

5.2.4 Training

The training methodology used in chapter 4 is followed in the experiments detailed in this chapter and training is performed with the Adam optimizer [9] using all of the default/recommended parameter values, including the base learning rate of 0.001. Also, as in both chapter 4 and [2], the base learning rate is exponentially decayed. For the experiments in this chapter, which trained for 300 epochs, an exponential decay to the learning rate at a rate of 0.98 per epoch is applied.

Test accuracy was measured using the exponential moving average of prior weights with a decay rate of 0.999. [143]

5.3 Experimental Results

5.3.1 Individual Models

32 trials were executed for both of the capsule construction methods (*XY-Derived Capsules* and *Unbroken Z-Derived Capsules*) and each of the three merge strategies (see subsection 5.2.2). Each trial had weights randomly initialized prior to training and, due to the stochastic nature of the data augmentation, a different set of training images. As a result, training progressed to different points in the loss surface resulting in a range of values for the top accuracies that were achieved on the test set. See Table 5.1.

5.3.2 Ensembles

Ensembling multiple models together and predicting based on the majority vote among the ensembled models routinely outperforms the individual models' performances.

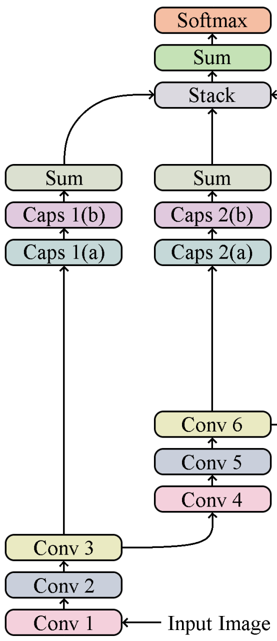


Figure 5.1: The proposed network from input to classification.

Table 5.1: Test Accuracy of the Individual Models

HVC Configuration	Experiment	Min	Max	Mean	Std. Dev.
Using XY-Derived Capsules	Not Learnable	99.71%	**99.79%**	0.997500	0.0002190
	Random Init.	99.72%	**99.78%**	0.997512	0.0001499
	Ones Init.	99.70%	99.77%	0.997397	0.0001885
Using Z-Derived Capsules	Not Learnable	99.74%	**99.81%**	0.997731	0.0001825
	Random Init.	99.73%	**99.80%**	0.997684	0.0002023
	Ones Init.	99.72%	**99.83%**	0.997747	0.0002509

In all cases, using the Z-Derived Capsules was superior to using the XY-Derived Capsules. For Z-Derived Capsules, no merge strategy produced statistically significantly superior test accuracy. For XY-Derived Capsules, the only statistically significant test accuracy result was that the Ones Init. strategy produced inferior accuracy. It should be noted that, though no strategy produced statistically significantly superior *test accuracies*, when branches were allowed to learn their weights, the weights learned were statistically significant. (Bold indicates a surpassing of the previous state of the art for individual models on MNIST.)

Ensembling can refer to either completely different model architectures with different weights or the same model architecture after being trained multiple times and finding different sets of weights that correspond to different locations in the loss surface. The previous state of the art of 99.82% was achieved using an ensemble of 30 different randomly generated model architectures [164]. The ensembling method used in this chapter uses the same architecture but with different weights. The majority vote of the predictions for all possible combinations of the weights produced by the 32 trials was calculated. See Table 5.2.

5.3.3 Branch Weights

What follows are visualizations of the final branch weights (after 300 epochs of training) for each of the branches in all 32 trials of the experiments wherein the branch weights were initialized to one for both HVC configurations.

Figure 5.2 shows that for all trials, the ratio between all three learned branch weights is consistent, demonstrating that the amount of contribution from each branch plays a significant role. Figure 5.3 shows a similar, though less pronounced consistency between the first branch's weight and the other two branches, however, branches two and three show no significant difference. Strikingly, using XY-Derived Capsules shows that branch three (the one having gone through all nine convolutions) has learned to be a more significant contributor. When using Z-Derived Capsules, branch one (the one having gone through only three convolutions) has learned to be a more significant contributor, but only slightly. Indeed, in the latter configuration, the contributions from all three branches are much more equal.

The experiments with randomly initialized branch weights showed the same relative weight of the branches for the magnitude of the weights learned. However, when the initial random branch weight was a negative number, it learned the negative value of that magnitude, and backpropagation took care of flipping the signs of weights as needed further up the network.

Because the models using Z-Derived Capsules are clearly superior to XY-Derived Capsules, unless otherwise stated, all analyses throughout the remainder of this work will restrict attention to these 96 trials, and thus, when the text reads "all 96 trials", it should be understood that this refers to all 96 trials *using Z-Derived Capsules*.

5.3.4 Troublesome Digits

Across all 96 trials there was total agreement on 9,912 out of the 10,000 test samples. (See Appendix A for the complete set of 88 digits that were predicted correctly by at least one model and incorrectly by at least one model.) There were only 14 digits that

Table 5.2: Test Accuracy of the Ensembles

HVC Configuration	Accuracy	Not Learnable	Random Init.	Ones Init.
Using XY-Derived Capsules	99.82%	1,183	2,069	1,292
	99.83%	4	21	19
	99.84%	0	0	1
	99.85%	0	0	0
	99.86%	0	0	0
	99.87%	0	0	0
Using Z-Derived Capsules	99.82%	121,731,146	554,104,195	1,279,126,811
	99.83%	17,746,467	148,600,238	426,947,909
	99.84%	1,587,152	17,319,668	34,635,994
	99.85%	89,384	533,318	1,113,217
	99.86%	4,029	1,226	9,920
	99.87%	184	0	64

Shown here are the number of ensembles that were generated that either matched the previous state of the art of 99.82% or exceeded it.

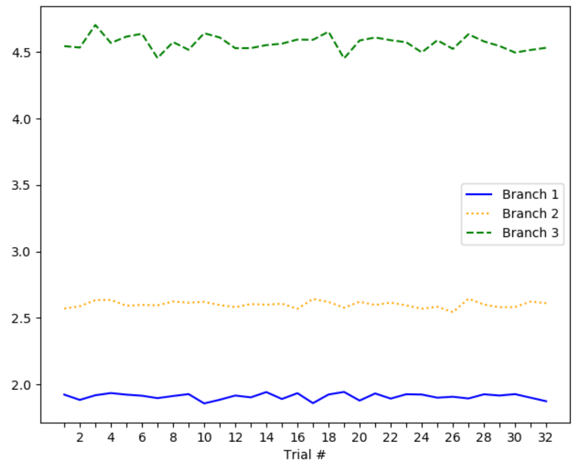


Figure 5.2: Final branch weights (after 300 epochs) for all 32 trials of the experiment using XY-Derived Capsules and for which the branch weights were initialized to one.

were misclassified more often than not across all 96 trials. This shows that although the accuracies of the models in the three experiments were quite similar, the different merge strategies of the three experiments did have a significant effect on classification. Across all 96 trials, only 5 samples were misclassified in all models. Those samples, as numbered by the order they appear in the MNIST test dataset (starting from 0) are 1901, 2130, 2597, 3422, and 6576 (see Figure 5.4).

5.3.5 MNIST State of the Art

Table 5.3 presents a comparison of previous state-of-the-art MNIST results for both single model evaluations and ensembles along with the results achieved in the experiments detailed in this chapter.

How long a model takes to train is an important factor to consider when evaluating a neural network. Indeed, it is an enabling factor during initial experimentation as faster training leads to a greater exploration of the design space. Table 5.4 presents a comparison of the number of epochs of training used in experiments for the results achieved in the networks shown in Table 5.3. Across all 96 trials, the design achieved peak accuracy in an average of 168 epochs, with a minimum peak achieved in 38 epochs and a maximum peak achieved at epoch 296. Since all trials were allowed to run for up to 300 epochs, that is the number reported in Table 5.4.

5.3.6 Interpreting Capsules' Dimensions

By adding a reconstruction sub-network to the overall network, it can be trained not just to classify the input digits, but also to reconstruct them. Then, by following the method in [2], the effects of perturbing individual dimensions of the second set of capsules in a pair of HVCs can be examined. The experiments using Z-Derived Capsules had capsules with 64, 112, and 160 dimensions. When perturbing only one of that many dimensions the changes to the resulting constructed images are very subtle. So another set of experiments with no branches, reconstruction, and using multiple 8-dimensional capsules for each distinct x and y coordinate of the feature maps were performed. By perturbing one of only eight dimensions the effects are more visible and allows for an interpretation of the meaning of values in the digits' capsules (see Table 5.5).

5.3.7 Ablation Experiments

In each of the following set of experiments, a comparison is made between the first 10 trials of the 32 trials for the Ones Init. merge strategy with 10 trials each of the additional experiments.

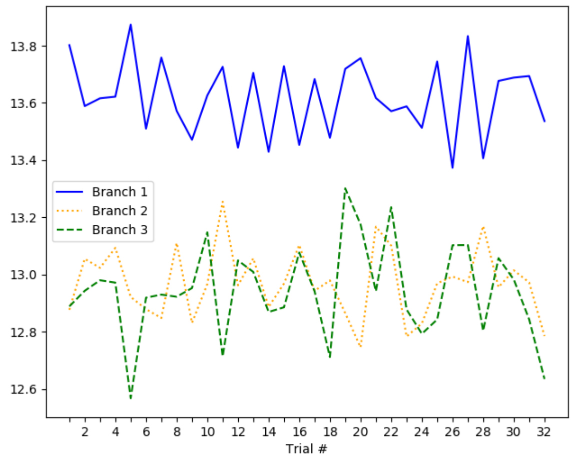


Figure 5.3: Final branch weights (after 300 epochs) for all 32 trials of the experiment using Z-Derived Capsules and for which the branch weights were initialized to one.

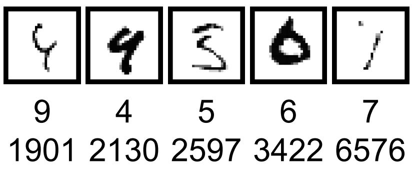


Figure 5.4: The Most Troublesome Digits

Table 5.3: Current and Previous MNIST State of the Art Results

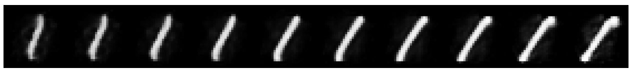
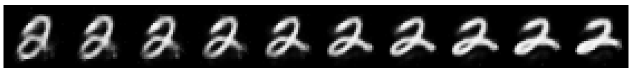
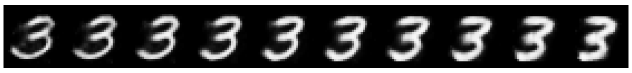
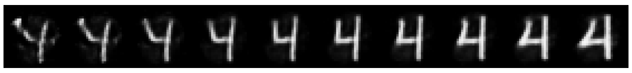
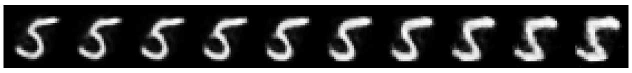
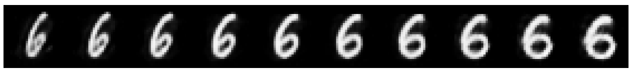
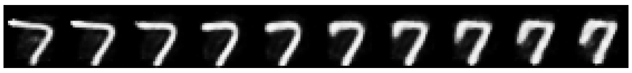
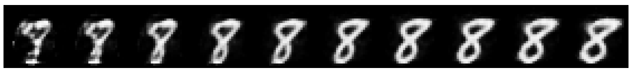
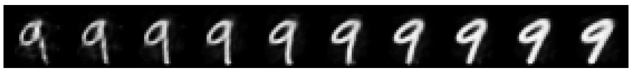
Paper	Year	Accuracy
Single Models		
Dynamic Routing Between Capsules[2]	2017	99.75%
Lets keep it simple, Using simple architectures to outperform deeper and more complex architectures[160]	2016	99.75%
Batch-Normalized Maxout Network in Network[161]	2015	99.76%
APAC:Augmented PAttern Classification with Neural Networks[162]	2015	99.77%
Multi-Column Deep Neural Networks for Image Classification[44]	2012	99.77%
Using the method proposed in this work	**2021**	**99.83%**
Ensembles		
Regularization of Neural Networks using DropConnect[159]	2013	99.79%
RMDL:Random Multimodel Deep Learning for Classification[164]	2018	99.82%
Using an ensemble of the method proposed in this work	**2021**	**99.87%**

Table 5.4: Epochs of Training

Paper	Epochs
Dynamic Routing Between Capsules[2]	1,200
APAC:Augmented PAttern Classification with Neural Networks[162]	15,000
Multi-Column Deep Neural Networks for Image Classification[44]	800
Regularization of Neural Networks using DropConnect[159]	1,200
RMDL:Random Multimodel Deep Learning for Classification[164]	120
The method proposed in this work	300

Neither [160] nor [161] report on how many epochs their designs were trained for.

Table 5.5: Dimensional Perturbations

Rightward tilt	
Top curl and height of lower loop	
Length of lower stroke	
Angle of the top part of one stroke	
Sharpness of the angle of the lower two curves	
Width of entire digit	
“Hook” in the initial part of the stroke	
Width of lower loop	
Lean angle	

Each row shows the reconstruction when one of the 8 dimensions in a digit’s capsule is perturbed by intervals of 0.1 in the range [-0.5, 0.5].

In [2], the authors used a custom loss function they called *margin* loss combined with the mean squared error of the difference between the input images and the result of reconstructing them. This work relied solely on categorical cross-entropy and did not use a reconstruction loss, as reconstruction adds a considerable number of parameters to the model (2.1M). Two additional sets of experiments were conducted to understand the effect of the chosen loss strategy (which used categorical cross-entropy and no reconstruction). The first used margin loss and reconstruction, and the second used categorical cross-entropy and reconstruction. There was no statistically significant difference among the three loss methods (see Table 5.6).

In order to understand the relative importance of using HVCs vs. a fully connected layer and 3 branches vs. a single branch, a series of experiments that ablated these components of the architecture was performed. Table 5.7 shows that HVCs are statistically significantly superior to a fully connected layer for both 1 and 3 branches and shows that 3 branches are superior to 1 branch for both HVCs and a fully connected layer.

In [2], the authors used translation, by a maximum of 2-pixels, as the only data augmentation method. In this work, a method for translating by up to the full margin available in any given direction was used. Then experiments were conducted comparing the effect of using only 2-pixel translation, only maximum margin translation, and the full suite of data augmentation methods. Using the full suite of data augmentation methods was shown to be statistically superior to either of the other two methods. Surprisingly, the experiments show that the 2-pixel translation method just barely crossed the threshold of being statistically significantly superior to the full margin translation method (see Table 5.8).

The result obtained by when using 2-pixel translation as the only data augmentation strategy allows for a direct comparison to the work of [2]. This work obtained the same level of accuracy as they did, but using 5.5$\times$ fewer parameters, 4$\times$ fewer training epochs, no reconstruction sub-network, and requiring no routing mechanism.

5.3.8 Additional Datasets

In order to better understand the effect of the Z-Derived HVCs and additional branches, additional sets of paired experiments were performed for several additional datasets wherein the first set of experiments in a pair used the network design as described in this chapter and the second set of experiments excluded the Z-Derived HVCs and additional branches. These second sets of experiments thus use a very small and typical convolutional neural network with 9 3$\times$3 convolutions and a final fully connected layer.

For MNIST and Fashion-MNIST the data augmentation strategy discussed in subsection 5.2.3 was used. For CIFAR-10 and CIFAR-100, this data augmentation strategy is inappropriate, so a very typical strategy of randomly flipping the images horizontally and applying random adjustments to brightness, contrast, hue, and saturation was used.

For all four datasets, the model that included Z-Derived HVCs and 3 branches achieved the higher mean accuracy with statistical significance (see Table 5.9).

The fact that the accuracies for Fashion-MNIST [38], CIFAR-10, and CIFAR-100 [40] were not competitive with current state of the art for those datasets is not especially surprising for several reasons. First, the network detailed in this chapter was designed for optimal accuracy on classification of Hindu-Arabic numerals which are highly structured and significantly simpler than the types of data in the other three datasets. Second, due to the significantly simpler nature of MNIST, the network used a small number of parameters (1.5M). For comparison, models competitive with state of the art for CIFAR-10 and CIFAR-100 use 10s and even 100s of millions of parameters. Finally, models competitive with state of the art for CIFAR-10 and CIFAR-100 use additional training data beyond the canonical set for each, and these experiments used no additional training data.

5.4 Summary

This chapter proposed using a simple convolutional neural network and established design principles as a basis for a network architecture. Then it presented a design that branched out of the series of stacked convolutions at different points to capture different levels of abstraction and effective receptive fields, and from these branches, rather than flattening to individual scalar neurons, used Homogeneous Vector Capsules instead.

Additionally investigated were three different methods of merging the output of the branches back into a single set of logits. Each of the three merge strategies generated models that could be ensembled to create new state of the art results.

Beyond the network architecture, this chapter proposed a robust and domain specific data augmentation strategy aimed at simulating a wider variety of renderings of the digits.

In doing this work, new MNIST state of the art accuracies for both a single model and an ensemble were established.

Table 5.6: Comparison of Loss Methods

Loss Method	Mean Accuracy	Std. Dev.
Categorical Cross-Entropy (no reconstruction)	99.7741%	0.000186455
Categorical Cross-Entropy (with reconstruction)	99.7740%	0.000245764
Margin Loss (with reconstruction)	99.7820%	0.000198997

Table 5.7: Comparison of Network Structures

Network Structure	Mean Accuracy	Std. Dev.
Using HVCs and 3 branches	99.7741%	0.000186455
Using HVCs and 1 branch	99.7140%	0.000185472
Using a fully connected layer and 3 branches	99.7550%	0.000111803
Using a fully connected layer and 1 branch	99.6870%	0.000141774

Table 5.8: Comparison of Data Augmentation Strategies

Data Augmentation Strategy	Mean Accuracy	Std. Dev.
Translation (full margin), rotation, width adjustment, and random erasure	99.7741%	0.000186455
Translation only (max. 2 pixels) (as in [2])	99.7570%	0.000195192
Translation only (using full margin)	99.7430%	0.000118743

Table 5.9: Effects of Z-Derived HVCs and Branching on Additional Datasets

MNIST

Network Architecture	Max	Mean	Std. Dev.	p-value
Z-Derived HVCs and 3 Branches	**99.81%**	**99.7741%**	0.0001864	1.824×10^{-7}
A Fully Connected Layer with 1 Branch	99.71%	99.6870%	0.0001417	

Fashion-MNIST

Network Architecture	Max	Mean	Std. Dev.	p-value
Z-Derived HVCs and 3 Branches	**93.89%**	**93.6850%**	0.0016391	5.243×10^{-6}
A Fully Connected Layer with 1 Branch	93.36%	93.0410%	0.0014616	

CIFAR-10

Network Architecture	Max	Mean	Std. Dev.	p-value
Z-Derived HVCs and 3 Branches	**89.23%**	**88.9290%**	0.0015514	0.020898
A Fully Connected Layer with 1 Branch	89.06%	88.7500%	0.0017515	

CIFAR-100

Network Architecture	Max	Mean	Std. Dev.	p-value
Z-Derived HVCs and 3 Branches	**64.15%**	**63.8260%**	0.0026743	6.859×10^{-6}
A Fully Connected Layer with 1 Branch	62.96%	62.3760%	0.0035046	

MNIST results come from the same experiments detailed in Table 5.7 and are repeated here to facilitate ease of comparison. 10 trials of each unique type of experiment were conducted in order to establish statistical significance.

Chapter 6

Case Study: Creating a micro-PCB Dataset

Prior studies in capsule networks ([1][2][6]) have demonstrated their ability to capture the equivariant properties of the source material. Printed Circuit Boards (PCBs) are like the human face in that, for a given PCB, the individual elements (such as capacitors, resistors, and integrated circuits (ICs)) are not present invariantly relative to one another, but rather at very specific locations (equivariantly) relative to one another. Compared to the human face, even on small PCBs, there are a greater number of features with greater similarity to one another (some modern small surface-mounted capacitors and resistors are nearly indistinguishable from one another, whereas eyes, noses, and mouths are quite distinctive from one another). This chapter presents a dataset consisting of high-resolution images of 13 micro-PCBs captured in various rotations and perspectives relative to the camera, with each sample labeled for PCB type, rotation category, and perspective categories. Then presented is the design and results of experimentation on combinations of rotations and perspectives used during training and the resulting impact on test accuracy. The results of the experimentation show when and how well data augmentation techniques are capable of simulating rotations vs. perspectives not present in the training data. All experiments are performed using CNNs with and without homogeneous vector capsules (HVCs) and investigate and show the capsules' ability to better encode the equivariance of the sub-components of the micro-PCBs.

6.1 Image Acquisition

A total of 8,125 images of the 13 micro-PCBs (see Figure 6.2) were captured for the dataset using a Sony SLT-A35, 16 Megapixel DSLR Camera, in Advanced Auto mode

The work presented in this chapter has been published in *Intelligent Decision Technologies*, vol. 238, pp. 209–219, 2021 [20].

and using a polarization filter. After cropping the excess area around the micro-PCBs in each image, the average size of all images is 1949$\times$2126 (width$\times$height). The micro-PCBs were captured in 25 different positions relative to the camera (see Figure 6.1) under ideal lighting conditions using (4) 85-Watt CFL Full Spectrum 5500K color bulbs each producing approximately 5,000 lumens.

In each position, each micro-PCB was captured in 5 different rotations (see Figure 6.3 and Figure 6.4). This creates 125 unique orientations of each micro-PCB relative to the camera. Each unique orientation was captured 4 times and coded for training and then another micro-PCB of the same make and model was captured once and coded for testing. Thus, no micro-PCB that is used in training is the same that is used in testing. Although the micro-PCBs coded for training are nearly identical to those coded for testing, very subtle differences exist in some cases (see Figure 6.5). In total, each micro-PCB in the dataset has 500 training images and 125 test images, creating an overall train/test split of 6,500/1,625.

The micro-PCBs being placed in 25 different positions in the capture surface results in the creation of 25 unique perspectives of each micro-PCB relative to the camera. This work refers to the position directly under the camera as the neutral perspective, the 8 positions directly adjacent to the neutral position as “near” perspectives and the outer 16 positions as “far” perspectives. To fully distinguish the 25 perspectives, when looking down from the camera’s position to the capture surface, this work refers to those that are to the left or above the camera as “negative” and those that are to the right or below the camera as “positive”. In each perspective, each micro-PCB was rotated across 5 rotations manually without attempting to place them in any exact angle. Instead, each micro-PCB was placed (1) straight, which this work refers to as the neutral rotation, (2–3) rotated slightly to the left and right, which this work refers to as the left shallow and right shallow rotations respectively, and (4–5) rotated further to the left and right, which this work refers to as the left wide and right wide rotations respectively.

The goal of computer vision applications is to learn true representations of the objects and not merely to memorize pixel intensities and locations. The manual placement of the micro-PCBs as well as placing them in rotation without measurement helps to facilitate learning the true representation by introducing small perturbations in the positions of the micro-PCBs.

After image acquisition, an edge detection algorithm was used to detect the left and right edges of each image. Using the left edge, the angle of each micro-PCB relative to an ideal neutral was computed. See Table 6.1 for statistics regarding these angles. The distance of the left edge to the right edge at the bottom and top of each image was then measured and between the two distances was created. This ratio is representative of the true perspective along the y-axis. See Table 6.2 for statistics regarding these ratios. The presence of various connectors on the top edge of the micro-PCBs made algorithmically determining an accurate top edge of the micro-PCBs impossible, so ratios to be representative of the true perspective along the x-axis are presented. However, a reasonable estimate can be calculated using the corresponding ratio for the y-axis multiplied by the ratio of an image's width to its height.

6.2 Experimental Design

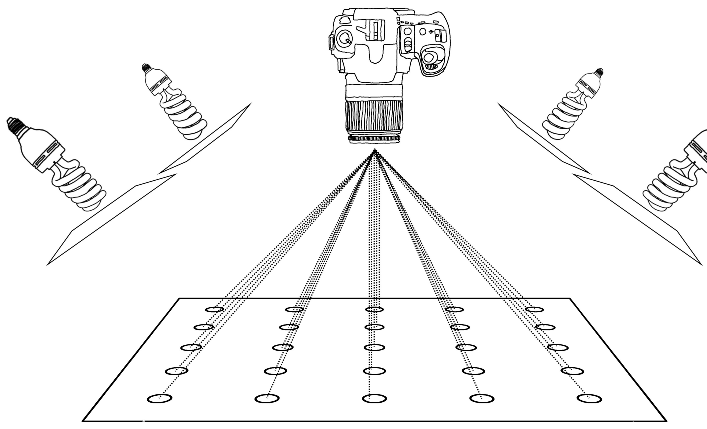
Chapter 4 detailed experiments using a simple monolithic CNN. In those experiments, a baseline model that used the common method of flattening the final convolution operation and classifying through a layer of fully connected scalar neurons was compared with a variety of configurations of homogeneous vector capsules (HVCs). That chapter demonstrated that classifying through HVCs is superior to classifying through a layer of fully connected scalar neurons on three different datasets with differing difficulty profiles. This chapter extends that work to include this micro-PCB dataset. In all experiments performed, comparisons are made between classifying through a fully connected layer of scalar neurons to the best performing HVC configuration for the simple monolithic CNN in chapter 4. In these experiments, the network using a fully connected layer is labeled M1 and the network using HVCs is labeled M2.

In addition to investigating the impact of HVCs on this dataset, this chapter investigates (a) the ability of the networks to accurately predict novel rotations and perspectives of the micro-PCBs by excluding training samples with similar rotations and perspectives and (b) the ability of data augmentation techniques to mimic the excluded training samples.

Table 6.3 shows rotations and perspectives that were used during training for experiments E1-E9. Testing always included all images from all rotations and perspectives. For these experiments, data augmentation techniques were *not* used to simulate the rotations and perspectives that were excluded during training. Table 6.4 and Table 6.5 show, for experiments A1-A16, both which rotations and perspectives were used during training, as well as whether data augmentation techniques were used to simulate the excluded rotations, excluded perspectives, or both. Again, testing always included all images from all rotations and perspectives.

For all experiments, including those in which data augmentation techniques were not used *to simulate the rotations and perspectives that were excluded*, a small amount of random translation was applied during training in order to encourage *translational invariance*. This translation was limited to no more than 5% in either or both of the x and y directions.

Figure 6.1: A depiction of the image acquisition environment.



Each micro-PCB were captured in 25 different positions in a 5×5 grid on the capture surface such that each position was 6 inches from its horizontal and vertical neighbors. The camera was positioned 16 inches directly above the central position in the grid. The light sources were placed 16 inches outside of the grid in the four corners of the grid and 32 inches above the capture surface, each with diffusion material directly in front of the bulbs.

Table 6.1: Angles of Rotated Images

Position	Min	Mean	Std. Dev.	Max
Left Wide	10.43	21.31	4.50481	32.78602043
Left Shallow	4.23	12.39	3.59036	23.72892221
Neutral	0	2.475	2.04823	12.76094982
Right Shallow	0.16	14.73	4.46218	31.52155152
Right Wide	0.76	24.31	5.27139	42.84832537

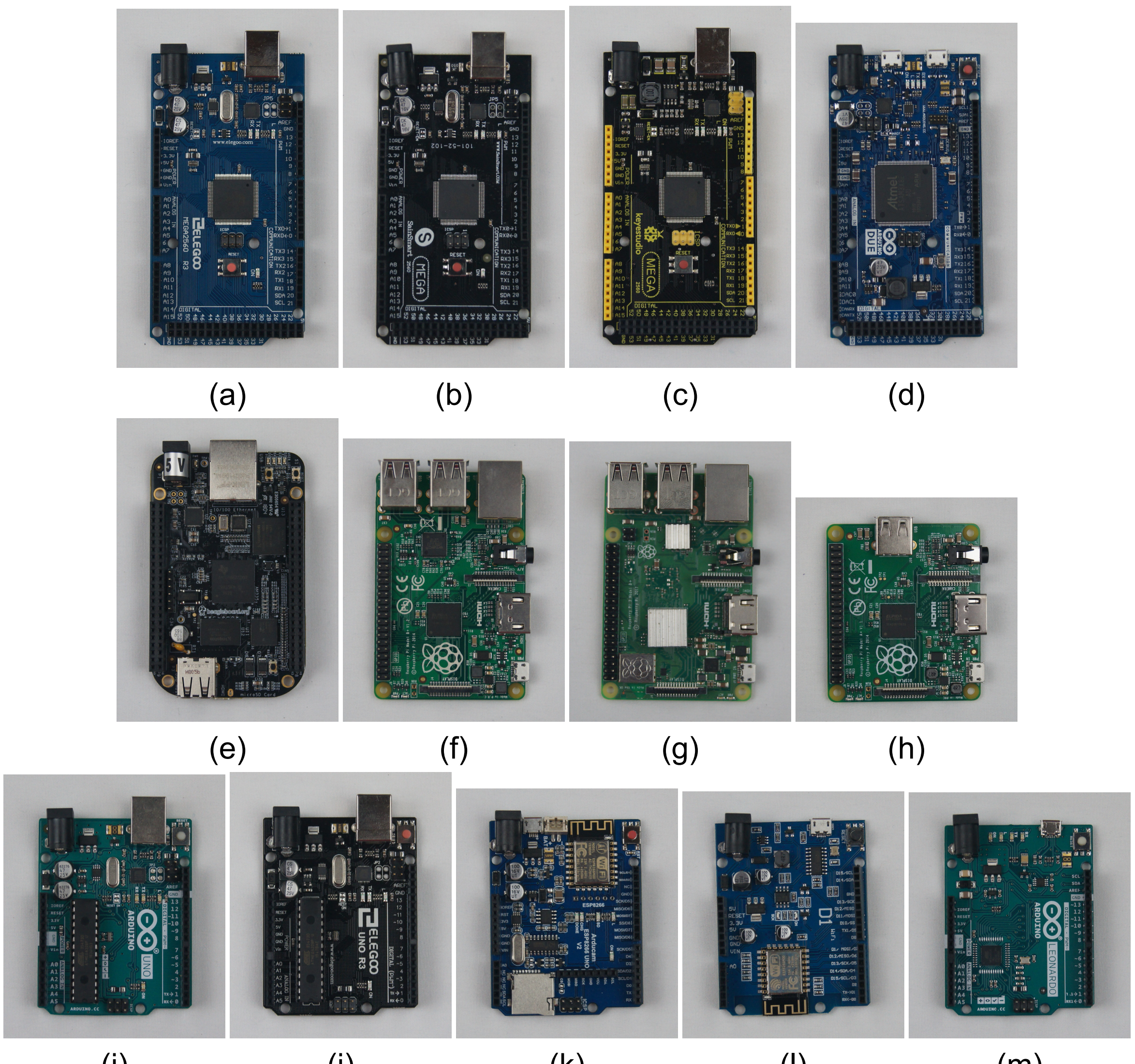
Values are in degrees and represent the absolute value of the deviation from a true neutral rotation.

Table 6.2: Ratios of micro-PCB Width Differences

Position	Min	Mean	Std. Dev.	Max
Negative Far	0	12.71%	6.27130	48.18%
Negative Near	0	7.94%	5.05283	29.66%
Neutral	0	4.40%	3.50910	20.90%
Positive Near	0	7.09%	3.59487	26.76%
Positive Far	0	11.45%	4.36561	23.13%

Edge detection algorithms were used to identify the left and right edges of the micro-PCBs. The values here are absolute values of ratios of the width between the detected edges at the bottom and top of the image.

Figure 6.2: The 13 micro-PCBs for which images were acquired.



(a-c) Arduino Mega 2560 from 3 different manufacturers. (d) Arduino Due. (e) Beaglebone Black. (f) Raspberry Pi 1 B+. (g) Raspberry Pi 3 B+. (h) Raspberry Pi A+. (i-j) Arduino Uno from 2 different manufacturers. (k) Arduino Uno WiFi Shield. (l) Arduino Uno Camera Shield. (m) Arduino Leonardo.

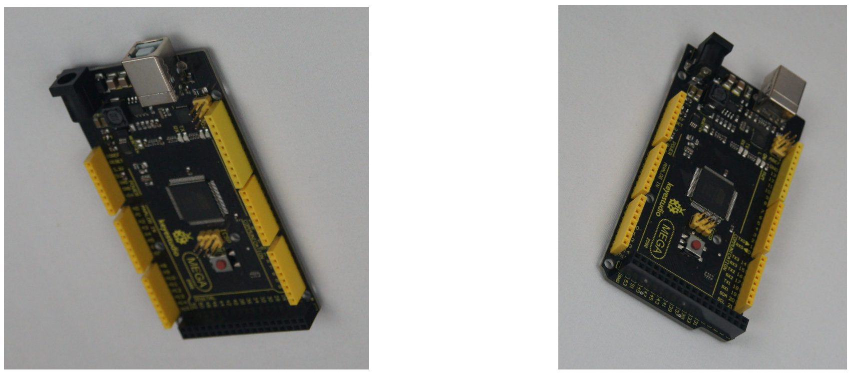


Figure 6.3: Examples of one of the manufacturer's Arduino Mega 2560 captured from two different extreme perspectives on the capture surface relative to the camera.

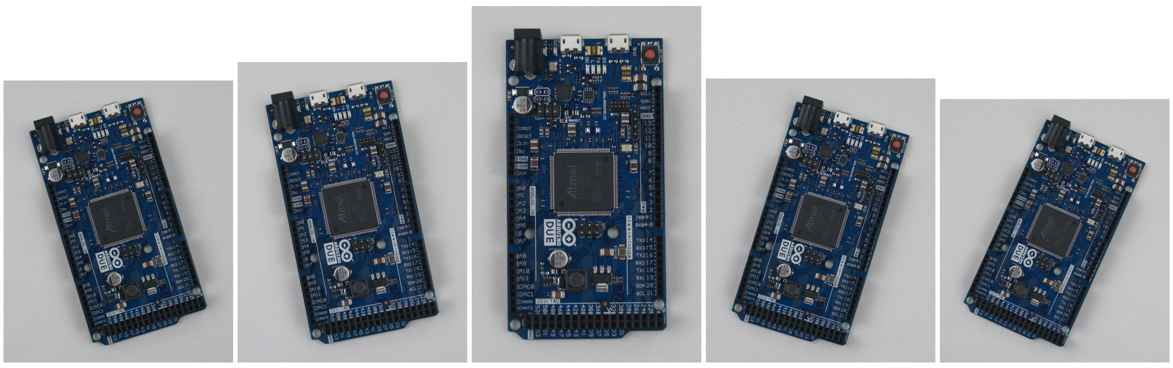
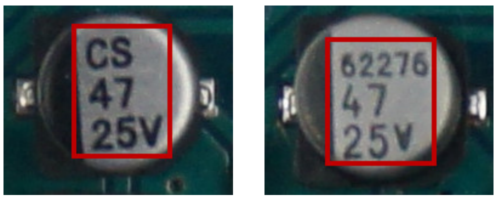
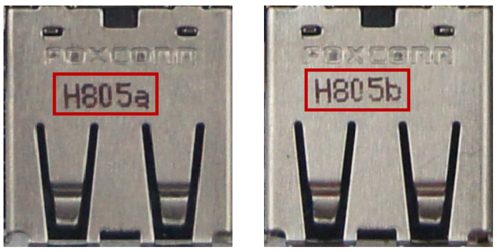


Figure 6.4: Examples of the Arduino Due captured in the neutral perspective position from the five different rotations each micro-PCB was captured from.



(a) Printing on the capacitors present on one manufacturer's Arduino Uno that differs between the micro-PCB coded for training and the one coded for testing.



(b) Printing on the USB receiver present on the Beaglebone Black that differs between the micro-PCB coded for training and the one coded for testing.

Figure 6.5: Examples of inconsequential printing differences present on the micro-PCBs that were coded for training vs. those coded for testing.

Table 6.3: Experimental Design for Experiments Excluding Rotations and Perspectives

	Train Rotations				Train Perspectives			
Experiment	Left Wide	Left Shallow	Right Shallow	Right Wide	Neg. Far	Neg. Near	Pos. Near	Pos. Far
E1	✓	✓	✓	✓	✓	✓	✓	✓
E2	✓	✓	✓	✓		✓	✓	
E3	✓	✓	✓	✓				
E4		✓	✓		✓	✓	✓	✓
E5		✓	✓			✓	✓	
E6		✓	✓					
E7					✓	✓	✓	✓
E8						✓	✓	
E9								

Checkmarks indicate training included images for the specified rotations or perspectives.

Table 6.4: Experimental Design for Experiments using Data Augmentation to Simulate the Excluded Rotations and Perspectives

	Train Rotations				Train Perspectives			
Experiment	Left Wide	Left Shallow	Right Shallow	Right Wide	Neg. Far	Neg. Near	Pos. Near	Pos. Far
A1	✓	✓	✓	✓		✓	✓	
A2	✓	✓	✓	✓				
A3		✓	✓		✓	✓	✓	✓
A4		✓	✓			✓	✓	
A5		✓	✓			✓	✓	
A6		✓	✓			✓	✓	
A7		✓	✓					
A8		✓	✓					
A9		✓	✓					
A10					✓	✓	✓	✓
A11						✓	✓	
A12						✓	✓	
A13						✓	✓	
A14								
A15								
A16								

Table 6.5: Experimental Design for Experiments using Data Augmentation to Simulate the Excluded Rotations and Perspectives

Experiment	Augment Excluded Rotations	Augment Excluded Perspectives
A1		✓
A2		✓
A3	✓	
A4	✓	✓
A5	✓	
A6		✓
A7	✓	✓
A8	✓	
A9		✓
A10	✓	
A11	✓	✓
A12	✓	
A13		✓
A14	✓	✓
A15	✓	
A16		✓

Checkmarks in the second and third column indicate if data augmentation was used to simulate the excluded rotations, perspectives, or both. Checkmarks in the following columns indicate training included images for the specified rotations or perspectives.

6.3 Experimental Results

6.3.1 Experiments E1-E9

Table 6.6 and Table 6.7 show the results from experiments E1-E9 wherein data augmentation was not used to simulate the excluded rotations and perspectives.

These experiments showed that model M2 (using HVCs) is superior to model M1 (using a fully connected layer) for all 9 experiments, though this was statistically significant for only 8 of the 9 experiments. Both models M1 and M2 were better able to cope with excluded rotations than excluded perspectives. This is not especially surprising given that rotation is an affine transformation whereas perspective changes are not. For both M1 and M2, accuracy was especially poor when excluding all non-neutral perspectives (experiments E3, E6, and E9) to the point that the accuracy for model M1 for experiments E3 (including all rotation variants) and E9 (including no rotation variants) was equivalent to that of random guessing. A surprising result is that model M1 for experiment E6 (which included only the near rotation variants) had a mean accuracy that was approximately twice as accurate as either E3 or E9 (for model M1). One of the trials for model M1 experiment E6 achieved an accuracy approximately four times higher than the mean accuracy. Indeed, the standard deviation of the trials for model M1 experiment E6 was more than an order of magnitude higher than the standard deviations of either E3 or E9 (for model M1). Most surprising for model M1 was that the maximum accuracy achieved by one trial of E8 (which included no rotation variants and only near perspective variants) was 100%. There is not enough data to come to any firm conclusion regarding this anomaly, but a reasonable hypothesis is that the loss manifold for model M1 for the training data used in E8 contains many global minima and that the less structured nature of fully connect layers relative to HVCs allows for greater exploration of the loss manifold. The fact that E8 was the one experiment for which model M2 was not shown to be statistically significantly superior to model M1 would seem to support this.

6.3.2 Experiments A1-A16

Table 6.9 and Table 6.10 show the results from experiments A1-A16 wherein data augmentation was used to simulate the excluded rotations and perspectives.

These experiments showed that model M2 (using HVCs) is superior to model M1 (using a fully connected layer) for 11 out of 16 experiments, though this was statistically significant for only 8 of the 16 experiments. Model M1 is superior twice with statistical significance. The lack of statistical significance in 8 out of the 16 experiments is due to the high variance across trials for the experiments using model M1. A large contributor to

Table 6.6: Results of Experiments Excluding Rotations and Perspectives using Model M1 (Fully Connected)

Experiment	Mean	Max	Std. Dev.
E1	93.28%	96.90%	0.02025
E2	92.30%	92.30%	0
E3	9.78%	11.76%	0.01415
E4	92.30%	92.30%	2.58×10^{-8}
E5	25.14%	35.10%	0.06826
E6	18.10%	71.67%	0.19831
E7	22.89%	41.01%	0.10565
E8	44.98%	100.00%	0.44777
E9	10.10%	13.05%	0.01831

Table 6.7: Results of Experiments Excluding Rotations and Perspectives using Model M2 (HVCs)

Experiment	Mean	Max	Std. Dev.	p-value
E1	**99.45%**	100.00%	0.01117	0.00033
E2	**98.84%**	99.14%	0.00244	6.68×10^{-12}
E3	**39.45%**	42.55%	0.02703	2.11×10^{-8}
E4	**98.92%**	99.45%	0.00533	3.05×10^{-9}
E5	**89.74%**	91.63%	0.02087	3.71×10^{-8}
E6	**34.29%**	42.73%	0.05567	0.02304
E7	**78.62%**	81.77%	0.02368	2.95×10^{-6}
E8	**46.48%**	50.37%	0.03487	0.91692
E9	**27.61%**	36.45%	0.05395	0.00013

In all cases, model M2 achieved a higher mean accuracy. 5 trials of each of experiments E1-E5, E7, and E9 were conducted. 10 trials of E6 were conducted in order to establish statistical significance. After 10 trials of experiment E8, the higher mean of accuracy of model M2 was not shown to be statistically significant.

the higher variances is the unusual number of trials using model M1 that achieved 100% test accuracy. The present theoretical understanding of non-convex optimization beyond its NP-hardness is limited to a small number of special cases ([165][166][167][168]), none of which apply to modern CNNs for image classification. As such, only conjecture based on observation regarding this phenomenon can be offered. This conjecture being that the loss manifold of the test data has an unusually large number of local minima near the global minimum, and further that using an adaptive gradient descent method along with a fully connected layer for classification allows training to proceed in a highly exploratory manner (as opposed to an exploitative manner). Of the experiments that had trials which achieved 100% test accuracy with model M1, none of them included the far perspectives in their training and only 2 out of the 6 included even the near perspectives (A11 and A12), neither of which included any rotations. 3 out of the 6 (A2, A7, and A11) fully simulated the excluded rotations and perspectives with data augmentation. One experiment (A12) had a trial that achieved 100% test accuracy with model M1 without attempting to simulate the excluded far perspectives. One experiment (A8) had a trial that achieved 100% test accuracy with model M1 without attempting to simulate either the excluded far or near perspectives. And one experiment (A16) had a trial that achieved 100% test accuracy with model M1 without attempting to simulate either the excluded wide or shallow rotations.

6.3.3 Comparing Experiments E1-E9 with Experiments A1-A16

Table 6.11 shows a comparison of mean accuracies achieved by the trials of experiments E1-E9 with those of experiments A1-A16, grouping the experiments together by the rotations and perspectives that were excluded during training. Not surprisingly, in all cases, the experiments that included data augmentation to replace some or all of the excluded samples achieved higher mean accuracy. In 9 out of the 16 comparisons, the superiority was statistically significant. Of those that did not produce a statistically significant difference, (1) experiments E2 and A1 each excluded only the far perspectives, (2) experiments E4 and A3 each excluded only the wide rotations, (3) experiments E8, A12, and A13 excluded all rotations and the far perspectives, and (4) experiments E9, A14, A15, and A16 excluded all rotations and all perspectives. In 6 out of 7 these comparisons, the data augmentation was attempting to synthesize excluded perspectives. As perspective warp is a non-affine transformation, it makes sense that synthesizing excluded perspectives with it meets with limited (but not no) success.

Table 6.8: Comparing Results with and without Augmentation of Excluded Rotations and Perspectives

Model	Experiment	Mean Accuracy	Experiment	Mean Accuracy	p-value
M2	E2	98.84%	A1	**99.27%**	0.05656
M2	E3	39.45%	A2	**48.90%**	0.02243
M2	E4	98.92%	A3	**99.40%**	0.11293
M2	E5	89.74%	A4	**96.37%**	0.00016
M2	E5	89.74%	A5	**94.52%**	0.00727
M2	E5	89.74%	A6	**92.51%**	0.04678
M1	E6	18.10%	A7	**83.44%**	2.46×10^{-5}
M1	E6	18.10%	A8	**58.13%**	0.01604
M2	E6	34.29%	A9	**45.04%**	0.00358
M2	E7	78.62%	A10	**93.85%**	1.05×10^{-6}
M1	E8	44.98%	A11	**98.03%**	0.02226
M1	E8	44.98%	A12	**72.77%**	0.26219
M2	E8	46.48%	A13	**49.68%**	0.32177
M2	E9	27.61%	A14	**35.07%**	0.31777
M2	E9	27.61%	A15	**34.29%**	0.09152
M1	E9	10.10%	A16	**40.46%**	0.05605

Horizontal lines in this table are used to group together experiments E1-E9 with their counterpart experiments A1-A16 based on the samples that were used during training. For example, E5 and A4-A6 were all trained on the subset of training samples that excluded the wide rotations and the far perspectives.

Table 6.9: Results of Experiments using Data Augmentation to Simulate the Excluded Rotations and Perspectives using Model M1 (Fully Connected)

Experiment	Mean	Max	Std. Dev.
A1	92.30%	92.30%	0
A2	35.73%	100.00%	0.39620
A3	90.76%	92.30%	3.44×10^{-2}
A4	22.73%	26.17%	0.03215
A5	23.82%	31.40%	0.05243
A6	20.87%	29.62%	0.05037
A7	**83.44%**	100.00%	0.15976
A8	**58.13%**	100.00%	0.37197
A9	10.00%	11.45%	0.00899
A10	24.14%	29.43%	0.04385
A11	**98.03%**	100.00%	0.04406
A12	**72.77%**	100.00%	0.39731
A13	23.72%	51.54%	0.20104
A14	21.01%	61.82%	0.22826
A15	18.88%	53.63%	0.19435
A16	**40.46%**	100.00%	0.42248

Table 6.10: Results of Experiments using Data Augmentation to Simulate the Excluded Rotations and Perspectives using Model M2 (HVCs)

Experiment	Mean	Max	Std. Dev.	p-value
A1	**99.27%**	99.69%	0.00357	8.44×10^{-11}
A2	**48.90%**	58.56%	0.06991	0.48483
A3	**99.40%**	99.82%	0.00284	0.00052
A4	**96.37%**	97.48%	0.00785	2.95×10^{-11}
A5	**94.52%**	95.75%	0.02143	2.93×10^{-9}
A6	**92.51%**	93.97%	0.01614	1.53×10^{-9}
A7	36.44%	44.58%	0.06076	0.00027
A8	40.87%	44.95%	0.03404	0.33187
A9	**45.04%**	53.51%	0.05463	6.05×10^{-7}
A10	**93.85%**	94.70%	0.01037	5.33×10^{-10}
A11	49.62%	62.87%	0.10311	1.10×10^{-5}
A12	58.00%	69.77%	0.08962	0.44100
A13	**49.68%**	57.27%	0.08771	0.02941
A14	**35.07%**	39.96%	0.05824	0.21863
A15	**34.29%**	41.56%	0.05613	0.12698
A16	36.11%	47.48%	0.06581	0.82585

In 11 out of 16 experiments, model M2 achieved a higher mean accuracy. 5 trials of all experiments were conducted. In all but 4 experiments, there was greater variance across trials with model M1.

6.3.4 Experiments Including All Rotations and Perspectives During Training and Using Data Augmentation to Synthesize Variations of Those Rotations and Perspectives

Table 6.11 shows the results of a final set of experiments wherein all training samples were included, and for each training sample, a range of rotation and perspective warp augmentations were applied. Each training sample was subjected to a random rotation drawn from a normal distribution with a zero mean and the standard deviation for its rotational label (see Table 6.1). For example, if the training sample was labeled as left wide, it was rotated by a random amount chosen from a normal distribution with a mean of 0 degrees and a standard deviation of 4.50481 degrees. Perspective warp transformations were applied using the same procedure with standard deviations for its perspective label (see Table 6.2).

As is shown in Table 6.11, model M1, using a fully connected layer, achieved a mean accuracy of 94.52%, surpassing 15 out of 16 of the experiments detailed in Table 6.9 (for model M1). Experiment A11 performed slightly better, but it should be noted that that experiment also used data augmentation for both rotation and perspective warp, with the difference being that experiment A11 excluded all of the rotated training samples, and the far perspectives. Model M2, using HVCs, achieved a mean accuracy of 99.06%, surpassing 14 out of 16 of the experiments detailed in Table 6.10 (for model M2). Experiment A1 and A3 performed slightly better, but once again, it should be noted that those experiments also used data augmentation that covered the rotations and perspectives of the training samples that were excluded. Model M2's maximum accuracy was 100% surpassing the maximum accuracy of all experiments (for model M2) A1-A11.

6.4 Regarding Synthesizing Alternate Perspectives

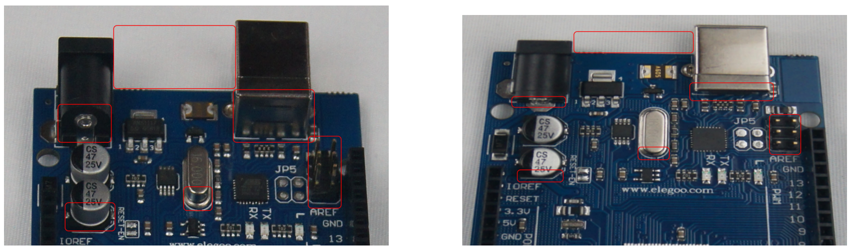
The experiments detailed in this chapter demonstrate that using data augmentation to supply excluded and/or greater variations of rotations works better than data augmentation to supply excluded and/or greater variations of perspective. Indeed, rotation is generally considered a staple data augmentation technique irrespective of the subject matter. As mentioned earlier, this is because rotation is an affine transformation and as such, rotating the image produces the same result as rotating the capturing apparatus would have. Perspective warp of captured *3-D* subject matter is not affine. Moving the capturing apparatus to a different perspective relative to 3-D subject matter will produce larger or smaller patches of components that extend into the third dimension (see Figure 6.6). While the simulation of perspective differences did improve upon

Table 6.11: Results of Experiments with no Exclusions and Using Data Augmentation

Model	Mean	Max	Std. Dev.
M1 (Fully Connected)	94.52%	96.90%	0.02009
M2 (Capsules)	**99.06%**	**100.00%**	0.01862

10 trials were conducted for each model. M2, using homogeneous vector capsules, is shown to be superior with a p-value of 5.52×10^{-5}.

Figure 6.6: A Comparison of Actual vs. Simulated Perspective



The image on the left shows a micro-PCB actually captured in a negative far position. The image to the right shows the same micro-PCB captured from a neutral position and then subjected to an (exaggerated) perspective warp. The annotating boxes show important differences between an image actually captured from a perspective position vs. simulating a perspective.

accuracy as compared to when not using such, the results of the experiments detailed in this chapter show that, when possible, capturing a variety of perspectives during training is the best avenue for generating higher accuracy during subsequent evaluations that could include analogous perspective varieties.

6.5 Summary

The results of the experiments performed and elucidated in this chapter show that (1) classification of these micro-PCBs from novel rotations and perspectives is possible, but, in terms of perspectives, better accuracy is achieved when networks have been trained on representative examples of the perspectives that will be evaluated. (2) That, even though perspective warp is non-affine, using it as a data augmentation technique in the absence of training samples from actually different perspectives is still effective and improves accuracy. (3) And that using homogeneous vector capsules (HVCs) is superior to using fully connected layers in convolutional neural networks, especially when the subject matter has many sub-components that vary equivariantly (as is the case with micro-PCBs), and when using the full training dataset and applying rotational and perspective warp data augmentation the mean accuracy of the network using HVCs is 4.8% more accurate then when using a fully connected layer for classification.

The experiments in this chapter have been limited to using the same simple monolithic CNN as presented in chapter 4, with and without HVCs. The focus of these experiments was on the excluding rotations, excluding perspectives, and which data augmentation methods could compensate for these exclusions. Given the novel nature of this dataset, important future work should include all of these factors, but additionally as applied to a variety of neural network architectures, such as Inception v3.

Chapter 7

Towards an Analytical Definition of Sufficient Data

Chapter 3 demonstrated the trend towards massively increasing the training data used in neural network for image classification research. As the amount of additional training data has continued to grow, the accuracy being achieved on the Imagenet-1K [89] evaluation benchmark seems to be approaching a horizontal asymptote. This is supported by [95], wherein the authors observed that for more than two orders of magnitude, compute budget and accuracy followed a power-law, and at the high end of the compute budget, the largest models were not tending toward perfect accuracy, suggesting that a model with infinite capacity would achieve less than perfect accuracy. Knowing that perfect accuracy is unachievable, at least in the case of the Imagenet-1K benchmark, motivates a study of the sufficiency of data required to achieve a desired accuracy. In addition, as was noted in the conclusion of chapter 3, achieving state-of-the-art accuracy, again in the case of the Imagenet-1K benchmark, is prohibitively expensive in terms of the training resources required as well as requiring a model comprised of 2.4B parameters, which is excessively large for on-device or real-time inferences on presently available consumer hardware. All this is to say that simply aiming for the highest possible accuracy should not precede the definition of the problem the model will be applied to. Once accuracy and performance requirements are defined, then there exists an unknown amount of training data required to meet those requirements. In this chapter, a step is made towards being able to identify the data sufficient to meet stated requirements *a priori* to experimentation.

The experimental results of the previous chapter demonstrated that for the micro-PCB dataset, a portion of the data could be excluded from training and models could achieve comparable accuracy when using data augmentation to simulate the particular attributes of the excluded data, particularly when those models used HVCs. The micro-

The work presented in this chapter has been submitted for publication.

PCB dataset consists of images coded for rotation and perspective and that allowed for data augmentation techniques to simulate known excluded rotations and perspectives.

The next step in this line of research was to investigate datasets that consist of images that are not coded in any way beyond their class membership. When no coding is available, the goal then becomes to extract metrics from the image data that can be acted upon.

Since the primary goal in classification is to distinguish between classes, finding metrics that are indicative of those classes, or more specifically, indicative of those classes relative to the other classes, is a corollary goal. The main hypothesis being tested in this chapter is that there exists a distance metric that can be leveraged during the training of a CNN in order to improve or at least maintain testing accuracy by using that metric to *exclude* a portion of the training data.

An obvious candidate for a distance metric is Euclidian distance from the centroid for the class. An obvious problem with Euclidian distance is the curse of dimensionality that results in near uniformity of distance from the centroid among samples when measuring distance in a large number of dimensions. For example, take a small 3-class dataset, consisting of 10 images in each of the classes cat, dog, and truck derived from a Google Image search for each of the 3 terms “cat”, “dog” and “truck” as shown in Figure 7.1. These images are 180 pixels square and made up of 3 color channels. Thus, the dataset is 97,200-dimensional. Fortunately, there exists a class of non-linear dimensionality reduction techniques that can learn to represent high-dimensional data in lower dimensions. These include t-distributed stochastic neighbor embedding (t-SNE) [169] and Uniform Manifold Approximation and Projection (UMAP) [87]. Due to its relative speed (as compared to t-SNE) and strong theoretical foundations, UMAP is quickly becoming one of the most popular non-linear general dimensionality reduction algorithms in use. Figure 7.2 shows the result of using UMAP to reduce the dog, cat, truck dataset to 2 and 3 dimensions. In this low-dimensional space the Euclidian distances for each sample from each class’s centroid take on meaningful differences among themselves.

Figure 7.3 shows 2 and 3-dimensional UMAP reductions of the micro-PCB training data. When viewed as a collection of small epsilon balls around each point it is difficult to observe much useful structure. Compare these with Figure 7.4, which shows a 3-dimensional reduction of the micro-PCB training data produced with UMAP after removing the outliers from each class (those points more than one standard deviation away from their class’s centroid) and enclosing the points in a convex hull. Using this visualization method, it is possible to observe interesting structure that was is not apparent in the visualizations in Figure 7.3. In the micro-PCB dataset, there are three classes of micro-PCBs that are of the same model (Arduino Mega 2560) but produced by different manufacturers. The boards’ components and layouts are the same, differing

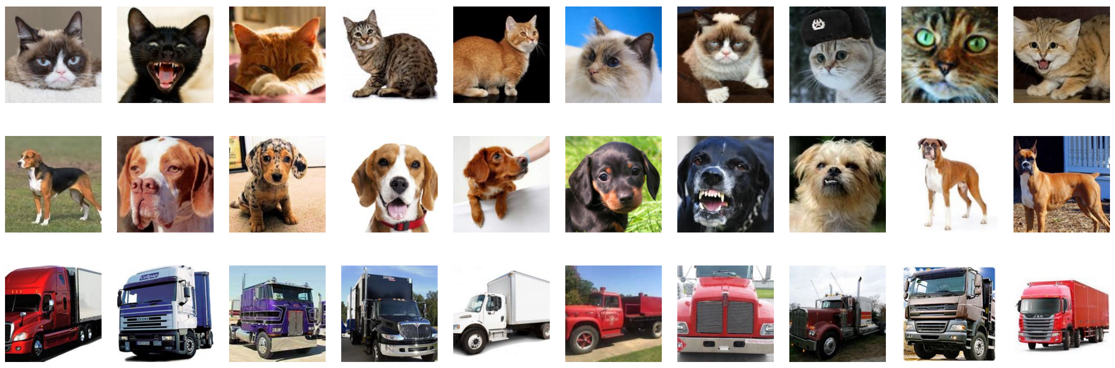


Figure 7.1: A Small 3-Class Dataset Consisting of Cats, Dogs, and Trucks

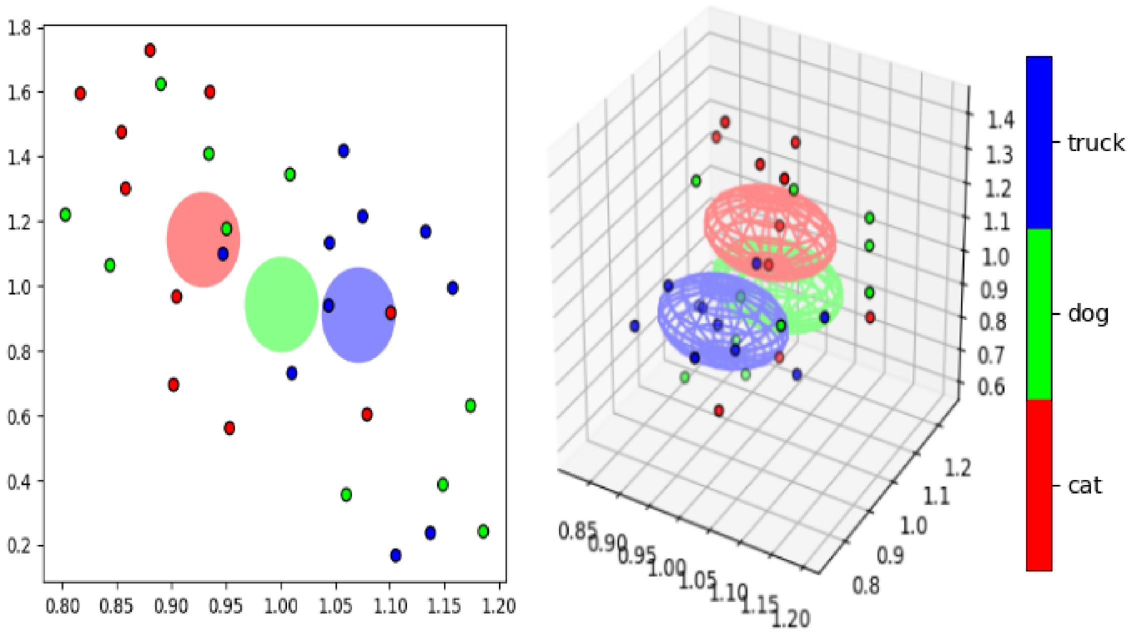


Figure 7.2: Visualization of 2-Dimensional and 3-Dimensional Reductions of the Cat, Dog, Truck Dataset. The circles (in 2D) and spheres (in 3D) represent the classes' centroids.

almost entirely by the colors of the substrates and the colors of plastics used for the pins. The reduction resulted in these classes being tightly clustered together and mostly separate from the remaining classes. A similar tight clustering is exhibited with the two Arduino Unos created by different manufacturers, the two different generations of the Raspberry Pi B+, and the two different Arduino shields.

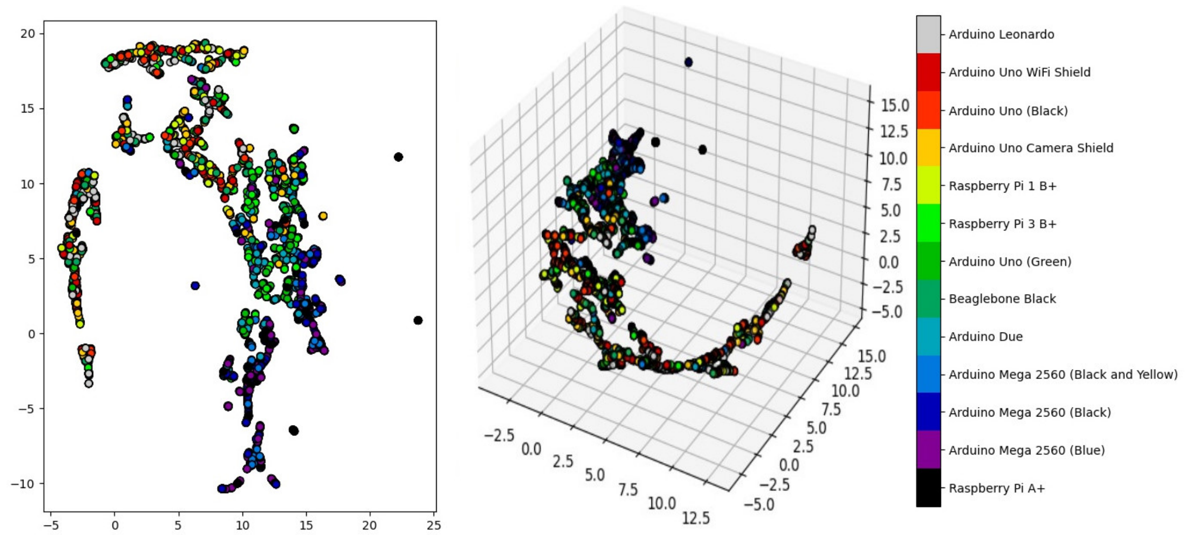


Figure 7.3: Visualization of 2-Dimensional and 3-Dimensional Reductions of the micro-PCB Dataset

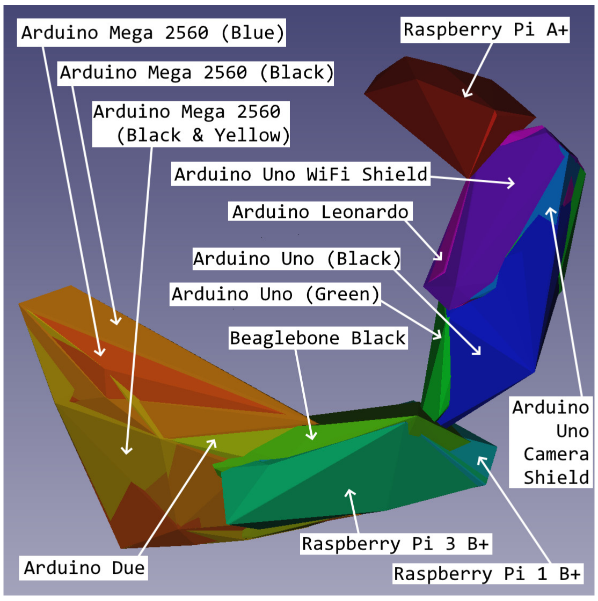


Figure 7.4: Visualization of 3-Dimensional Reduction of the micro-PCB Dataset (Excluding Outliers)

The ability for this 3-dimensional UMAP reduction to effectively separate the classes such that important semantic information of the micro-PCB dataset emerges in the locations of the samples in the reduced space serves as evidence that a distance metric can indeed be leveraged.

The experiments detailed in this chapter use the dimensional reduction of each sample as produced by UMAP to determine what samples to exclude during training. These experiments are performed on standard benchmark datasets in order to make the results comparable to other research. It should be noted that with all experiments detailed in this chapter, all exclusionary methods were applied only to the training set of each dataset and only during training. All testing for all experiments used the entire test set of each dataset.

7.1 Network Architecture and Training

For all experiments, regardless of the dataset used:

1. The network used for training consisted of a single set of stacked 3$\times$3 convolutions, wherein the first convolutional operation produced 32 feature maps.
2. All subsequent convolutional operations in the network produced an additional 16 feature maps.
3. After all convolutional layers in the network a set of Unbroken Z-Derived HVCs were used to produce the final classification.
4. Optimization was performed with the Adam optimizer with an initial learning rate of 0.001 that was exponentially decayed every epoch by 0.98.
5. Training proceeded for 300 epochs.

However, because different datasets are formed from images with different sizes, differing number of color channels, and differing complexity of the features present, some slight differences were required depending on the dataset being trained on.

7.1.1 Training on MNIST, Fashion-MNIST, and EMNIST-Digits

These datasets are composed of single color-channel images comprised of the simplest features relative to the other datasets. They all also use the smallest initial image size of 28$\times$28 pixels. Thus, the network used on these datasets consisted of the fewest convolutional layers (9) and thus the fewest final set of feature maps (160). No padding was used during the convolutional operations, so the final set of feature maps were 10$\times$10. When training this network, a batch size of 120 was used. Data augmentation used during the training for these datasets was uniform and the same strategy as described in chapter 5.

7.1.2 Training on CIFAR-10 and CIFAR-100

These datasets are composed of 3 color-channel images comprised of more complex features than MNIST, Fashion-MNIST, and EMNIST-Digits [170]. Additionally, the images are slightly larger at 32$\times$32 pixels. By using a similar network as with MNIST, Fashion-MNIST, and EMNIST-Digits, but with 2 more convolutional layers and thus 192 feature maps coming out of the final layer, the final set of feature maps were also 10$\times$10. As in the case with MNIST, Fashion-MNIST, and EMNIST-Digits, when training this network, a batch size of 120 was used. Data augmentation used during the training for these datasets was uniform and the same strategy as used for the experiments in chapter 4.

7.1.3 Training on Imagenette

This dataset is composed of 3 color-channel images that are substantially larger than CIFAR-10 and CIFAR-100 with more complex features. The raw images vary in size but were all resized to 299$\times$299. To cope with the larger image size, 15 convolutional layers were used with the first convolutional layer having a stride of 2 and, given the addition of 16 feature maps per convolutional operation, this resulted in the presence of 256 feature maps after the final convolution. Max pooling was applied after the fifth and tenth convolutional operations. Using this configuration, the final set of feature maps were 10$\times$10, consistent with all other datasets experimented on. Due to the larger number of parameters required for this network, a batch size of 32 was used, as was dictated by the constraints of available hardware. Data augmentation used during the training for this dataset was the same strategy as used for the experiments in chapter 4.

7.2 Baseline Results

All subsequent experiments detailed in this chapter involve excluding some subset of the training data during training. In order to understand the impact of those exclusions, a set of baseline experiments was conducted for which all training samples were included for all of the investigated datasets. The results of those experiments are presented in Table 7.1. For these experiments and all subsequent experiments, five trials of each were conducted.

Table 7.1: Baseline Results Wherein No Training Samples Were Excluded

Dataset	Accuracy	Standard Deviation
MNIST	99.716%	0.000162481
Fashion-MNIST	93.404%	0.001380724
CIFAR-10	89.146%	0.001518684
CIFAR-100	61.896%	0.001786169
Imagenette	92.390%	0.002333238

7.3 Data Reduction Strategies

7.3.1 Experimental Design

For the first set of data reduction experiments detailed in this chapter, the high-dimensional image data was reduced using UMAP to 3 dimensions. Then, three data reduction strategies for selecting the data to exclude during training were devised. For the first two data reduction strategies, the 3-dimensional centroid for each class was calculated. The *Lateral Exclusion* data reduction strategy excluded samples furthest from the centroid (compare Figure 7.5 to Figure 7.6). The *Central Exclusion* data reduction strategy excluded samples nearest to the centroid (compare Figure 7.5 to Figure 7.7). The *Random Exclusion* data reduction strategy excluded samples randomly (compare Figure 7.5 to Figure 7.8). Then for each data reduction strategy experiments were conducted excluding 1%, 2%, 5%, 10%, 25%, and 50% of the training data. These experiments were conducted on CIFAR-10, CIFAR-100, MNIST, Fashion-MNIST, and Imagenette.

7.3.2 Experimental Results

Table 7.2 shows the results of the experiments for the MNIST dataset. Using the *Random Exclusion* strategy was inferior in all cases. Using the *Lateral Exclusion* strategy produced a mean accuracy of 0.002% greater than when using the *Central Exclusion* strategy when 2% of the data was excluded, which is not a statistically significant difference. For all other levels of exclusion, using the *Central Exclusion* strategy was statically significantly superior. When excluding 5% of the training data using the *Central Exclusion* strategy, the accuracy achieved was higher than the baseline that included all samples, though not enough trials were conducted to confirm this as statistically significant. When excluding 10%, the accuracy was identical to that of the baseline. When using the *Central Exclusion* strategy, only when excluding 25% or 50% was the accuracy statistically significantly lower than the baseline.

Table 7.3 shows the results of the experiments for the Fashion-MNIST dataset. In all

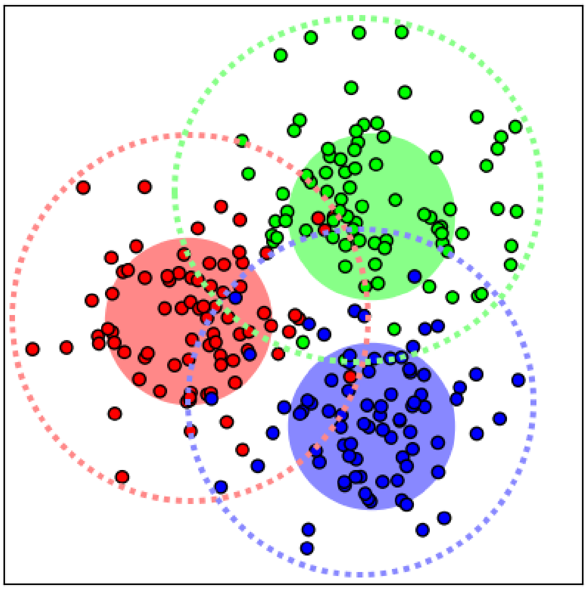


Figure 7.5: Visualization of 2-Dimensional Data Generated for 3 Relatively Well Separated Classes (All Points)

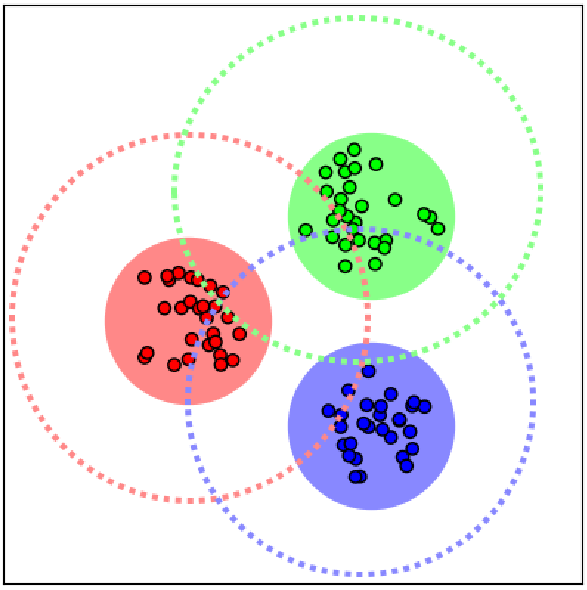


Figure 7.6: Visualization of 2-Dimensional Data Generated for 3 Relatively Well Separated Classes (*Lateral Exclusion* Visualized)

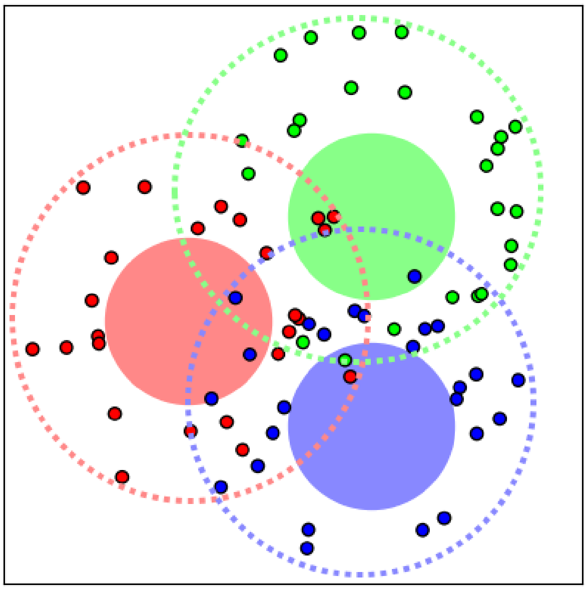


Figure 7.7: Visualization of 2-Dimensional Data Generated for 3 Relatively Well Separated Classes (*Central Exclusion* Visualized)

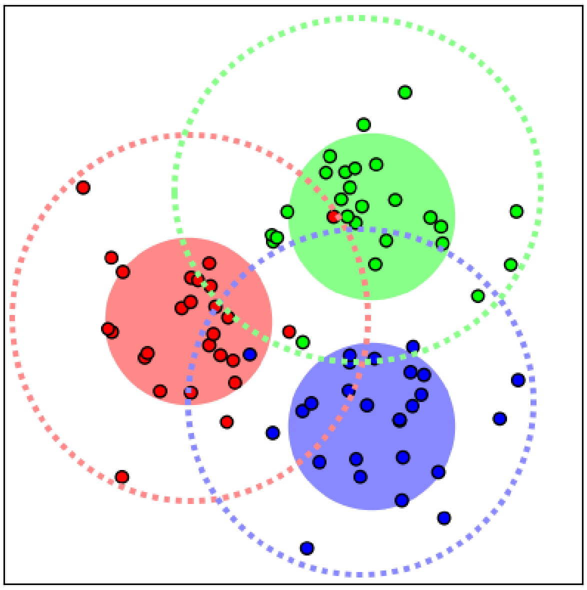


Figure 7.8: Visualization of 2-Dimensional Data Generated for 3 Relatively Well Separated Classes (*Random Exclusion* Visualized)

cases, the *Central Exclusion* strategy proved statistically significantly superior to either of the other two strategies. The *Lateral Exclusion* strategy was superior to the *Random Exclusion* strategy in all but the case of 2% exclusion. However, this superiority was only shown to be statistically significant in the case of the 25% exclusion. There was no statistically significant difference from the baseline when excluding either 1% or 2% of the training data.

The experiments performed on CIFAR-10 (see Table 7.4) demonstrated the greatest amount of ambiguity among all datasets. All three exclusion strategies outperformed the baseline for exclusions of both 1% and 2%, although not enough trials were conducted to show this to be statistically significant. The *Lateral Exclusion* strategy achieved the highest accuracy when excluding 1%, the *Random Exclusion* strategy achieved the highest accuracy when excluding 10%, and the *Central Exclusion* strategy achieved the highest accuracy in all other cases. The statistically significant differences occurred when excluding 5%, 25%, and 50%. When excluding 5% the *Central Exclusion* strategy was statistically significantly superior to both of the other two strategies. When excluding 25% and 50%, the *Central Exclusion* strategy was superior to both of the other two strategies, but only statistically significantly superior to the *Lateral Exclusion* strategy.

Table 7.5 shows the results of the experiments for the CIFAR-100 dataset. In all cases, the accuracy when using the *Central Exclusion* was superior to the other two strategies. The superiority was shown to be statistically significant in the cases of the 10%, 25%, and 50% exclusion experiments. There was no statistically significant difference from the baseline when excluding either 1% or 2% of the training data.

Table 7.6 shows the results of the experiments for the Imagenette dataset. The *Random Exclusion* strategy achieved a statistically insignificant superior accuracy relative to the other two methods for the 1% and 2% exclusion experiments. For the remainder of the experiments, the *Central Exclusion* strategy achieved a higher accuracy than the other two methods, although this was only statistically significant for 3 out of 4 such experiments (5%, 25%, and 50% exclusion). There was no statistically significant difference from the baseline when excluding 1% the training data.

After examining these 5 datasets, it can safely be concluded that, in general, the *Central Exclusion* strategy is the superior strategy among the three. In no experiments did either of the other two strategies show a statistically significant superiority to it. Excluding data resulted in equivalent or superior accuracy over the baseline for experiments using 3 of the 5 datasets, including CIFAR-10 when excluding 1%, 2%, or 5%, CIFAR-100 when excluding 1%, and MNIST when excluding 5% or 10%. However, due to the high variance across trials, this was not able to be demonstrated as statistically significant. Similarly, no experiment for any dataset when excluding 1% or 2% was shown to be statistically significantly inferior to the baseline.

Table 7.2: Data Reduction Strategy Experimental Results — MNIST

	Lateral Exclusion		Central Exclusion		Random Exclusion	
Excl. %	Acc.	Std. Dev.	Acc.	Std. Dev.	Acc.	Std. Dev.
Baseline Accuracy — 99.716%						
1%	99.702%	1.47×10^{-4}	**99.714%**	1.74×10^{-4}	99.694%	4.90×10^{-5}
2%	**99.714%**	2.42×10^{-4}	99.712%	1.17×10^{-4}	99.696%	2.33×10^{-4}
5%	99.676%	1.62×10^{-4}	**99.720%**	1.67×10^{-4}	99.658%	1.83×10^{-4}
10%	99.658%	1.17×10^{-4}	**99.716%**	1.20×10^{-4}	99.638%	2.32×10^{-4}
25%	99.546%	2.58×10^{-4}	**99.698%**	2.14×10^{-4}	99.526%	2.33×10^{-4}
50%	99.190%	3.03×10^{-4}	**99.686%**	2.65×10^{-4}	99.236%	2.58×10^{-4}

Table 7.3: Data Reduction Strategy Experimental Results — Fashion-MNIST

	Lateral Exclusion		Central Exclusion		Random Exclusion	
Excl. %	Acc.	Std. Dev.	Acc.	Std. Dev.	Acc.	Std. Dev.
Baseline Accuracy — 93.404%						
1%	93.114%	7.89×10^{-4}	**93.346%**	8.71×10^{-4}	93.032%	1.33×10^{-3}
2%	92.896%	6.56×10^{-4}	**93.324%**	1.75×10^{-3}	92.940%	1.66×10^{-3}
5%	92.096%	1.33×10^{-3}	**93.198%**	5.49×10^{-4}	92.084%	7.42×10^{-4}
10%	91.088%	1.34×10^{-3}	**93.184%**	6.47×10^{-4}	91.032%	4.71×10^{-4}
25%	89.096%	9.00×10^{-4}	**92.162%**	9.45×10^{-4}	88.932%	5.38×10^{-4}
50%	85.690%	4.24×10^{-4}	**88.692%**	2.31×10^{-3}	85.558%	1.71×10^{-3}

Table 7.4: Data Reduction Strategy Experimental Results — CIFAR-10

	Lateral Exclusion		Central Exclusion		Random Exclusion	
Excl. %	Acc.	Std. Dev.	Acc.	Std. Dev.	Acc.	Std. Dev.
Baseline Accuracy — 89.146%						
1%	**89.332%**	2.83×10^{-3}	89.280%	1.86×10^{-3}	89.318%	1.18×10^{-3}
2%	89.148%	8.93×10^{-4}	**89.206%**	1.21×10^{-3}	89.148%	1.04×10^{-3}
5%	89.054%	1.45×10^{-3}	**89.224%**	1.30×10^{-3}	89.086%	8.09×10^{-4}
10%	88.708%	6.97×10^{-4}	88.620%	1.87×10^{-3}	**88.838%**	1.94×10^{-3}
25%	87.098%	1.44×10^{-3}	**87.930%**	4.00×10^{-4}	87.878%	4.29×10^{-3}
50%	84.044%	1.32×10^{-3}	**85.900%**	1.13×10^{-3}	85.706%	7.40×10^{-3}

Table 7.5: Data Reduction Strategy Experimental Results — CIFAR-100

	Lateral Exclusion		Central Exclusion		Random Exclusion	
Excl. %	Acc.	Std. Dev.	Acc.	Std. Dev.	Acc.	Std. Dev.
Baseline Accuracy — 61.896%						
1%	61.926%	2.37×10^{-3}	**61.940%**	3.72×10^{-3}	61.888%	4.26×10^{-3}
2%	61.766%	2.63×10^{-3}	**61.866%**	1.90×10^{-3}	61.594%	2.44×10^{-3}
5%	61.262%	3.26×10^{-3}	**61.414%**	2.07×10^{-3}	61.102%	3.81×10^{-3}
10%	60.410%	2.72×10^{-3}	**61.044%**	2.53×10^{-3}	60.012%	2.41×10^{-3}
25%	57.440%	2.25×10^{-3}	**59.158%**	2.01×10^{-3}	57.206%	4.07×10^{-3}
50%	50.754%	2.48×10^{-3}	**53.868%**	4.45×10^{-3}	51.334%	3.94×10^{-3}

Table 7.6: Data Reduction Strategy Experimental Results — Imagenette

	Lateral Exclusion		Central Exclusion		Random Exclusion	
Excl. %	Acc.	Std. Dev.	Acc.	Std. Dev.	Acc.	Std. Dev.
Baseline Accuracy — 92.390%						
1%	92.288%	1.46×10^{-3}	92.300%	3.71×10^{-3}	**92.316%**	2.34×10^{-3}
2%	92.022%	2.14×10^{-3}	92.196%	2.28×10^{-3}	**92.342%**	2.19×10^{-3}
5%	91.896%	6.77×10^{-4}	**92.214%**	1.57×10^{-3}	91.994%	8.36×10^{-4}
10%	91.544%	2.91×10^{-3}	**91.844%**	2.91×10^{-3}	91.680%	1.94×10^{-3}
25%	90.422%	2.20×10^{-3}	**90.998%**	2.35×10^{-3}	90.342%	1.78×10^{-3}
50%	87.998%	3.55×10^{-3}	**88.174%**	2.56×10^{-3}	87.914%	5.66×10^{-3}

7.4 The Cardinality of the Dimensionality of the Reduced Space

7.4.1 Experimental Design

To test the hypothesis that 3-dimensions was the appropriate choice for the dimensional reduction, an additional set of experiments using the *Central Exclusion* data reduction method, and excluding 1%, 2%, 5%, 10%, 25%, and 50% of the training data were conducted. This set of experiments used 2, 5, and 10 dimensional reductions and were conducted on CIFAR-10, CIFAR-100, MNIST, Fashion-MNIST, and Imagenette.

7.4.2 Experimental Results

Executing 5 trials of each of 6 different amounts of excluded data results in 30 total trials per dataset and number of dimensions being used to determine the exclusion. For each of the 2, 5, and 10 dimensional reductions the mean accuracy achieved across all 30 trials was compared to the 30 trials of the experiments that used a 3-dimensional reduction. Table 7.7 shows the results of those comparisons. The comparisons showed no statistically significant difference between any paired sets of 30 trials. 5 out of 15 experiments showed a statistically insignificant superiority when using a 3-dimensional reduction, including all 3 comparisons of MNIST and 2 out of 3 comparisons of Imagenette. Although, not reaching a reasonable threshold for statistical significance ($p < 0.05$), the MNIST comparisons had the lowest p-values. At first, this may seem surprising, but upon reflection, it seems reasonable that points UMAP places far from a class's centroid in 3 dimensions would also be likely to be placed far from the class's centroid in 2, 5, or 10 dimensions as well.

7.5 Distributions of the Dimensional Reductions

UMAP generates numeric values for each dimension of the reduction performed. Visualizations of 2-dimensional and 3-dimensional reductions are usually generated by drawing these points as small epsilon balls around the positions of those numeric values. While these visualizations can provide some sense of where the classes' data are located in space, drawing convex hulls that surround each class's points provides a sharper distinction between the boundaries of the classes in the space. Figure 7.9 and Figure 7.10 show the result of drawing these hulls around the 3-dimensional reductions of the full MNIST and Imagenette training data, respectively. These visualizations make it clear that when including all of the training data, there is very little distinction of

Table 7.7: Comparison of Mean Accuracies for Exclusions Based on Differing Dimensional Reductions

Dataset	Dimensions	Accuracy	3D Accuracy	p-value
MNIST	2	99.7000%	**99.7077%**	0.111615712
MNIST	5	99.7027%	**99.7077%**	0.232822313
MNIST	10	99.6977%	**99.7077%**	0.083150845
Fashion-MNIST	2	**92.3803%**	92.3177%	0.44224238
Fashion-MNIST	5	**92.4217%**	92.3177%	0.405938742
Fashion-MNIST	10	**92.3490%**	92.3177%	0.471922235
CIFAR-10	2	**88.3677%**	88.3600%	0.490392657
CIFAR-10	5	**88.3710%**	88.3600%	0.486238005
CIFAR-10	10	**88.3747%**	88.3600%	0.481346744
CIFAR-100	2	**59.9347%**	59.8817%	0.471617385
CIFAR-100	5	**60.0760%**	59.8817%	0.396706608
CIFAR-100	10	**59.9790%**	59.8817%	0.448318135
Imagenette	2	**91.3237%**	91.2877%	0.463151304
Imagenette	5	91.2153%	**91.2877%**	0.427306126
Imagenette	10	91.2417%	**91.2877%**	0.453380991

boundaries in the space.

However, when excluding some of the data furthest from each class's centroid, the classes' hulls start to separate, in some cases, partially and in some cases entirely. The amount of data that must be elided to achieve this is referred to in this dissertation as *Dataset Severability*.

Definition 7.1 (Dataset Severability)**.** *Dataset Severability* is a qualitative judgment regarding the number of outliers that must be removed from each class in an m-dimensional reduction of the dataset so that structure and/or clustering is able to be observed.

Figure 7.11 and Figure 7.12 show well severed classes for visualizations of the MNIST and EMNIST-Digits training data, respectively. In each case, the data within three standard deviations of all dimensions was included, and data outside of these bounds was omitted. This means that high severability is achieved by excluding $\approx$ 0.27% of the outliers for these datasets. Despite consisting of images with the same size and number of color channels (monochromatic) as MNIST and EMNIST-Digits, Fashion-MNIST shows a dissimilar level of severability until all but one standard deviation from the centroid has been excluded (excluding $\approx$ 31.73%).

Especially interesting in the case of the Fashion-MNIST reduction is that there is a readily identifiable semantic difference between each of the four completely severed sets of overlapping hulls. The set of "overlapping" hulls consisting of the single class “trouser” contains the only class in the dataset that is legwear. The set of “overlapping” hulls consisting of the single class “bag” contains the only class in the dataset that is not a type of clothing that covers any part of the body. The set of overlapping hulls consisting of the classes “sneaker”, “sandal”, and “ankle boot” contains the only classes in the dataset that are footwear. Finally, the set of overlapping hulls consisting of the classes “pullover”, “coat”, “shirt”, “dress”, and “t-shirt” contains the only classes in the dataset that primarily cover the torso.

Imagenette (Figure 7.14), CIFAR-10 (Figure 7.15), and CIFAR-100 (Figure 7.16) require all but those samples within half a standard deviation of each class’s centroid to be elided before structure emerges (excluding $\approx 61.71\%$). Of these three, the reduction of CIFAR-10 displays the most interesting semantic relationships among the classes’ hull’s locations in space. The classes on the left of the visualization in Figure 7.15 are all man-made (specifically vehicles) whereas the classes on the right are all lifeforms. Especially interesting among the lifeforms is that all of the mammals are grouped together with the one amphibian (frog) on the outer edge of the group.

In Figure 7.17 through Figure 7.36 the values of each of the 3 dimensions of the reductions for each individual class of the training data for MNIST and Imagenette are plotted. The first thing that can be learned from looking at the MNIST plots is that the reduction for each class produces values significantly different than the others and further, the values in each dimension of each class are tightly grouped in the number line, with the noticeable exception of the third dimension of the class that represents the digit 1. It is worth noting that the stylization of the Hindu-Arabic numeral ’1’ contains most of its information in 2-dimensions (accounting for translation) thus providing an explanation for the larger variance in the third dimension. The plots of the Imagenette classes, on the other hand, are much more similar to one another and display greater variance in each of the 3 dimensions.

7.6 Summary

The experiments performed in this chapter show that, for the datasets examined, certain training samples are more informative of class membership than others. These samples can be identified *a priori* to training by analyzing their position in reduced dimensional space relative to the classes’ centroids. Specifically, it was demonstrated that samples nearer the classes’ centroids are less informative than those that are furthest from it. For the five datasets investigated, it was shown that there was no statistically significant difference from the baseline when excluding up to 2% of the data nearest to each class’s centroid. For CIFAR-100, superior accuracy was achieved when excluding 1% of the data nearest to each class’s centroid. For CIFAR-10, superior accuracy was achieved when excluding 5% of the data nearest to each class's centroid. And for the MNIST dataset, identical accuracy to the baseline was achieved when excluding 10% of the data nearest to each class's centroid.

Additionally, *Dataset Severability* was defined in this chapter to be a qualitative, yet quantifiable, judgement regarding the separation of classes in a reduced dimensional space. High severability was demonstrated for MNIST and Fashion-MNIST, whereas low severability was demonstrated for CIFAR-10, CIFAR-100, and Imagenette. Those datasets that demonstrated high severability all achieved higher accuracies in the experiments in this chapter compared to the accuracies of those experiments for the datasets that demonstrated low severability.

One limitation of the approach used in this study is that it was limited to analyzing benchmark datasets of natural images. These benchmarks are thus subject to the potential for sampling bias and latent properties of the labeling process. The next step in this line of research will be to generate synthetic datasets with well understood distributions and then apply the methods discussed to those.

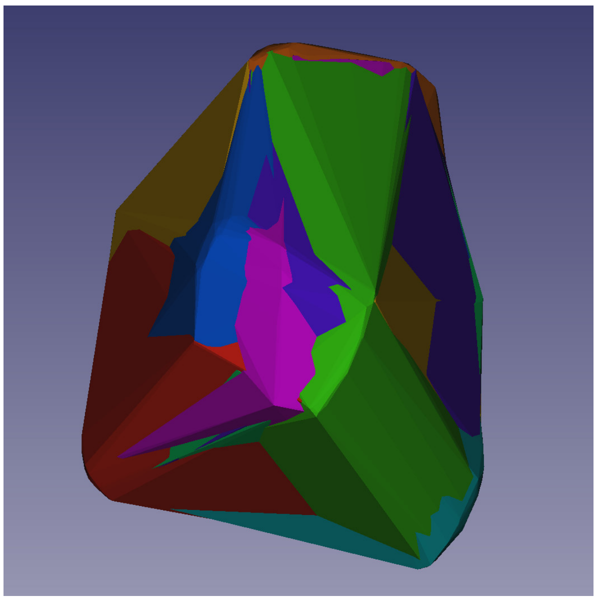


Figure 7.9: Visualization of 3-Dimensional Reduction of MNIST

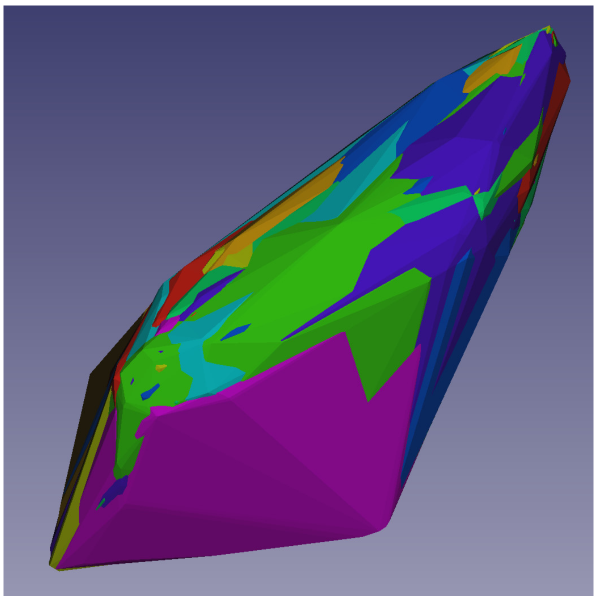


Figure 7.10: Visualization of 3-Dimensional Reduction of Imagenette

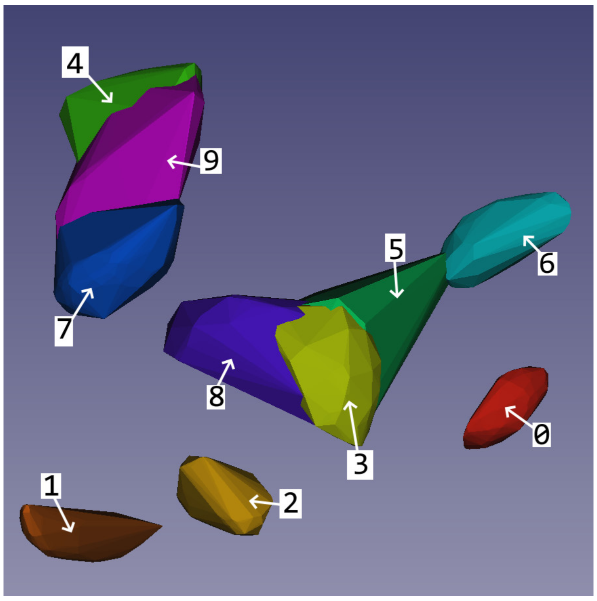


Figure 7.11: Visualization of 3-Dimensional Reduction of MNIST w/ Classes Labeled (Excluding Outliers)

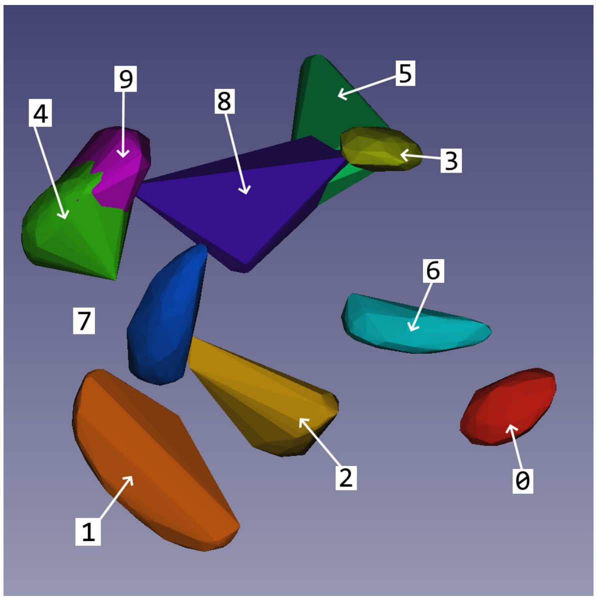


Figure 7.12: Visualization of 3-Dimensional Reduction of EMNIST-Digits w/ Classes Labeled (Excluding Outliers)

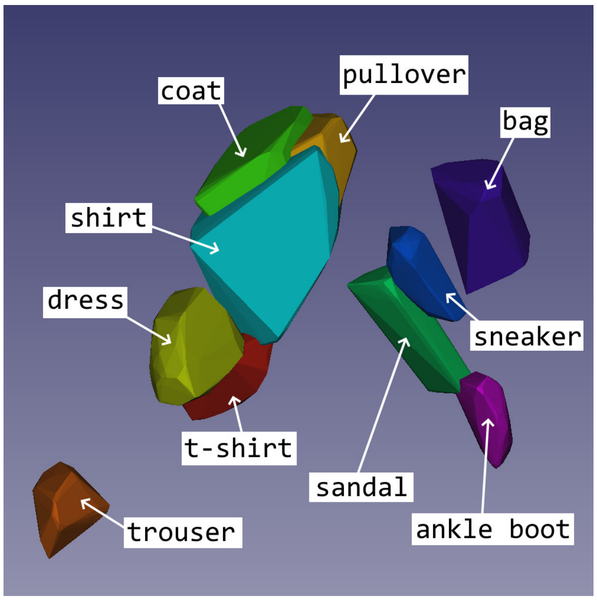


Figure 7.13: Visualization of 3-Dimensional Reduction of Fashion-MNIST w/ Classes Labeled (Excluding Outliers)

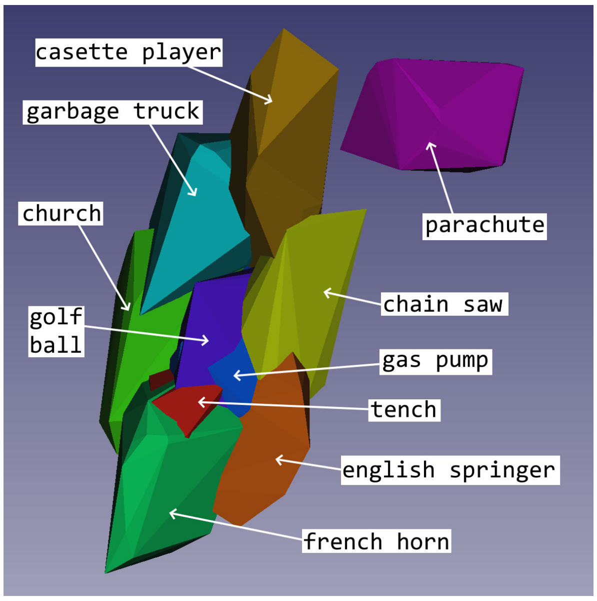


Figure 7.14: Visualization of 3-Dimensional Reduction of Imagenette w/ Classes Labeled (Excluding Outliers)

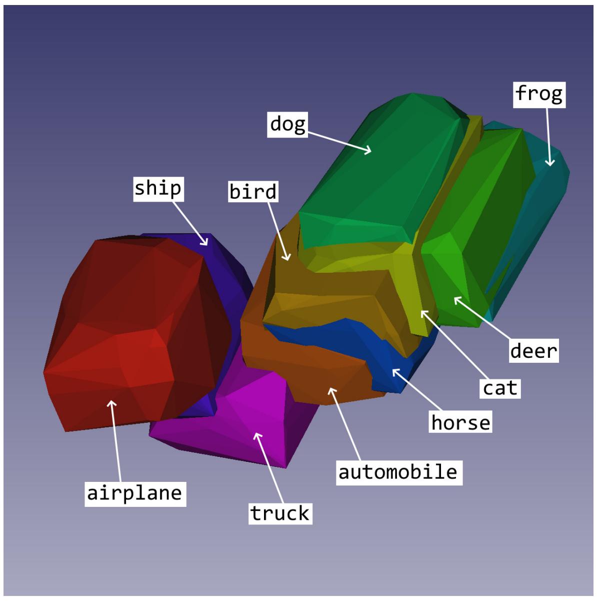


Figure 7.15: Visualization of 3-Dimensional Reduction of CIFAR-10 w/ Classes Labeled (Excluding Outliers)

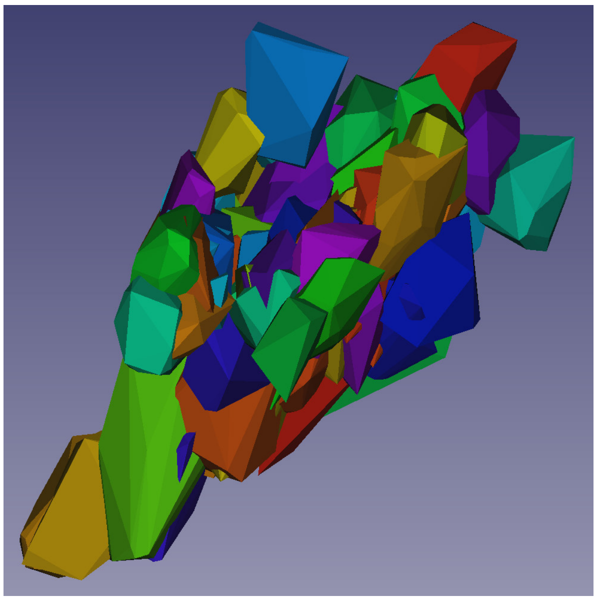


Figure 7.16: Visualization of 3-Dimensional Reduction of CIFAR-100 w/ Classes Labeled (Excluding Outliers)

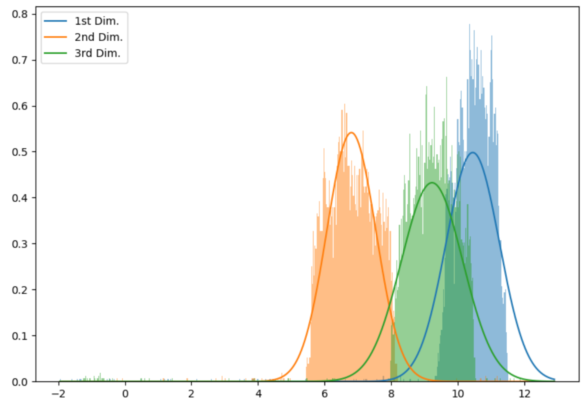


Figure 7.17: MNIST Reduction Distributions — Class ‘0’

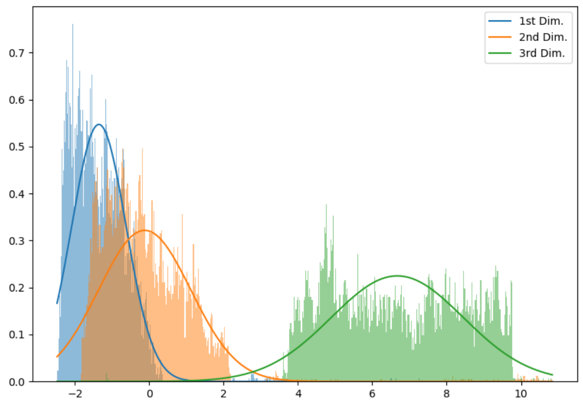


Figure 7.18: MNIST Reduction Distributions — Class ‘1’

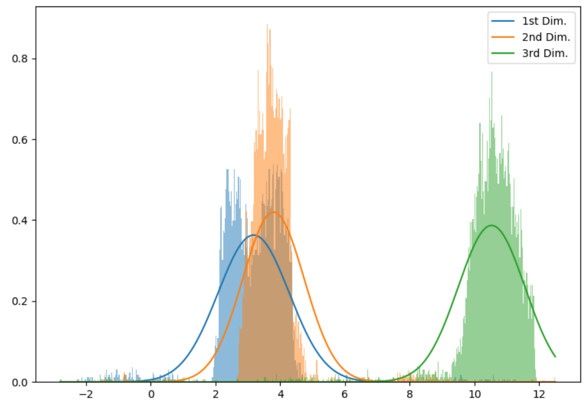


Figure 7.19: MNIST Reduction Distributions — Class ‘2’

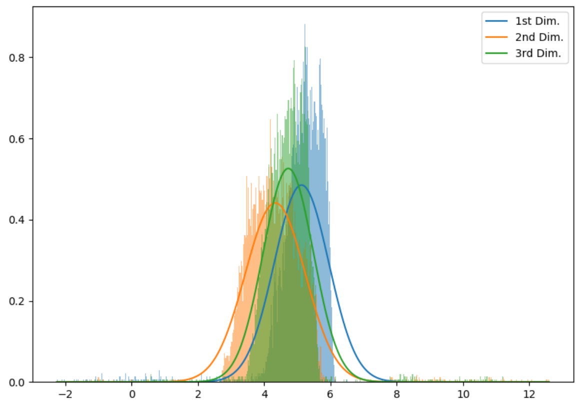


Figure 7.20: MNIST Reduction Distributions — Class ‘3’

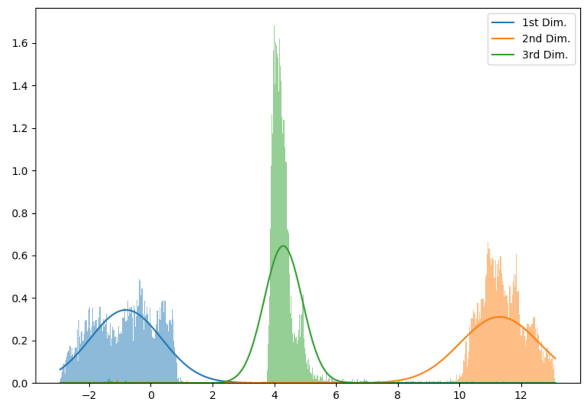


Figure 7.21: MNIST Reduction Distributions — Class ‘4’

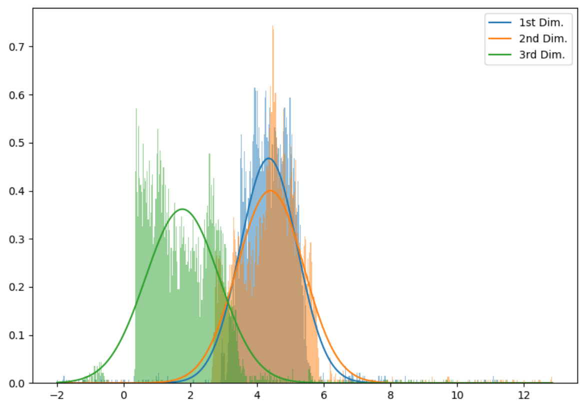


Figure 7.22: MNIST Reduction Distributions — Class ‘5’

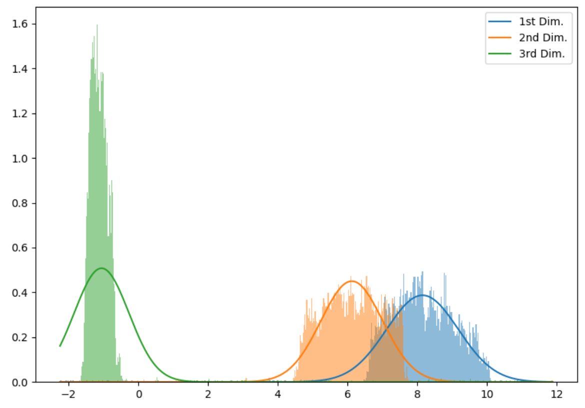


Figure 7.23: MNIST Reduction Distributions — Class ‘6’

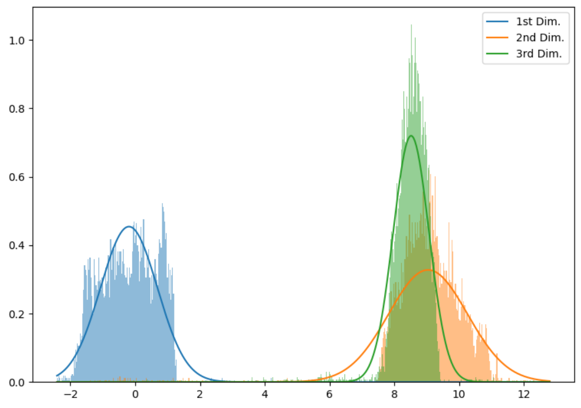


Figure 7.24: MNIST Reduction Distributions — Class ‘7’

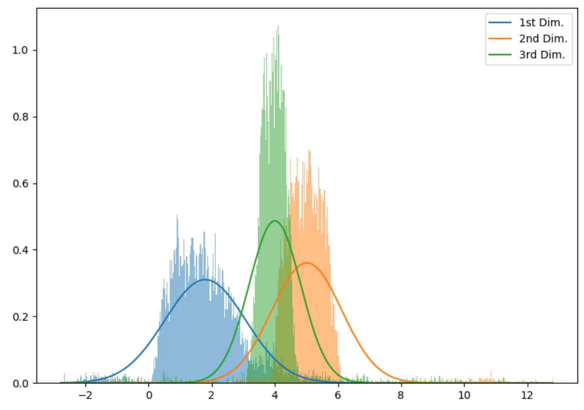


Figure 7.25: MNIST Reduction Distributions — Class ‘8’

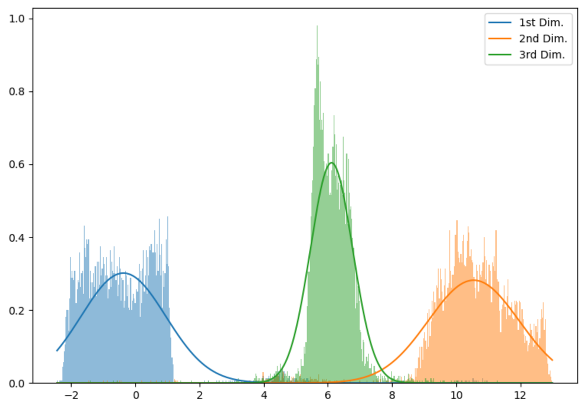


Figure 7.26: MNIST Reduction Distributions — Class ‘9’

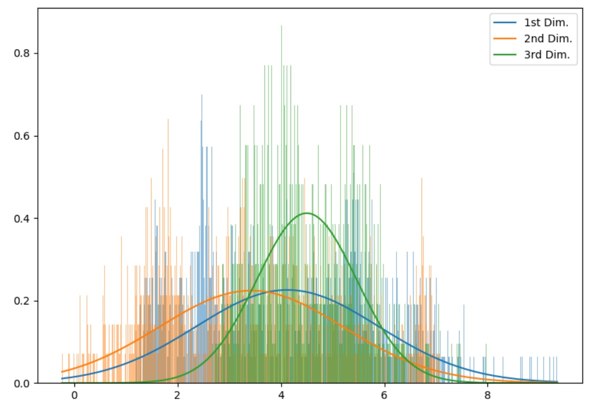


Figure 7.27: Imagenette Reduction Distributions — Class ‘Tench’

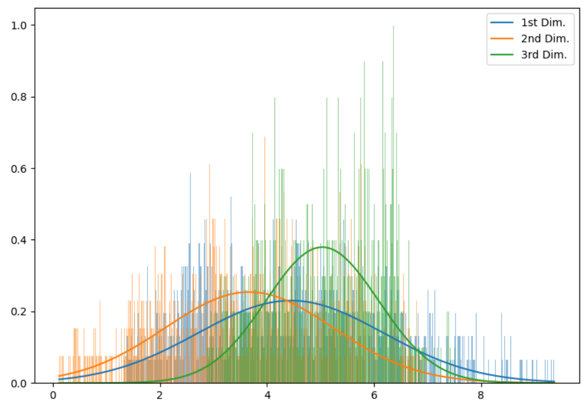


Figure 7.28: Imagenette Reduction Distributions — Class ‘English Springer’

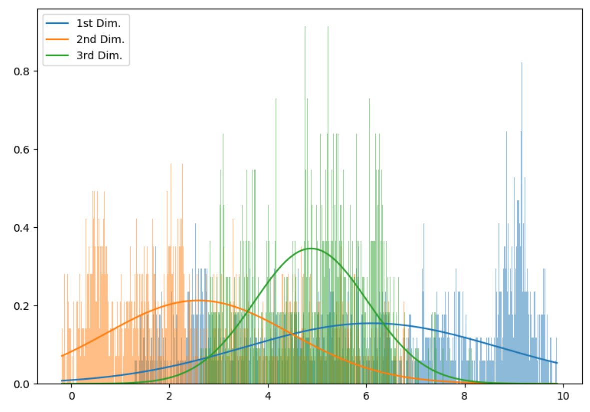


Figure 7.29: Imagenette Reduction Distributions — Class ‘Cassette Player’

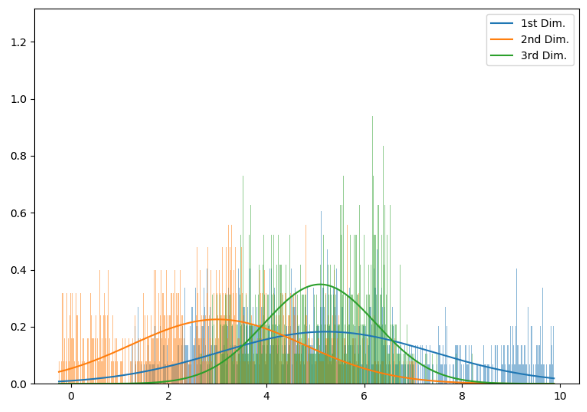


Figure 7.30: Imagenette Reduction Distributions — Class ‘Chain Saw’

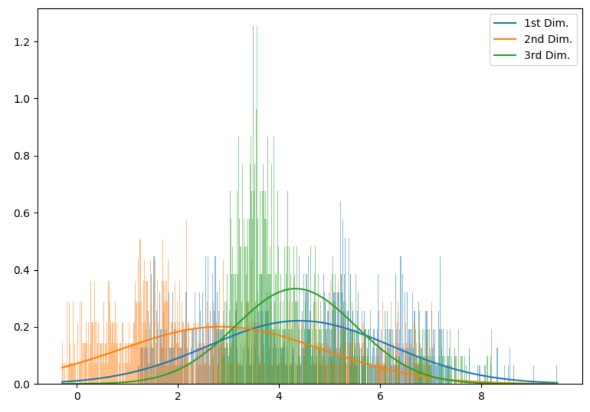


Figure 7.31: Imagenette Reduction Distributions — Class ‘Church’

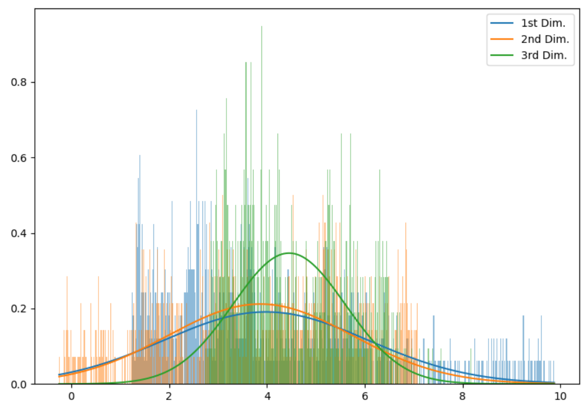


Figure 7.32: Imagenette Reduction Distributions — Class ‘French Horn’

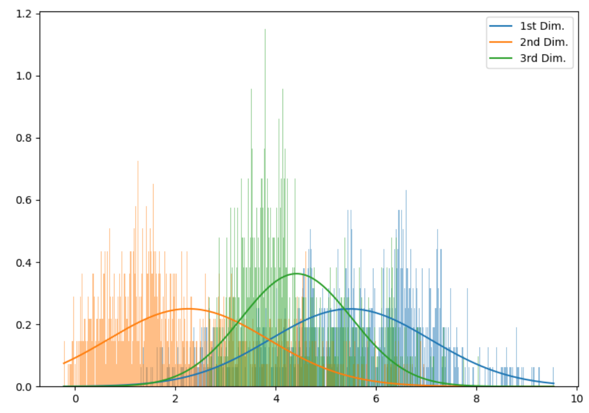


Figure 7.33: Imagenette Reduction Distributions — Class ‘Garbage Truck’

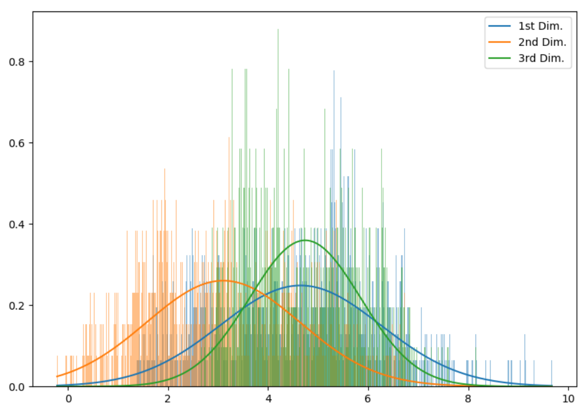


Figure 7.34: Imagenette Reduction Distributions — Class ‘Gas Pump’

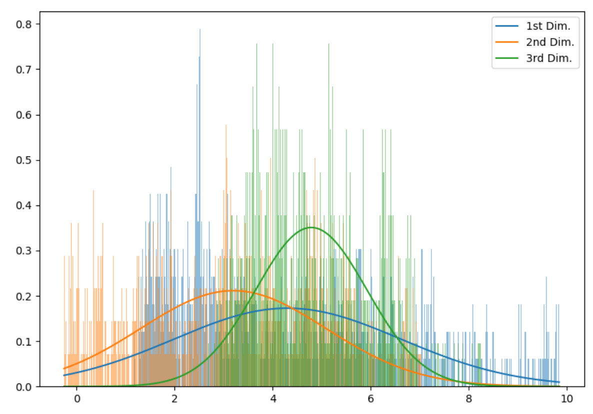


Figure 7.35: Imagenette Reduction Distributions — Class ‘Golf Ball’

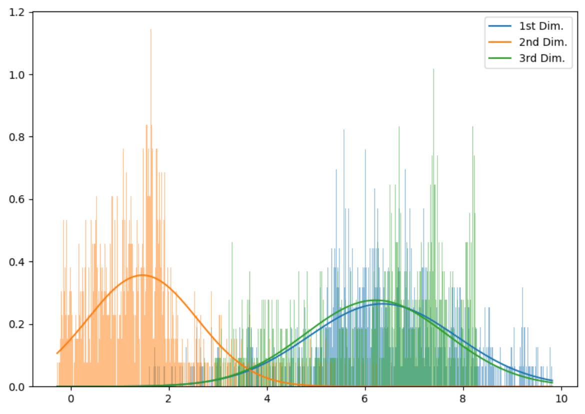


Figure 7.36: Imagenette Reduction Distributions — Class ‘Parachute’

Chapter 8

Class Density and Dataset Completeness

The fact that the previous chapter showed the *Central Exclusion* data reduction strategy to be the superior strategy suggests that too much data that is too similar within a single class impedes, or at least does not help, the model's ability to classify accurately. This implies a set of experiments that, rather than reducing the same percentage of each class of the training data, tests reducing potentially differing amounts of data in each class relative to some measure of that class's density.

Definition 8.1 (Class Density)**.** The *class density* of a class is a measure of the aggregate concentration of the data points in the class.

The hypothesis being tested in this chapter is that there exists a positive correlation between the accuracy of the predictions for that class and the density of data within that class. Once confirmed, a subsequent hypothesis will be tested. That hypothesis being that using the *Central Exclusion* data reduction strategy to reduce the density of the more dense classes to achieve parity among the classes will produce equivalent or superior overall accuracy.

Three candidates for the calculation of a class's density were initially considered, all of them based on the distribution of points in each dimension of any m-dimensional reduction. The three candidates are based on the minimum (see Equation 8.1), the maximum (see Equation 8.2), and the mean (see Equation 8.3) standard deviation of the m gaussians of the m-dimensional reduction.

For all three candidate calculations, the density for class i is calculated using a statistic (min, max, or mean) of the σ_i standard deviations of the m gaussians of the m-dimensional reduction for class i.

The densities were calculated using each of these calculations for each of the classes for MNIST, Fashion-MNIST, CIFAR-10, CIFAR-100, and Imagenette. Then the individual

The work presented in this chapter has been submitted for publication.

$$d_i^{\text{min}} = \text{min}\left(\sigma_i\right)^{-1} \tag{8.1}$$

Min-Derived Density Calculation

$$d_i^{\text{max}} = \text{max}\left(\sigma_i\right)^{-1} \tag{8.2}$$

Max-Derived Density Calculation

$$d_i = \left(\frac{1}{m}\sum_{k}^{m}\sigma_{i_k}\right)^{-1} \tag{8.3}$$

Mean-Derived Density Calculation

class accuracies for every class in each of these datasets were averaged across the five trials of the baseline experiments. Then correlations were calculated, using the Pearson Product-Moment Correlation Coefficient (PPMCC), between the class densities and the class accuracies looking for a correlation between higher accuracy and higher density. The results of this correlation study are presented in Table 8.1, Table 8.2, Table 8.3, Table 8.4, and Table 8.5 (a tabulation of the correlation of the individual classes in CIFAR-100 has not been tabulated here due to the large number of classes). The study showed that on average, all candidate calculations showed a moderate correlation. However, the min and max candidates each had a dataset for which the correlation was negative (MNIST and Fashion-MNIST respectively). Aside from having no datasets with a negative correlation, the mean candidate also had the highest mean correlation. Notably, CIFAR-10, and CIFAR-100 had the weakest non-negative correlation for all three candidates. When excluding these datasets, the difference in the mean correlation between the min and max candidates and the mean candidate was even greater. As such, the mean candidate, Equation 8.3, has been chosen for the calculation of class density.

Table 8.1: Mean Correlation Between Datasets' Class Accuracies and Class Densities for Each Candidate Class Density Calculation

Density Calculation	Dataset	Correlation	Correlation Excluding CIFAR-10/CIFAR-100
Min-Derived Density (Equation 8.1)	MNIST	-0.12523	-0.12523
	Fashion-MNIST	0.71220	0.71220
	CIFAR-10	0.29943	—
	CIFAR-100	0.12966	—
	Imagenette	0.40819	0.40819
	Mean	0.28485	0.33172
Max-Derived Density (Equation 8.2)	MNIST	0.62710	0.62710
	Fashion-MNIST	-0.10550	-0.10550
	CIFAR-10	0.27080	—
	CIFAR-100	0.15321	—
	Imagenette	0.33469	0.33469
	Mean	0.25606	0.28543
Mean-Derived Density (Equation 8.3)	MNIST	0.59902	0.59902
	Fashion-MNIST	0.46572	0.46571
	CIFAR-10	0.14169	—
	CIFAR-100	0.13030	—
	Imagenette	0.37388	0.37387
	Mean	**0.34212**	**0.47954**

Class #	Accuracy	Density (d_i)
0	99.94%	1.20361
1	99.75%	0.90039
2	99.79%	0.96696
3	99.96%	1.23406
4	99.57%	0.95438
5	99.44%	0.91778
6	99.71%	1.09510
7	99.65%	1.18253
8	99.81%	0.91090
9	99.50%	0.87517
Correlation Coefficient		0.59902

Table 8.2: Class Accuracy vs. Class Density using the *Mean-Derived* Density Calculation (Equation 8.3) — MNIST

Class #	Accuracy	Density (d_i)
0	89.44%	0.78580
1	98.97%	0.78724
2	90.45%	0.88673
3	88.85%	0.94408
4	93.40%	0.80576
5	98.73%	0.82540
6	80.40%	0.55017
7	98.13%	1.21943
8	99.26%	0.72068
9	96.46%	1.09453
Correlation Coefficient		0.46572

Table 8.3: Class Accuracy vs. Class Density using the *Mean-Derived* Density Calculation (Equation 8.3) — Fashion-MNIST

Class #	Accuracy	Density (d_i)
0	93.68%	0.67526
1	96.49%	0.69478
2	90.34%	0.56108
3	86.04%	0.53370
4	96.20%	0.59826
5	91.48%	0.59085
6	93.88%	0.70907
7	86.70%	0.69479
8	94.08%	0.56492
9	94.95%	0.74751
Correlation Coefficient		0.37387

Table 8.4: Class Accuracy vs. Class Density using the *Mean-Derived* Density Calculation (Equation 8.3) — Imagenette

Class #	Accuracy	Density (d_i)
0	90.97%	0.72625
1	95.01%	0.65924
2	85.27%	0.69837
3	75.59%	0.71100
4	88.89%	0.75716
5	81.51%	0.78188
6	92.85%	0.76589
7	92.78%	0.74160
8	94.28%	0.78661
9	93.99%	0.79986
Correlation Coefficient		0.14169

Table 8.5: Class Accuracy vs. Class Density using the *Mean-Derived* Density Calculation (Equation 8.3) — CIFAR-10

8.1 Dynamic Data Reduction

8.1.1 Experimental Design

Using the *Mean-Derived* Density Calculation (Equation 8.3), another set of experiments was conducted. In these experiments, the training data in each class was reduced using the *Central Exclusion* data reduction strategy by the number of samples necessary to ensure all classes in the training data that had a density greater than a target density value were reduced by the number of samples needed to achieve that target density value. In order to find how many samples would need to be excluded, a binary search through each class's samples, ordered by distance from the class's centroid was performed. The binary search terminated and selected the threshold distance for inclusion after 9 iterations, which was sufficient to choose the number of samples to be included to within a margin of .05% of the total number of samples in the class.

While studying dynamic data reduction, two additional sets of experiments were conducted. The first set was conducted on the micro-PCB dataset introduced in chapter 6. Data augmentation used during the training for the micro-PCB dataset was uniform and the same strategy as used for the experiments in chapter 4. The second set was conducted on EMNIST-Digits, which is a dataset with the same format as MNIST and with the same domain of interest, namely the Hindu-Arabic numerals. The difference is that EMNIST-Digits contains exactly four times as many samples in both the training and test sets. This allows for an investigation into whether the test accuracy achieved after reducing data to achieve a target threshold per class is sensitive to or independent of the number of samples used for training. Data augmentation used during the training for EMNIST-Digits was uniform and the same strategy as used for the experiments in chapter 5.

Each of MNIST, Fashion-MNIST, CIFAR-10, CIFAR-100, and EMNIST-Digits were subjected to this method using target density levels of 1.0, 0.9, 0.8, 0.7, 0.6, and 0.5. These target densities were chosen because they produced a similar number of excluded samples for these datasets. However, in the case of Imagenette and the micro-PCB dataset, the UMAP dimensional reduction produced values in each dimension with much larger variances (thus producing smaller densities). This is due to the fact that the UMAP algorithm assumes that the population in the higher dimensional space from which the data is sampled is uniformly distributed on some manifold. Since Imagenette and the micro-PCB dataset are both higher dimensional than the other datasets as well as having fewer samples in the training set, UMAP assumes the values of these samples in the higher dimensional space represent a much broader range of possible values for the assumed uniformity of the population. Therefore, a larger number of approximately evenly spaced target densities was chosen specifically for each of Imagenette and of the micro-PCB dataset.

8.1.2 Experimental Results

Table 8.6 through Table 8.12 show the results of these experiments. Bold typeface in the Accuracy columns indicate that the accuracy matched or exceeded the accuracy of the baseline.

For the MNIST dataset, there was no statistically significant difference ($p < 0.05$) between the accuracy of the baseline and the accuracies of the experiments with target densities 1.0, 0.9, 0.8, and 0.7. These target densities resulted in reducing the training dataset size by 4.4%, 7.8%, 15%, and 22.6%, respectively.

For the Fashion-MNIST dataset, there was no statistically significant difference ($p < 0.05$) between the accuracy of the baseline and the accuracy of the experiment with target density 1.0 which resulted in reducing the training dataset size by 2.0%.

For the CIFAR-10 dataset, there was no statistically significant difference ($p < 0.05$) between the accuracy of the baseline and the accuracies of the experiments with target densities 1.0, 0.9, 0.8, and 0.7. These target densities resulted in reducing the training dataset size by 0.2%, 0.2%, 0.2%, and 4.2%, respectively.

For the CIFAR-100 dataset, there was no statistically significant difference ($p < 0.05$) between the accuracy of the baseline and the accuracies of the experiments with target densities 1.0 and 0.8. These target densities resulted in reducing the training dataset size by 0.4% and 2.3%, respectively. The experiment with a target density of 0.9 produced a statistically significantly superior accuracy while reducing the dataset size by 0.4%.

For the Imagenette dataset, there was no statistically significant difference ($p < 0.05$) between the accuracy of the baseline and the accuracies of the experiments with target densities 0.70, 0.65, 0.60, and 0.55. These target densities resulted in reducing the training dataset size by 0.3%, 2.4%, 5.1%, and 9.2%, respectively.

For the micro-PCB dataset, there was no statically significant difference ($p < 0.05$) between the accuracy of the baseline and the accuracies of the experiments with target densities 0.25, 0.20, 0.15, 0.10, 0.075, and 0.05. These target densities resulted in reducing the training dataset size by 5.4%, 15.8%, 30.5%, 42.9%, and 53.2%, respectively. Of particular note with the micro-PCB dataset is that the baseline accuracy of 100% was able to be achieved with 53.2% of the training samples removed.

For the EMNIST-Digits dataset, there was no statistically significant difference ($p < 0.05$) between the accuracy of the baseline and the accuracies of the experiments with target densities 1.0, 0.9, and 0.8. These target densities resulted in reducing the training dataset size by 3.1%, 7.2%, and 13.4%, respectively.

Table 8.6: Class Accuracies Achieved at Target Densities — MNIST

Target Density (d_i)	# Samples Included	Accuracy	Std. Dev.	p-value
N/A	100.0%	99.716%	0.000162481	—
1.0	95.6%	99.714%	0.000080000	0.41537
0.9	92.2%	**99.732%**	0.000172047	0.10664
0.8	85.0%	99.708%	0.000097980	0.21179
0.7	77.4%	**99.716%**	0.000135647	0.50000
0.6	69.4%	99.682%	0.000116619	0.00468
0.5	61.0%	99.694%	0.000080000	0.02062

Table 8.7: Class Accuracies Achieved at Target Densities — Fashion-MNIST

Target Density (d_i)	# Samples Included	Accuracy	Std. Dev.	p-value
N/A	100.0%	93.404%	0.001380724	—
1.0	98.0%	93.298%	0.000928224	0.11917
0.9	96.4%	93.202%	0.000982649	0.02214
0.8	93.2%	93.254%	0.000611882	0.04111
0.7	86.4%	92.994%	0.001330564	0.00135
0.6	78.2%	92.574%	0.001089220	6.52×10^{-6}
0.5	68.8%	91.922%	0.001750885	4.90×10^{-7}

Table 8.8: Class Accuracies Achieved at Target Densities — CIFAR-10

Target Density (d_i)	# Samples Included	Accuracy	Std. Dev.	p-value
N/A	100.0%	89.146%	0.001518684	—
1.0	99.8%	**89.342%**	0.002688048	0.11994
0.9	99.8%	**89.154%**	0.002361864	0.47798
0.8	99.8%	**89.346%**	0.002151836	0.08366
0.7	95.8%	**89.192%**	0.001828004	0.35438
0.6	86.2%	88.700%	0.002044505	0.00452
0.5	75.4%	88.064%	0.002030369	1.37×10^{-5}

Table 8.9: Class Accuracies Achieved at Target Densities — CIFAR-100

Target Density (d_i)	# Samples Included	Accuracy	Std. Dev.	p-value
N/A	100.0%	61.896%	0.001786169	—
1.0	99.6%	**62.186%**	0.003273286	0.07923
0.9	99.6%	**62.220%**	0.002830548	0.04444
0.8	97.7%	61.876%	0.003501200	0.46072
0.7	93.1%	61.642%	0.002057571	0.04963
0.6	85.4%	60.398%	0.003592993	3.58×10^{-5}
0.5	75.1%	59.108%	0.001561282	5.71×10^{-9}

Table 8.10: Class Accuracies Achieved at Target Densities — Imagenette

Target Density (d_i)	# Samples Included	Accuracy	Std. Dev.	p-value
N/A	100.0%	92.390%	0.002333238	—
0.70	99.7%	92.104%	0.002514438	0.0670
0.65	97.6%	**92.486%**	0.001497465	0.2541
0.60	94.9%	92.250%	0.003331066	0.2553
0.55	90.8%	92.120%	0.002728369	0.0855
0.50	85.1%	91.798%	0.001984339	0.0024
0.45	78.6%	91.434%	0.001416474	5.60×10^{-5}
0.40	72.2%	90.650%	0.002086145	1.91×10^{-6}
0.35	65.1%	90.026%	0.003338622	1.38×10^{-6}
0.30	57.6%	89.626%	0.001276871	1.51×10^{-8}
0.25	49.8%	88.206%	0.002298347	2.95×10^{-9}
0.20	41.7%	86.304%	0.005313229	1.40×10^{-8}
0.15	32.5%	83.866%	0.00475546	4.73×10^{-10}
0.10	22.4%	79.134%	0.004674441	1.26×10^{-11}
0.05	9.15%	43.564%	0.037764248	2.72×10^{-9}

Table 8.11: Class Accuracies Achieved at Target Densities — micro-PCB

Target Density (d_i)	# Samples Included	Accuracy	Std. Dev.	p-value
N/A	100.0%	100.000%	0	—
0.25	94.6%	**100.000%**	0	N/A
0.20	84.2%	**100.000%**	0	N/A
0.15	69.5%	**100.000%**	0	N/A
0.10	57.1%	**100.000%**	0	N/A
0.075	**46.8%**	**100.000%**	0	N/A
0.05	34.5%	99.912%	1.48×10^{-3}	0.13399
0.04	25.6%	99.024%	9.49×10^{-3}	0.03687
0.03	17.2%	75.112%	5.52×10^{-2}	9.14×10^{-6}
0.02	11.8%	48.088%	2.24×10^{-1}	0.00085
0.01	6.90%	18.800%	5.45×10^{-2}	8.78×10^{-10}

Table 8.12: Class Accuracies Achieved at Target Densities — EMNIST-Digits

Target Density (d_i)	# Samples Included	Accuracy	Std. Dev.	p-value
N/A	100.0%	99.790%	0	—
1.0	96.9%	**99.790%**	6.32456E-05	0.50000
0.9	92.8%	99.786%	4.89898E-05	0.07056
0.8	86.6%	99.782%	9.79796E-05	0.07056
0.7	79.8%	99.784%	4.89898E-05	0.01998
0.6	71.5%	99.772%	4.00000E-05	9.27×10^{-6}
0.5	62.8%	99.766%	1.01980E-04	0.00076

8.2 Dynamic Data Reduction Redux

8.2.1 Experimental Design

As was noted in the previous section, the UMAP dimensionality reduction algorithm makes assumptions about the distribution of the data in the higher dimensional space that results in density values that are dependent on the dataset. In order to account for this, the *Mean-Derived* Density Calculation (Equation 8.3) has been modified to include a normalization term.

In Equation 8.4, $\overline{d_i}$ is the density of class i among all n classes. σ_i and σ_j are the standard deviations of the m gaussians of the m-dimensional reduction for classes i and j, respectively.

$$\overline{d_i} = \frac{1}{n}\sum_{j}^{n}\left(\frac{1}{m}\sum_{k}^{m}\sigma_{j_k}\right)\cdot\left(\frac{1}{m}\sum_{k}^{m}\sigma_{i_k}\right)^{-1} \tag{8.4}$$

Mean-Derived and Normalized Density Calculation

Table 8.13 through Table 8.18 show a comparison of the densities calculated using the *Mean-Derived* Density Calculation (Equation 8.3) vs. the *Mean-Derived* and Normalized Density Calculation (Equation 8.4) (a comparison of the individual classes in CIFAR-100 has not been tabulated here due to the large number of classes).

As in the previous section, but instead using the *Mean-Derived* and Normalized Density Calculation (Equation 8.4), another set of experiments was conducted. In these experiments, the training data in each class was reduced using the *Central Exclusion* data reduction strategy by the number of samples necessary to ensure all classes in the training data that had a density greater than a target density value were reduced by the number of samples needed to achieve that target density value. Each of MNIST, Fashion-MNIST, CIFAR-10, CIFAR-100, Imagenette, the micro-PCB dataset, and EMNIST-Digits were subjected to this method using target density levels of 1.1, 1.05, 1.0, 0.95, and 0.9.

Class #	Mean-Derived	Mean-Derived and Normalized
0	1.20360	1.20888
1	0.90039	0.79448
2	0.96696	0.96549
3	1.23407	1.19742
4	0.95438	0.97185
5	0.91778	1.00717
6	1.09510	1.10083
7	1.18253	1.12287
8	0.91090	0.92615
9	0.87517	0.87516

Table 8.13: *Mean-Derived* Density Calculation (Equation 8.3) vs. *Mean-Derived* and Normalized Density Calculation (Equation 8.4) — MNIST

Class #	Mean-Derived	Mean-Derived and Normalized
0	1.18447	1.22587
1	0.69633	0.72067
2	1.04573	1.08228
3	1.16389	1.20457
4	0.96319	0.99685
5	0.96351	0.99719
6	1.11844	1.15753
7	0.99621	1.03103
8	0.90471	0.93633
9	0.85151	0.88128

Table 8.14: *Mean-Derived* Density Calculation (Equation 8.3) vs. *Mean-Derived* and Normalized Density Calculation (Equation 8.4) — EMNIST-Digits

Class #	Mean-Derived	Mean-Derived and Normalized
0	0.67526	1.05405
1	0.69478	1.09361
2	0.56108	0.84936
3	0.53370	0.93503
4	0.59826	0.95569
5	0.59085	0.92904
6	0.70907	1.10913
7	0.69479	1.12181
8	0.56492	0.89294
9	0.74751	1.17047

Table 8.15: *Mean-Derived* Density Calculation (Equation 8.3) vs. *Mean-Derived* and Normalized Density Calculation (Equation 8.4) — Imagenette

Class #	Mean-Derived	Mean-Derived and Normalized
0	0.72625	0.98098
1	0.65924	0.89047
2	0.69837	0.94332
3	0.71100	0.96038
4	0.75716	1.02273
5	0.78188	1.05613
6	0.76589	1.03452
7	0.74160	1.00172
8	0.78661	1.06252
9	0.79986	1.08041

Table 8.16: *Mean-Derived* Density Calculation (Equation 8.3) vs. *Mean-Derived* and Normalized Density Calculation (Equation 8.4) — CIFAR-10

Class #	Mean-Derived	Mean-Derived and Normalized
0	0.27756	1.00728
1	0.26071	0.94615
2	0.25507	0.92565
3	0.26446	0.95975
4	0.27365	0.99307
5	0.30833	1.11895
6	0.27437	0.99571
7	0.28613	1.03837
8	0.28186	1.02288
9	0.27820	1.00959
10	0.27685	1.00467
11	0.27626	1.00253
12	0.27582	1.00096

Table 8.17: *Mean-Derived* Density Calculation (Equation 8.3) vs. *Mean-Derived* and Normalized Density Calculation (Equation 8.4) — micro-PCB

Class #	Mean-Derived	Mean-Derived and Normalized
0	0.78581	0.95250
1	0.78725	0.95424
2	0.88674	1.07483
3	0.94408	1.14435
4	0.80576	0.97669
5	0.82540	1.00049
6	0.55017	0.66688
7	1.21943	1.47810
8	0.72068	0.87355
9	1.09453	1.32671

Table 8.18: *Mean-Derived* Density Calculation (Equation 8.3) vs. *Mean-Derived* and Normalized Density Calculation (Equation 8.4) — Fashion-MNIST

8.2.2 Experimental Results

Table 8.19 through Table 8.25 show the results of these experiments. Bold typeface in the Accuracy columns indicate that the accuracy matched or exceeded the accuracy of the baseline.

For the MNIST dataset, there was no statistically significant difference ($p < 0.05$) between the accuracy of the baseline and the accuracies of any of the experiments with target densities. These target densities resulted in reducing the training dataset size by 4.2%, 8.0%, 11.8%, 17.2%, and 24.2%, respectively.

For the Fashion-MNIST dataset, all target densities were statistically significantly inferior to the baseline ($p < 0.05$). This is consistent with the results from the prior section as the highest target density resulted in removing 15% of the data whereas the only statistically insignificant data reduction using Equation 8.3 elided just 2% of the data.

For the CIFAR-10 dataset, there was no statistically significant difference ($p < 0.05$) between the accuracy of the baseline and the accuracies of the experiments with target densities 1.1, 1.05, and 1.0. These target densities resulted in reducing the training dataset size by 0.2%, 1.1%, and 5.4%, respectively.

For the CIFAR-100 dataset, all target densities were statistically significantly inferior to the baseline ($p < 0.05$). This is consistent with the results from the prior section as the highest target density resulted in removing 5.7% of the data whereas statistically insignificant data reduction using Equation 8.3 was only achieved when eliding 2.3% of the data.

For the Imagenette dataset, there was no statistically significant difference ($p < 0.05$) between the accuracy of the baseline and the accuracies of the experiments with target density 1.1. This target density resulted in reducing the training dataset size by 1.7%. This is inconsistent with the results from the prior section as statistically insignificant data reduction using Equation 8.3 was achieved when eliding as much as 9.2% of the data.

For the micro-PCB dataset, all target densities achieved 100% test accuracy while reducing the training dataset size by 0.5%, 0.5%, 2%, 8.4%, and 19.9%, respectively.

For the EMNIST-Digits dataset, there was no statistically significant difference ($p < 0.05$) between the accuracy of the baseline and the accuracies of the experiments with target densities 1.1 and 1.05. These target densities resulted in reducing the training dataset size by 5.1% and 8.5%, respectively. After 5 trials, a statistical significance to the superior accuracy of the experiment with target density of 1.0 was narrowly missed ($p < 0.07$). Of worthy note for this dataset is the remarkable consistency and low variance across trials for the baseline (which all achieved the same accuracy of 99.79%) and across trials for each of the target densities. The trials for target density

1.1 produced two accuracies of 99.79% and three accuracies of 99.78%. The trials for target density 1.05 produced one accuracy of 99.79%, three accuracies of 99.78%, and the lowest accuracy of all trials of 99.76%. The trials for target density 1.0 produced two of the highest accuracies at 99.8% and three at 99.79%. The trials for target density 0.95 produced three accuracies of 99.78% and two of 99.77%. The trials for target density 0.9 produced one accuracy of 99.79%, two accuracies of 99.78%, and two accuracies of 99.77%.

Table 8.19: Class Accuracies Achieved at Target Densities — MNIST

Target Density ($\overline{d_i}$)	# Samples Included	Accuracy	Std. Dev.	p-value
N/A	100.0%	99.716%	0.000162481	—
1.10	95.8%	**99.722%**	0.000172047	0.31288
1.05	92.0%	99.714%	0.000101980	0.42001
1.00	88.2%	**99.730%**	0.000209762	0.16106
0.95	82.8%	**99.720%**	0.000167332	0.37022
0.90	75.8%	99.706%	0.000185472	0.22038

Table 8.20: Class Accuracies Achieved at Target Densities — Fashion-MNIST

Target Density ($\overline{d_i}$)	# Samples Included	Accuracy	Std. Dev.	p-value
N/A	100.0%	93.404%	0.001380724	—
1.10	85.0%	84.058%	0.004762100	1.35×10^{-10}
1.05	82.6%	83.752%	0.001151347	3.16×10^{-14}
1.00	74.6%	83.526%	0.001400857	5.39×10^{-14}
0.95	74.2%	82.834%	0.001293986	2.30×10^{-14}
0.90	65.8%	81.716%	0.002239286	1.44×10^{-13}

Table 8.21: Class Accuracies Achieved at Target Densities — CIFAR-10

Target Density ($\overline{d_i}$)	# Samples Included	Accuracy	Std. Dev.	p-value
N/A	100.0%	89.146%	0.001518684	—
1.10	99.8%	89.104%	0.001504128	0.35230
1.05	98.9%	**89.166%**	0.002129413	0.44112
1.00	94.6%	89.130%	0.002399167	0.45652
0.95	86.6%	88.742%	0.002066301	0.00679
0.90	75.8%	88.140%	0.001052616	2.24×10^{-6}

Table 8.22: Class Accuracies Achieved at Target Densities — CIFAR-100

Target Density ($\overline{d_i}$)	# Samples Included	Accuracy	Std. Dev.	p-value
N/A	100.0%	61.896%	0.001786169	—
1.10	94.3%	61.194%	0.002620382	0.00110
1.05	90.2%	60.736%	0.001651181	6.04×10^{-6}
1.00	84.5%	58.470%	0.001255388	5.78×10^{-10}
0.95	78.0%	57.134%	0.004187410	1.43×10^{-8}
0.90	70.1%	55.368%	0.003521023	3.81×10^{-10}

Table 8.23: Class Accuracies Achieved at Target Densities — Imagenette

Target Density ($\overline{d_i}$)	# Samples Included	Accuracy	Std. Dev.	p-value
N/A	100.0%	92.390%	0.002333238	—
1.10	98.3%	92.224%	0.000705975	0.10516
1.05	94.2%	92.126%	0.001473228	0.04602
1.00	88.8%	92.080%	0.000748331	0.01762
0.95	82.7%	91.316%	0.002465441	0.00011
0.90	73.9%	90.428%	0.002318103	1.12×10^{-6}

Table 8.24: Class Accuracies Achieved at Target Densities — micro-PCB

Target Density ($\overline{d_i}$)	# Samples Included	Accuracy	Std. Dev.	p-value
N/A	100.0%	100.000%	0	—
1.10	99.5%	**100.000%**	0	N/A
1.05	99.5%	**100.000%**	0	N/A
1.00	98.0%	**100.000%**	0	N/A
0.95	91.6%	**100.000%**	0	N/A
0.90	80.1%	**100.000%**	0	N/A

Table 8.25: Class Accuracies Achieved at Target Densities — EMNIST-Digits

Target Density ($\overline{d_i}$)	# Samples Included	Accuracy	Std. Dev.	p-value
N/A	100.0%	99.790%	0	—
1.10	94.9%	99.784%	4.90×10^{-5}	0.01998
1.05	91.5%	99.778%	9.80×10^{-5}	0.01998
1.00	87.2%	**99.794%**	4.90×10^{-5}	0.07056
0.95	80.7%	99.776%	4.90×10^{-5}	0.00022
0.90	73.4%	99.778%	7.48×10^{-5}	0.00624

8.3 Proposed Dataset Completeness Metric

The experiments in this chapter have demonstrated that, for the datasets studied, there exists an amount of redundant data in each such dataset and that that data can be located near the centroid of a dimensional reduction of the data.

This observation inspires the possibility of a measure of the dataset completeness that can be used *a priori* to determine whether or not the data distributed in the lower-dimensional space is capable of leading to optimal or near-optimal classification accuracy.

Equation 8.5 represents an attempt to capture this measure numerically in a single value, such that the closer the value is to zero, the completeness of the dataset is lower, or from another point of view, there exists room for improvement. In this equation, d are the densities of the classes as calculated using Equation 8.4. As such, the equation simply multiplies the standard deviation of the class densities by the range of density values. Thus, and simply, the greater variance among the class densities, the lower the completeness of the data.

$$q = \frac{1}{\sigma_d \cdot (\max(d) - \min(d))} \tag{8.5}$$

Dataset Completeness Calculation

Table 8.26 shows this calculation as applied to the training datasets used in this chapter. MNIST, EMNIST-Digits, CIFAR-10, and the micro-PCB dataset all had values for completeness > 10 and all of these datasets were shown to have had excess data in their training data sets. Fashion-MNIST and CIFAR-100 each had values for completeness < 10 and both of these datasets were shown not to have had an excess of data in their training data sets. Imagenette was inconsistent with this. It is hypothesized that in the case of Imagenette, that the lower number of training samples coupled with the complexity of the features that comprise the subject matter, compared to the other datasets, dominates any ability to elide redundant data. Table 8.27 shows the same calculation as applied to the validation data of the studied data sets. Consistent with the training data, the validation data had values for completeness > 10 for MNIST, EMNIST-Digits, CIFAR-10, and the micro-PCB dataset, and Fashion-MNIST and CIFAR-100 each had values for completeness < 10.

Table 8.26: Statistical Properties and the Completeness of Training Datasets

Training Dataset	Std. Dev.	Range	Completeness
MNIST	0.1304920	0.4143995	18.492560
EMNIST-Digits	0.1470203	0.5051971	13.463625
Fashion-MNIST	0.2176022	0.8112188	5.664984
CIFAR-10	0.0568050	0.1899450	92.680124
CIFAR-100	0.1623155	0.8421434	7.315669
Imagenette	0.1056541	0.3211061	29.475744
micro-PCB	0.0450495	0.1933003	114.835613

Table 8.27: Statistical Properties and the Completeness of Validation Datasets

Validation Dataset	Std. Dev.	Range	Completeness
MNIST	0.1308110	0.4797679	15.933995
EMNIST-Digits	0.1268322	0.3672039	21.471530
Fashion-MNIST	0.2274090	0.8411320	5.227911
CIFAR-10	0.0495897	0.1651205	122.125472
CIFAR-100	0.1321202	0.7995342	9.466595
Imagenette	0.1223012	0.3698493	22.107753
micro-PCB	0.1455047	0.4808525	14.2925951

8.4 Summary

In this chapter, a definition for class density along with several methods for calculating such was put forth. Analysis of seven datasets showed that Equation 8.4, derived from the normalized mean standard deviations of the data points in lower dimensional space, was the appropriate choice for the calculation for density. Additionally, a definition for dataset completeness was put forth in Equation 8.5, which showed that those datasets that met a certain completeness threshold (experimentally demonstrated to be > 10 for the datasets studied) were candidates for eliding redundant data using Equation 8.4. The experiments showed that for all datasets except Imagenette, datasets with a completeness of > 10 could be reduced to a target density of at most 1.0 and achieve accuracy that is statistically insignificantly different from the baseline. Unique to Imagenette is the relatively low number of training samples. That combined with the large image size coupled with the complexity of the features that comprise the subject matter and the assumptions made by UMAP suggest that there is a lower bound on the number of training samples such that any elision whatsoever risks underfitting in the training procedure.

As was the case in chapter 7, a limitation of the approach used in this study is that it was limited to analyzing benchmark datasets of natural images. These benchmarks are thus subject to the potential for sampling bias and latent properties of the labeling process. The next step in this line of research will be to generate synthetic datasets with well understood distributions and then apply the methods discussed to those.

Chapter 9

Conclusion and Future Work

9.1 Conclusion

The advent of convolutional layers led to considerable improvement in the performance of neural networks in image classification tasks as compared to networks composed entirely of fully connected layers [3]. This is correctly attributed to the convolutional layers' ability to extract localized features that are more complicated than a single pixel. The feature extractors do this by assigning meaning to the spatial relationships among pixels that are close to each other. Such meaning is absent when using fully connected layers. As the term "fully connected" implies, in fully connected layers every pixel is able to be associated with every other pixel without regard to their relative positions in the image. Giving meaning to spatial relationships among the pixels can be understood as enforcing constraints upon which neurons are allowed to be associated with each other using trainable parameters. Understood in this way, the success of convolutional neural networks can thus be understood as, in part, resulting from applying constraints on which neurons are allowed to affect other neurons in the next layer.

Homogeneous vector capsules can be interpreted as performing a similar function, at the output stage of a convolutional neural network, as convolutional layers perform at the input stage. In the traditional design of the classification stage of a CNN, every neuron is able to adapt independently during backpropagation. This work hypothesizes that this fact combined with the fact that adaptive gradient descent methods adapt independent learning rates for every parameter imparts two orders of adaptability—or stated another way, "too much" "freedom" (to adapt to the training data). This would indeed result in overfitting and a generalization gap as has been observed when using adaptive gradient descent with CNNs. By reshaping the output of the final convolutional layer into vectors and then connecting those vectors to a classification layer also composed of vectors, groups of n-dimensional vectors of neurons are constrained to train together.

Using HVCs in combination with multiple branches capturing different levels of abstraction and effective receptive fields which were allowed to learn their own weights resulted in establishing new state of the art accuracies for the MNIST dataset for multiple individual models as well as multiple ensembles. This accomplished using 5.5$\times$ fewer parameters, 4$\times$ fewer epochs of training, and using no reconstruction sub-network compared to the previous state-of-the-art capsule network.

This work additionally presented a dataset consisting of high-resolution images of 13 micro-PCBs captured in various rotations and perspectives relative to the camera, with each sample labeled for PCB type, rotation category, and perspective categories. The results of the experiments performed and elucidated on this dataset show that (1) classification of these micro-PCBs from novel rotations and perspectives is possible, but, in terms of perspectives, better accuracy is achieved when networks have been trained on representative examples of the perspectives that will be evaluated. (2) That, even though perspective warp is non-affine, using it as a data augmentation technique in the absence of training samples from actually different perspectives is still effective and improves accuracy. (3) And that using homogeneous vector capsules (HVCs) is superior to using fully connected layers in convolutional neural networks, especially when the subject matter has many sub-components that vary equivariantly (as is the case with micro-PCBs), and when using the full training dataset and applying rotational and perspective warp data augmentation the mean accuracy of the network using HVCs is 4.8% more accurate then when using a fully connected layer for classification.

This work also showed that certain training samples are more informative of class membership than others. These samples can be identified *a priori* to training by analyzing their position in reduced dimensional space relative to the classes' centroids. Specifically, it was demonstrated that samples nearer the classes' centroids are less informative than those that are furthest from it. This led to the definition of *Dataset Severability*, which is a qualitative, yet quantifiable, judgement regarding the separation of classes in a reduced dimensional space. Those datasets that demonstrated high severability all achieved higher accuracies in the experiments in this work compared to the accuracies of those experiments for the datasets that demonstrated low severability.

Finally, a definition for class density along with several methods for calculating such was put forth. Analysis of seven datasets showed that the calculation proposed in chapter 8, derived from the normalized mean standard deviations of the data points in lower dimensional space, was the appropriate choice for the calculation for density. Additionally, in chapter 8, a definition for a proposed dataset completeness metric was put forth, which showed that those datasets that met a certain completeness threshold (experimentally demonstrated to be > 10 for the datasets studied) were candidates for eliding redundant data. The experiments showed that for all datasets except Imagenette, datasets with a completeness metric of > 10 could be reduced to a target density of at most 1.0 and achieve accuracy that is statistically insignificantly different from the baseline. Unique to Imagenette is the relatively low number of training samples. That combined with the large image size and the assumptions made by UMAP suggest that there is a lower bound on the number of training samples such that any elision whatsoever risks underfitting in the training procedure.

In summary, this work showed how to achieve higher or equivalent validation accuracy using a combination of HVCs and *a priori* analysis of training samples by *reducing* the overall training effort both in terms of having to finely tune the optimization method as well as in terms of the number of training samples used.

At a higher level, and in a bigger-picture sense, the work presented in this dissertation has demonstrated methods for making the training of neural networks simpler and less reliant on a "more data is always better" mentality. This in turn has the potential to lead to massive savings in future neural network training efforts and data collection efforts, which in turn leads to massive savings in energy expenditure and carbon footprints associated with those efforts.

9.2 Future Work

Transformers [93] as applied to computer vision tasks [99] is a very new area of study and was not experimentally investigated in this work. As was noted in chapter 3, transformer-based networks are known for needing a large amount of training data. A future direction for what was started in this work will include investigating whether or not HVCs and/or the analytical methods developed in chapter 7 and chapter 8 can reduce this burden.

Additionally, the conclusion of chapter 3 noted that the establishment of datasets that fill the gaps between CIFAR-100 and Imagenet-1K and between Imagenet-1K and the internal giga-scale datasets of large corporations would benefit the computer vision community. Future work should consider seeking grant funding to establish these.

Constraints on the computational resources available limited the study presented in chapter 4 to only three datasets. This study should be expanded to include additional datasets, most notably the Imagenet-1K benchmark. Additional base network architectures beyond Inception v3 and a simple monolithic CNN should also be investigated. Finally, multiple trials of all of these, as well as the experiments already conducted, should be undertaken so that more fine grained comparisons among the results can be made and investigated for statistical significance.

The results presented in chapter 5 should also be extended to additional network architectures, notably architectures capable of dealing with images sized larger than those in MNIST, Fashion-MNIST, CIFAR-10, and CIFAR-100, and then as a result, to additional datasets. Additionally, an analysis of the digits in MNIST in reduced dimensional space relative to the methods elucidated in chapter 7 and chapter 8 should be performed. This analysis should pay special attention to the position of the (a) 88 digits that were predicted incorrectly by at least one model, (b) the 14 digits that were predicted incorrectly more often than not, and (c) the 5 digits that were predicted incorrectly in all models. An additional systematic study on how many branches and after how many convolutions those branches should come after would be beneficial in the case of deeper networks designed for larger images, perhaps investigating the possibility of learning these properties during training.

Chapter 6 included experiments on data that is known to have differing perspectives. Perspective warp is not generally performed in most data augmentation procedures. A systematic study into whether perspective warp data augmentation should be applied using the benchmark datasets used elsewhere in this work should be undertaken.

The work presented in chapter 7 and chapter 8 should be performed on additional datasets. Notably, the Imagenet-1K benchmark. Additionally, synthetic datasets with well understood distributions should be generated and then have these methods applied to them.

Additionally, the work presented in chapter 7 and chapter 8 took advantage of UMAP for dimensionality reduction. Current implementations of UMAP require the entire dataset to fit in memory. Creating an implementation of UMAP that uses mini-batches and stochastic gradient descent to circumvent this requirement should be attempted so that this method can be applied to datasets larger than can fit entirely in memory.

Finally, the work presented in this dissertation focused exclusively on the supervised, classification task. All of the methods discussed and studied should be applied to (a) other computer vision tasks including segmentation, objection detection, and generative models, (b) transformer networks, (c) other learning regimes such as self-supervised and semi-self-supervised learning, pseudo labeling and using teacher-student networks, and (d) non-image data, particularly time-series data. Additionally, the work done on analyzing the dimensional reductions and density of class data should be coupled with research into data augmentation with the goal of using the reduced dimensionality and density information to inform data augmentation strategies.

References

[1] Geoffrey E Hinton, Alex Krizhevsky, and Sida D Wang. “Transforming auto-encoders”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (2011), pp. 44–51.

[2] Sara Sabour, Nicholas Frosst, and Geoffrey E. Hinton. “Dynamic Routing Between Capsules”. In: *Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS 2017)*. 2017.

[3] Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE*. Vol. 86. 11. 1998, pp. 2278–2323.

[4] Yann LeCun, Fu Jie Huang, and L. Bottou. “Learning methods for generic object recognition with invariance to pose and lighting”. In: *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.* Vol. 2. 2004, pp. 97–104. DOI: 10.1109/CVPR.2004.1315150.

[5] A. P. Dempster, N. M. Laird, and D. B. Rubin. “Maximum Likelihood from Incomplete Data Via the EM Algorithm”. In: *Journal of the Royal Statistical Society: Series B (Methodological)* 39.1 (1977), pp. 1–22.

[6] Geoffrey E. Hinton, Sara Sabour, and Nicholas Frosst. “Matrix Capsules with EM Routing”. In: *Proceedings of the 6th International Conference on Learning Representations (ICLR 2018)*. 2018.

[7] Ashia C. Wilson et al. “The Marginal Value of Adaptive Gradient Methods in Machine Learning”. In: *NIPS 2017 - 31st Conference on Neural Information Processing Systems*. 2018.

[8] J Duchi, E Hazan, and Y Singer. “Adaptive subgradient methods for online learning and stochastic optimization”. In: *The Journal of Machine Learning Research* (2011), pp. 2121–2159.

[9] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *Proceedings of the 6th International Conference on Learning Representations (ICLR 2014)*. 2014.

[10] Jinghui Chen and Quanquan Gu. *Closing the Generalization Gap of Adaptive Gradient Methods in Training Deep Neural Networks*. 2018. arXiv: 1806.06763 [cs.LG].

[11] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *ICLR 2015 - International Conference on Learning Representations*. 2015.

[12] Christian Szegedy et al. "Going Deeper with Convolutions". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), pp. 1–9.

[13] Christian Szegedy et al. "Rethinking the Inception Architecture for Computer Vision". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), pp. 2818–2826.

[14] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), pp. 770–778.

[15] Christian Szegedy et al. *Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning*. 2016. arXiv: 1602.07261 [cs.CV].

[16] François Chollet. "Xception: Deep learning with depthwise separable convolutions". In: *CVPR 2017 - Proceedings of the 30th IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 1800–1807.

[17] Adam Byerly, Tatiana Kalganova, and Richard Ott. "The Current State of the Art in Deep Learning for Image Classification: A Review". In: *Computing Conference 2022*. London, UK, 2022.

[18] Adam Byerly and Tatiana Kalganova. "Homogeneous Vector Capsules Enable Adaptive Gradient Descent in Convolutional Neural Networks". In: *IEEE Access* 9 (2021), pp. 48519–48530. DOI: 10.1109/ACCESS.2021.3066842.

[19] Adam Byerly, Tatiana Kalganova, and Ian Dear. "No Routing Needed Between Capsules". In: *Neurocomputing* 463 (2021), pp. 545–553. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2021.08.064.

[20] Adam Byerly, Tatiana Kalganova, and Anthony J. Grichnik. "On the Importance of Capturing a Sufficient Diversity of Perspective for the Classification of Micro-PCBs". In: *Intelligent Decision Technologies*. Vol. 238. Springer Singapore, 2021, pp. 209–219. ISBN: 978-981-16-2765-1.

[21] P. Besl and Ramesh C. Jain. "Three-dimensional object recognition". In: *ACM Comput. Surv.* 17 (1985), pp. 75–145.

[22] R. Chin and C. Dyer. “Model-based recognition in robot vision”. In: *ACM Comput. Surv.* 18 (1986), pp. 67–108.

[23] Hiroshi Murase and Shree K. Nayar. “Visual learning and recognition of 3D”. In: *International Journal of Computer Vision* 14 (1995), pp. 5–24. DOI: 10.1007/BF01421486.

[24] Serge Belongie, Jitendra Malik, and Jan Puzicha. “Matching shapes”. In: *Proceedings of the IEEE International Conference on Computer Vision* 1.July (2001), pp. 454–461. DOI: 10.1109/ICCV.2001.937552.

[25] Ralph Gross et al. “Multi-PIE”. In: *8th IEEE International Conference on Automatic Face & Gesture Recognition*. 2008. ISBN: 9781424421541. DOI: 10.1109/AFGR.2008.4813399.

[26] Taco S. Cohen and Max Welling. “Group Equivariant Convolutional Networks”. In: *Proceedings of the 33rd International Conference on Machine Learning* (2016).

[27] Qishuo Lu et al. “G-CNN: Object Detection via Grid Convolutional Neural Network”. In: *IEEE Access* 5 (2017), pp. 24023–24031. DOI: 10.1109/ACCESS.2017.2770178.

[28] Rosanne Liu et al. “An intriguing failing of convolutional neural networks and the CoordConv solution”. In: *Advances in Neural Information Processing Systems* 2018-Decem (2018), pp. 9605–9616.

[29] Karel Lenc and Andrea Vedaldi. “Understanding Image Representations by Measuring Their Equivariance and Equivalence”. In: *International Journal of Computer Vision* 127.5 (2019), pp. 456–476. DOI: 10.1007/s11263-018-1098-y.

[30] D. G. Lowe. “Object recognition from local scale-invariant features”. In: *Proceedings of the Seventh IEEE International Conference on Computer Vision*. Vol. 2. 1999, pp. 1150–1157. DOI: 10.1109/ICCV.1999.790410.

[31] David H Hubel and Torsten N Wiesel. “Receptive Fields and Functional Architecture of Monkey Striate Cortex”. In: *Journal of Physiology* 195.1 (1968), pp. 215–243.

[32] Vasily Morzhakov and Alexey Redozubov. *An Artificial Neural Network Architecture Based on Context Transformations in Cortical Minicolumns*. 2017. arXiv: 1712.05954 [cs.CV].

[33] Andrew G. Howard et al. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. 2017. arXiv: 1704.04861 [cs.CV].

[34] Prasun Roy et al. *Effects of Degradations on Deep Neural Network Architectures*. 2018. arXiv: 1807.10108 [cs.CV].

[35] Chao Fang, Yi Shang, and Dong Xu. *Improving Protein Gamma-Turn Prediction Using Inception Capsule Networks*. 2018. arXiv: 1806.07341 [q-bio.QM].

[36] K Guruprasad and S Rajkumar. "Beta-and gamma-turns in proteins revisited: a new set of amino acid turn-type dependent positional preferences and potentials". In: *Journal of Biosciences* 25 (2000), pp. 143–156.

[37] Prem Nair, Rohan Doshi, and Stefan Keselj. *Pushing the Limits of Capsule Networks*. Tech. rep. 2018.

[38] Han Xiao, Kashif Rasul, and Roland Vollgraf. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. 2017. arXiv: 1708.07747 [cs.LG].

[39] Yuval Netzer et al. "Reading Digits in Natural Images with Unsupervised Feature Learning". In: *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*. 2011.

[40] Alex Krizhevsky. *Learning Multiple Layers of Features from Tiny Images*. Tech. rep. 2009.

[41] Zhenhua Chen and David Crandall. *Generalized Capsule Networks with Trainable Routing Procedure*. 2018. arXiv: 1808.08692 [cs.CV].

[42] B. Zhou et al. "BBN: Bilateral-Branch Network With Cumulative Learning for Long-Tailed Visual Recognition". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 9716–9725. DOI: 10.1109/CVPR42600.2020.00974.

[43] C. Wang et al. "CSPNet: A New Backbone that can Enhance Learning Capability of CNN". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2020, pp. 1571–1580. DOI: 10.1109/CVPRW50498.2020.00203.

[44] Dan C. Ciresan, Ueli Meier, and Jürgen Schmidhuber. "Multi-Column Deep Neural Networks for Image Classification". In: *2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2012), pp. 3642–3649.

[45] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. *Highway Networks*. 2015. arXiv: 1505.00387 [cs.LG].

[46] S. Xie et al. "Aggregated Residual Transformations for Deep Neural Networks". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 5987–5995. DOI: 10.1109/CVPR.2017.634.

[47] S. Jégou et al. "The One Hundred Layers Tiramisu: Fully Convolutional DenseNets for Semantic Segmentation". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (2017), pp. 1175–1183.

[48] Hang Zhang et al. *ResNeSt: Split-Attention Networks*. 2020. arXiv: `2004.08955 [cs.CV]`.

[49] J Denker et al. "Large Automatic Learning, Rule Extraction, and Generalization". In: *Complex Systems* 1 (1987), pp. 877–922.

[50] P.Y. Y Simard, D. Steinkraus, and J. C. C. Platt. "Best practices for convolutional neural networks applied to visual document analysis". In: *Proceedings of the Seventh International Conference on Document Analysis and Recognition, 2003*. Vol. 1. 2003, pp. 958–963. DOI: `10.1109/ICDAR.2003.1227801`.

[51] N Chawla et al. "SMOTE: Synthetic minority over-sampling technique". In: *Journal of Artificial Intelligence Research* 16 (2002), pp. 321–357. DOI: `10.1613/jair.953`.

[52] Sebastien C Wong and Mark D Mcdonnell. *Understanding Data Augmentation for Classification: When to Warp?* 2016. arXiv: `1609.08764 [cs.CV]`.

[53] Ekin D Cubuk et al. *AutoAugment: Learning Augmentation Policies from Data*. 2018. arXiv: `1805.09501 [cs.CV]`.

[54] Chaoyue Wang et al. "Tag Disentangled Generative Adversarial Networks for object image re-rendering". In: *IJCAI 2017 - Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*. 2017, pp. 2901–2907.

[55] Alexander J Ratner et al. *Learning to Compose Domain-Specific Transformations for Data Augmentation*. 2017. arXiv: `1709.01643 [stat.ML]`.

[56] Joseph Lemley and Peter Corcoran. *Smart Augmentation - Learning an Optimal Data Augmentation Strategy*. 2017. arXiv: `1703.08383 [cs.AI]`.

[57] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *NIPS 2012 - 25th Conference on Neural Information Processing Systems*. 2012, pp. 1097–1105. DOI: `10.1145/3065386`.

[58] Christopher Pramerdorfer. and Martin Kampel. "PCB Recognition using Local Features for Recycling Purposes". In: *Proceedings of the 10th International Conference on Computer Vision Theory and Applications - Volume 1: VISAPP, (VISIGRAPP 2015)*. INSTICC. SciTePress, 2015, pp. 71–78. ISBN: 978-989-758-091-8. DOI: `10.5220/0005289200710078`.

[59] Hangwei Lu et al. "FICS-PCB: A Multi-Modal Image Dataset for Automated Printed Circuit Board Visual Inspection". In: *Cryptology ePrint Archive, Report 2020/366* (2020). `https://eprint.iacr.org/2020/366`.

[60] Christopher Pramerdorfer and Martin Kampel. “A dataset for computer-vision-based PCB analysis”. In: *Proceedings of the 14th IAPR International Conference on Machine Vision Applications, MVA 2015* (2015), pp. 378–381. DOI: 10.1109/MVA.2015.7153209.

[61] Gayathri Mahalingam, Kevin Marshall Gay, and Karl Ricanek. “PCB-METAL: A PCB image dataset for advanced computer vision machine learning component analysis”. In: *Proceedings of the 16th International Conference on Machine Vision Applications, MVA 2019* (2019). DOI: 10.23919/MVA.2019.8757928.

[62] Wei Li, Bernhard Esders, and Matthias Breier. “SMD segmentation for automated PCB recycling”. In: *IEEE International Conference on Industrial Informatics (INDIN)* (2013), pp. 65–70. DOI: 10.1109/INDIN.2013.6622859.

[63] Daniel Herchenbach, Wei Li, and Matthias Breier. “Segmentation and classification of THCs on PCBAs”. In: *IEEE International Conference on Industrial Informatics (INDIN)* (2013), pp. 59–64. DOI: 10.1109/INDIN.2013.6622858.

[64] Wei Li et al. “Text recognition for information retrieval in images of printed circuit boards”. In: *IECON Proceedings (Industrial Electronics Conference)* (2014), pp. 3487–3493. DOI: 10.1109/IECON.2014.7049016.

[65] Weibo Huang and Peng Wei. *A PCB Dataset for Defects Detection and Classification*. 2019. arXiv: 1901.08204 [cs.CV].

[66] T. Cover and P. Hart. “Nearest neighbor pattern classification”. In: *IEEE Transactions on Information Theory* 13.1 (1967), pp. 21–27. DOI: 10.1109/TIT.1967.1053964.

[67] P. Hart. “The condensed nearest neighbor rule (Corresp.)” In: *IEEE Transactions on Information Theory* 14.3 (1968), pp. 515–516. DOI: 10.1109/TIT.1968.1054155.

[68] G. Ritter et al. “An algorithm for a selective nearest neighbor decision rule (Corresp.)” In: *IEEE Transactions on Information Theory* 21.6 (1975), pp. 665–669. DOI: 10.1109/TIT.1975.1055464.

[69] Dennis L. Wilson. “Asymptotic Properties of Nearest Neighbor Rules Using Edited Data”. In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-2.3 (1972), pp. 408–421. DOI: 10.1109/TSMC.1972.4309137.

[70] D Randall Wilson and Tony R Martinez. “Reduction Techniques for Instance-Based Learning Algorithms”. In: *Machine Learning* 38 (2000), pp. 257–286.

[71] María Teresa Lozano Albalate. “Data reduction techniques in classification processes”. PhD thesis. 2007.

[72] Fernando Vázquez, J. Salvador Sánchez, and Filiberto Pla. “A Stochastic Approach to Wilson’s Editing Algorithm”. In: *Pattern Recognition and Image Analysis*. 2005, pp. 35–42. ISBN: 978-3-540-32238-2.

[73] Chien-Hsing Chou, Bo-Han Kuo, and Fu Chang. “The Generalized Condensed Nearest Neighbor Rule as A Data Reduction Method”. In: *18th International Conference on Pattern Recognition (ICPR’06)*. Vol. 2. 2006, pp. 556–559. DOI: 10.1109/ICPR.2006.1119.

[74] Stefanos Ougiaroglou and Georgios Evangelidis. “Efficient dataset size reduction by finding homogeneous clusters”. In: *Balkan Conference in Informatics (BCI)*. 2012, pp. 168–173. ISBN: 9781450312400. DOI: 10.1145/2371316.2371349.

[75] Mohammad Amin Shayegan and Saeed Aghabozorgi. “A new dataset size reduction approach for PCA-based classification in OCR application”. In: *Mathematical Problems in Engineering* (2014). ISSN: 15635147. DOI: 10.1155/2014/537428.

[76] Aisha Abdallah, Mohd Aizaini Maarof, and Anazida Zainal. “Fraud detection system: A survey”. In: *Journal of Network and Computer Applications* 68 (2016), pp. 90–113. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2016.04.007.

[77] Zhenchuan Li et al. “A hybrid method with dynamic weighted entropy for handling the problem of class imbalance with overlap in credit card fraud detection”. In: *Expert Systems with Applications* 175 (2021). ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2021.114750.

[78] Chumphol Bunkhumpornpat and Krung Sinapiromsaran. “DBMUTE: density-based majority under-sampling technique”. In: *Knowledge and Information Systems* 50 (2017), pp. 827–850. ISSN: 0219-3116. DOI: https://doi.org/10.1007/s10115-016-0957-5.

[79] Hui Han, Wen-Yuan Wang, and Bing-Huan Mao. “Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning”. In: *Advances in Intelligent Computing*. 2005, pp. 878–887. ISBN: 978-3-540-31902-3.

[80] Jia Pengfei, Zhang Chunkai, and He Zhenyu. “A new sampling approach for classification of imbalanced data sets with high density”. In: *2014 International Conference on Big Data and Smart Computing (BIGCOMP)*. 2014, pp. 217–222. DOI: 10.1109/BIGCOMP.2014.6741439.

[81] Richard Hyde and Plamen Angelov. “Data density based clustering”. In: *2014 14th UK Workshop on Computational Intelligence (UKCI)*. 2014, pp. 1–7. DOI: 10.1109/UKCI.2014.6930157.

[82] Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. “On the Surprising Behavior of Distance Metrics in High Dimensional Space”. In: *Database Theory — ICDT 2001*. 2001, pp. 420–434. ISBN: 978-3-540-44503-6.

[83] Leopoldo Bertossi, Flavio Rizzolo, and Lei Jiang. “Data Quality Is Context Dependent”. In: *Enabling Real-Time Business Intelligence*. 2011, pp. 52–67. ISBN: 978-3-642-22970-1.

[84] Leopoldo Bertossi and Floris Geerts. “Data Quality and Explainable AI”. In: *Journal of Data and Information Quality* 12.2 (2020). ISSN: 1936-1955. URL: https://doi.org/10.1145/3386687.

[85] Alexander Borek et al. “A classification of data quality assessment methods”. In: *The 16th International Conference on Information Quality (ICIQ)*. 2011, pp. 189–203.

[86] Roger Blake and Paul Mangiameli. “The Effects and Interactions of Data Quality and Problem Complexity on Classification”. In: *Journal of Data and Information Quality* 2.2 (2011). ISSN: 1936-1955. URL: https://doi.org/10.1145/1891879.1891881.

[87] Leland McInnes, John Healy, and James Melville. *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction*. 2018. arXiv: 1802.03426 [stat.ML].

[88] Lucas Beyer et al. *Are we done with ImageNet?* 2020. arXiv: 2006.07159 [cs.CV].

[89] Jia Deng et al. “ImageNet: A large-scale hierarchical image database”. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2010, pp. 248–255. ISBN: 9781424439911. DOI: 10.1109/cvpr.2009.5206848.

[90] Marcin Kardas et al. “AxCell: Automatic Extraction of Results from Machine Learning Papers”. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*. 2020, pp. 8580–8594. DOI: 10.18653/v1/2020.emnlp-main.692.

[91] Mingxing Tan and Quoc V. Le. *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. 2019. arXiv: 1905.11946 [cs.LG].

[92] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[93] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 5999–6009.

[94] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423.

[95] Xiaohua Zhai et al. *Scaling Vision Transformers*. 2021. arXiv: 2106.04560 [cs.CV].

[96] Carlos Riquelme et al. *Scaling Vision with Sparse Mixture of Experts*. 2021. arXiv: 2106.05974 [cs.CV].

[97] Michael S. Ryoo et al. *TokenLearner: What Can 8 Learned Tokens Do for Images and Videos?* 2021. arXiv: 2106.11297 [cs.CV].

[98] Chao Jia et al. “Scaling Up Visual and Vision-Language Representation Learning With Noisy Text Supervision”. In: *Proceedings of the 38th International Conference on Machine Learning (PMLR)*. 2021.

[99] Alexey Dosovitskiy et al. “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”. In: *Ninth International Conference on Learning Representations (ICLR)*. 2020.

[100] Alexander Kolesnikov et al. “Big Transfer (BiT): General Visual Representation Learning”. In: *16th European Conference on Computer Vision*. 2020.

[101] Xiaoyi Dong et al. *CSWin Transformer: A General Vision Transformer Backbone with Cross-Shaped Windows*. 2021. arXiv: 2107.00652 [cs.CV].

[102] Ze Liu et al. “Swin Transformer: Hierarchical Vision Transformer using Shifted Windows”. In: *The International Conference on Computer Vision (ICCV)*. 2021.

[103] Li Yuan et al. *VOLO: Vision Outlooker for Visual Recognition*. 2021. arXiv: 2106.13112 [cs.CV].

[104] Hugo Touvron et al. “Going deeper with Image Transformers”. In: *The International Conference on Computer Vision (ICCV)*. 2021.

[105] Zihang Jiang et al. *All Tokens Matter: Token Labeling for Training Better Vision Transformers*. 2021. arXiv: 2104.10858 [cs.CV].

[106] Hangbo Bao, Li Dong, and Furu Wei. *BEiT: BERT Pre-Training of Image Transformers*. 2021. arXiv: 2106.08254 [cs.CV].

[107] Prajit Ramachandran et al. “Stand-alone self-attention in vision models”. In: *33rd Conference on Neural Information Processing Systems (NeurIPS 2019)*. 2019.

[108] Tom Henighan et al. *Scaling Laws for Autoregressive Generative Modeling*. 2020. arXiv: 2010.14701 [cs.LG].

[109] Robert A. Jacobs et al. “Adaptive Mixtures of Local Experts”. In: *Neural Computation* 3.1 (1991), pp. 79–87. ISSN: 0899-7667. DOI: 10.1162/neco.1991.3.1.79.

[110] M.I. Jordan and R.A. Jacobs. “Hierarchical mixtures of experts and the EM algorithm”. In: *Proceedings of 1993 International Conference on Neural Networks (IJCNN-93-Nagoya, Japan)*. Vol. 2. 1993, pp. 1339–1344. DOI: 10.1109/IJCNN.1993.716791.

[111] K Chen, L Xu, and H Chi. “Improved learning algorithms for mixture of experts in multiclass classification”. In: *Neural networks : the official journal of the International Neural Network Society* 12.9 (Nov. 1999), pp. 1229–1252. ISSN: 0893-6080. DOI: 10.1016/s0893-6080(99)00043-x.

[112] Karim Ahmed, Mohammad Haris Baig, and Lorenzo Torresani. “Network of Experts for Large-Scale Image Categorization”. In: *Computer Vision – ECCV 2016*. Springer International Publishing, 2016, pp. 516–532. ISBN: 978-3-319-46478-7.

[113] Xiangxiang Chu et al. *Conditional Positional Encodings for Vision Transformers*. 2021. arXiv: 2102.10882 [cs.CV].

[114] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. “Self-attention with relative position representations”. In: *NAACL HLT 2018 - 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference* 2 (2018), pp. 464–468.

[115] Hongyi Zhang et al. “Mixup”. In: *International Conference on Learning Representations (ICLR)*. 2018.

[116] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. *Distilling the Knowledge in a Neural Network*. 2015. arXiv: 1503.02531 [stat.ML].

[117] Yuxin Wu and Kaiming He. “Group Normalization”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018. URL: https://research.fb.com/publications/group-normalization/.

[118] Siyuan Qiao et al. *Micro-Batch Training with Batch-Channel Normalization and Weight Standardization*. 2019. arXiv: 1903.10520 [cs.CV].

[119] Barret Zoph and Quoc V. Le. “Neural architecture search with reinforcement learning”. In: *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*. 2017.

[120] Hugo Touvron et al. “Fixing the train-test resolution discrepancy: FixEfficientNet”. In: *Advances in Neural Information Processing Systems* 32 (2019).

[121] Mingxing Tan and Quoc V. Le. *EfficientNetV2: Smaller Models and Faster Training*. 2021. arXiv: 2104.00298 [cs.CV].

[122] Hugo Touvron et al. “Fixing the train-test resolution discrepancy”. In: *Advances in Neural Information Processing Systems* 32 (2019). ISSN: 10495258.

[123] Zihang Dai et al. *CoAtNet: Marrying Convolution and Attention for All Data Sizes*. 2021. arXiv: 2106.04803 [cs.CV].

[124] Haiping Wu et al. “CvT: Introducing Convolutions to Vision Transformers”. In: *The International Conference on Computer Vision (ICCV)*. 2021.

[125] Jie Hu, Li Shen, and Gang Sun. “Squeeze-and-Excitation Networks”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2018), pp. 7132–7141. ISSN: 10636919. DOI: 10.1109/CVPR.2018.00745.

[126] Lukasz Kaiser, Aidan N Gomez, and François Chollet. “Depthwise Separable Convolutions for Neural Machine Translation”. In: *ICLR 2018 - International Conference on Learning Representations*. 2018.

[127] Ilya Tolstikhin et al. *MLP-Mixer: An all-MLP Architecture for Vision*. 2021. arXiv: 2105.01601 [cs.CV].

[128] Dan Hendrycks and Kevin Gimpel. *Gaussian Error Linear Units (GELUs)*. 2016. arXiv: 1606.08415 [cs.LG].

[129] Hieu Pham et al. *Meta Pseudo Labels*. 2020. arXiv: 2003.10580 [cs.LG].

[130] Andrew Brock et al. *High-Performance Large-Scale Image Recognition Without Normalization*. 2021. arXiv: 2102.06171 [cs.CV].

[131] Pierre Foret et al. “Sharpness-Aware Minimization for Efficiently Improving Generalization”. In: *Ninth International Conference on Learning Representations (ICLR)*. 2020.

[132] Qizhe Xie et al. “Self-training with noisy student improves imagenet classification”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2020, pp. 10684–10695. DOI: 10.1109/CVPR42600.2020.01070.

[133] Stanislav Fort et al. *Drawing Multiple Augmentation Samples Per Image During Training Efficiently Decreases Test Error*. 2021. arXiv: 2105.13343 [cs.LG].

[134] Dong-Hyun Lee. “Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks”. In: *ICML 2013 Workshop: Challenges in Representation Learning* July 2013 (2013).

[135] Ekin D. Cubuk et al. “Randaugment: Practical automated data augmentation with a reduced search space”. In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*. 2020, pp. 3008–3017. ISBN: 9781728193601. DOI: 10.1109/CVPRW50498.2020.00359.

[136] Nitish Srivastava et al. “Dropout: a simple way to prevent neural networks from overfitting.” In: *The Journal of Machine Learning Research* 15 (2014), pp. 1929–1958.

[137] Gao Huang et al. “Deep networks with stochastic depth”. In: *European Conference on Computer Vision (ECCV)*. 2016, pp. 646–661.

[138] H. J. Scudder. “Probability of Error of Some Adaptive Pattern-Recognition Machines”. In: *IEEE Transactions on Information Theory* 11.3 (1965), pp. 363–371. ISSN: 15579654. DOI: 10.1109/TIT.1965.1053799.

[139] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *Proceedings of the 32nd International Conference on Machine Learning (ICML 2015)*. 2015.

[140] Yang You, Igor Gitman, and Boris Ginsburg. *Large Batch Training of Convolutional Networks*. 2017. arXiv: 1708.03888 [cs.CV].

[141] Yang You et al. “Large Batch Optimization for Deep Learning: Training BERT in 76 minutes”. In: *Eighth International Conference on Learning Representations (ICLR)*. 2019.

[142] Jeremy Bernstein et al. “On the distance between two neural networks and the stability of learning”. In: *Advances in Neural Information Processing Systems*. Vol. 33. 2020, pp. 21370–21381.

[143] Pavel Izmailov et al. “Averaging Weights Leads to Wider Optima and Better Generalization”. In: *Proceedings of the 34th Conference on Uncertainty in Artificial Intelligence (UAI 2018)*. 2018.

[144] Elad Hoffer et al. *Augment your batch: better training with larger batches*. 2019. arXiv: 1901.09335 [cs.LG].

[145] Dami Choi et al. *Faster Neural Network Training with Data Echoing*. 2019. arXiv: 1907.05550 [cs.LG].

[146] Stanisław Jastrzębski et al. “Three Factors Influencing Minima in SGD”. In: *International Conference on Artificial Neural Networks and Machine Learning (ICANN)*. 2018.

[147] Yuanzhi Li, Colin Wei, and Tengyu Ma. “Towards explaining the regularization effect of initial large learning rate in training neural networks”. In: *Advances in Neural Information Processing Systems* (2019).

[148] Samuel L. Smith et al. “On the Origin of Implicit Regularization in Stochastic Gradient Descent”. In: *International Conference on Learning Representations (ICLR)*. 2021.

[149] Jeremy Howard. *Imagenette*. 2018. URL: `https://github.com/fastai/imagenette/` (visited on 11/29/2019).

[150] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. “Food-101 – Mining Discriminative Components with Random Forests”. In: *European Conference on Computer Vision*. 2014.

[151] Min Lin, Qiang Chen, and Shuicheng Yan. *Network In Network*. 2013. arXiv: `1312.4400 [cs.NE]`.

[152] Yann LeCun, Corinna Cortes, and CJ Burges. “MNIST handwritten digit database”. In: *ATT Labs [Online].* 2 (2010). URL: `http://yann.lecun.com/exdb/mnist`.

[153] Sai Raam Venkataraman, S. Balasubramanian, and R. Raghunatha Sarma. “Building Deep Equivariant Capsule Networks”. In: *International Conference on Learning Representations*. 2020.

[154] Mohammed Amer and Tomás Maul. “Path Capsule Networks”. In: *Neural Process Letters* 52 (2020), pp. 545–559. DOI: `10.1007/s11063-020-10273-0`.

[155] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. “Deep Sparse Rectifier Neural Networks”. In: *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS 2011)*. 2011.

[156] Geoffrey E. Hinton. *What’s wrong with convolutional nets?* MIT Tech TV. 2018. URL: `https://techtv.mit.edu/collections/bcs/videos/30698-what-s-wrong-with-convolutional-nets`.

[157] Geoffrey E Hinton. “Learning translation invariant recognition in a massively parallel networks”. In: *PARLE Parallel Architectures and Languages Europe*. Springer Berlin Heidelberg, 1987, pp. 1–13. ISBN: 978-3-540-47144-8.

[158] Geoffrey E Hinton et al. *Improving Neural Networks by Preventing Co-Adaptation of Feature Detectors*. 2012. arXiv: `1207.0580 [cs.NE]`.

[159] Li Wan et al. “Regularization of Neural Networks using DropConnect”. In: *Proceedings of the 30th International Conference on Machine Learning (ICML 2013)*. 2013.

[160] Seyyed Hossein Hasanpour et al. *Lets keep it simple, Using simple architectures to outperform deeper and more complex architectures*. 2016. arXiv: `1608.06037 [cs.CV]`.

[161] Jia-Ren Chang and Yong-Sheng Chen. *Batch-Normalized Maxout Network in Network*. 2015. arXiv: `1511.02583 [cs.CV]`.

[162] Ikuro Sato, Hiroki Nishimura, and Kensuke Yokoi. *APAC: Augmented PAttern Classification with Neural Networks*. 2015. arXiv: `1505.03229 [cs.CV]`.

[163] Zhun Zhong et al. *Random Erasing Data Augmentation*. 2017. arXiv: `1708.04896 [cs.CV]`.

[164] Kamran Kowsari et al. "RMDL: Random Multimodel Deep Learning for Classification". In: *Proceedings of the 2nd International Conference on Information System and Data Mining (ICISDM 2018)*. 2018, pp. 19–28.

[165] Tengyu Ma. "Non-convex Optimization for Machine Learning: Design, Analysis, and Understanding". 2017.

[166] Elad Hazan, K. Y. Levy, and S. Shalev-Shwartz. "On graduated optimization for stochastic nonconvex problems". In: *Proceedings of The 33rd International Conference on Machine Learning*. 2016, pp. 1833–1841.

[167] Sanjeev Arora et al. "Computing a Nonnegative Matrix Factorization - Provably". In: *Society for Industrial and Applied Mathematics* 45 (2016), pp. 1582–1611.

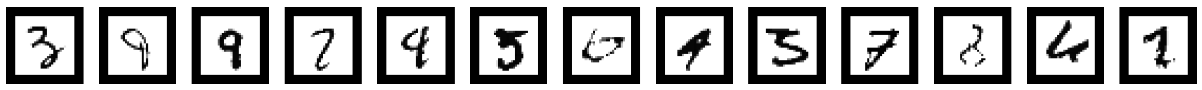
[168] Anima Anandkumar et al. "Tensor decompositions for learning latent variable models". In: *Journal of Machine Learning Research* 15 (2014), pp. 2773–2832.

[169] Laurens van der Maaten and Geoffrey Hinton. "Visualizing Data using t-SNE Laurens". In: *Journal of Machine Learning Research* 9 (2008), pp. 2579–2605.

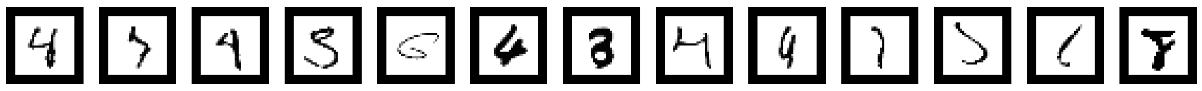
[170] Gregory Cohen et al. *EMNIST: an extension of MNIST to handwritten letters*. 2017. arXiv: `1702.05373 [cs.CV]`.

Appendix A: Digits Disagreed Upon in the Studies Performed in Chapter 5

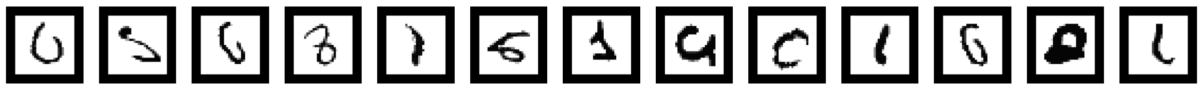
What follows is the complete set of 88 digits that were predicted correctly by at least one model and incorrectly by at least one model in the ensemble of models using Z-Derived capsules from chapter 5. These in combination with the digits from Figure 5.4 represent the complete set of digits that were not predicted correctly by all 96 trials of the Z-Derived ensemble experiments. Each image is captioned first by the class label in the test data set associated with the image, then the number of trials that predicted it correctly, and last the index of the digit in the test data. For example, the first image presented below has a class label of 3, 95 trials predicted that correctly, and it exists at index 87 in the MNIST test data.



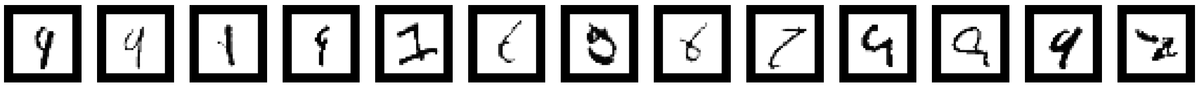
3	9	9	2	9	5	6	4	3	7	8	6	2
95	6	95	24	85	70	92	74	94	95	95	66	95
87	193	214	321	359	412	445	447	449	468	582	625	659



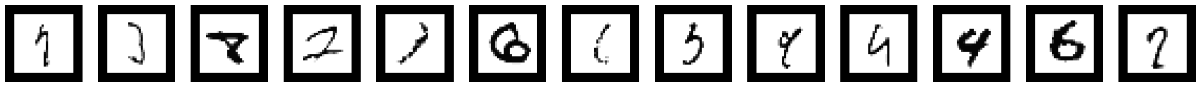
4	7	4	3	6	4	3	4	9	7	5	1	8
93	95	95	95	91	58	95	95	18	94	38	91	95
708	726	740	938	1014	1112	1114	1147	1232	1260	1393	1403	1530



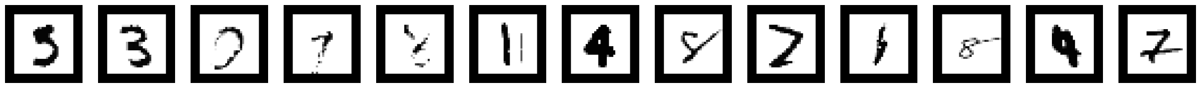
0	5	6	8	1	5	1	9	0	1	6	2	6
95	66	91	94	94	85	95	22	86	90	54	72	74
1621	1737	1822	1878	2018	2040	2266	2293	2326	2355	2454	2462	2654



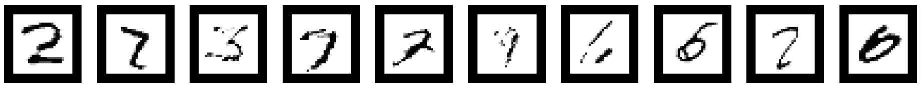
9	4	1	9	1	6	5	6	7	9	9	9	2
85	20	95	92	93	95	92	7	78	95	95	94	82
2720	2771	2803	3005	3073	3365	3558	3762	3808	3821	3859	3869	4176



1	3	8	2	1	6	6	3	9	9	4	6	7
4	93	95	80	84	93	70	49	87	28	95	95	45
4201	4443	4497	4504	4507	4571	4699	4740	4761	4823	4860	4934	5654



5	3	0	7	8	1	4	8	7	1	8	4	7
93	93	95	94	46	92	95	86	30	71	50	8	94
5937	6371	6597	6599	6625	8020	8061	8279	8316	8376	8408	8527	9015



2	7	3	7	2	9	6	5	2	0
91	59	95	90	82	94	94	26	95	78
9123	9505	9636	9637	9664	9692	9698	9729	9839	9850