Empirical Evaluation of Deep Convolutional Neural Networks as Feature Extractors

by

Alfred Kishek

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science
(Computer and Information Science)
in the University of Michigan-Dearborn
2017

Master's Thesis Committee:

Assistant Professor Luis Ortiz, Chair
Professor Willam Grosky
Associate Professor David Yoon

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my thesis advisor Dr. Luis Ortiz of the Department of Computer Information Science at the University of Michigan-Dearborn. He left no question unanswered, and we often found our conversations keeping us at the University late into the night. He instilled me with the focus, determination, and knowledge necessary to complete this research.

I would also like to thank my committee members for their review of this research and their support throughout both my undergraduate and graduate studies.

Finally, I would like to thank my family. Each and everyone of them played a vital role in encouraging me during my pursuit of higher education. Without their patience and support, this accomplishment would not have happened.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

LIST OF ABBREVIATIONS

CNN Convolutional Neural Network

SVM Support Vector Machine

RBF Radial Basis Function

MLP Multilayer Perceptron

DNN Deep Neural Network

ANN Artificial Neural Network

ReLU Rectified Linear Units

LReLU Leaky Rectified Linear Units

SIFT Scale-Invariant Feature Transform

ILSVRC13 ImageNet Large Scale Visual Recognition Challenge 2013

ABSTRACT

Convolutional Neural Networks (CNNs) trained through backpropagation are central to several, competition-winning visual systems. However, these networks often require a very large number of annotated samples, lengthy periods of training, and the estimation and tuning of a plethora of hyperparameters. This thesis evaluates the effectiveness of CNNs as feature extractors and assesses the transferability of their feature maps using a Support Vector Machine (SVM) for evaluation. To compare representations, the parameters learned from a CNN are transfered to various unseen datasets and tasks. The results reveal a significant performance gain on target tasks with small amounts of data. However, as the number of training samples increases, the performance advantage of using the extracted features is diminished and the resulting classifier has high variance.

Chapter I: Introduction

Traditionally, human practitioners single-handedly relied on hand-crafted filters for digital image processing. These filters could capture specific, non-trivial features crucial to a problem. Examples of these features include edge detectors, blob detectors, and corner detectors. Consequently, this allowed researchers to capture and add human insight (bias) into a problem.

Researchers apply these hand-crafted filters to original image representations in order to improve predictions. However, the filters tend to produce very specialized representations for a problem. In other words, there is no silver bullet. Finding the most optimal filter for a specific task is both computationally and labor-intensive.

In recent years, researchers made strides in creating machine learning methods to learn optimal features in a field called *representation learning*. The majority of representation learning conference papers and workshops are dominated by Deep Learning—machine learning methods that leverage Artificial Neural Networks (ANNs) to make predictions and additionally learn representations. Deep Neural Networks (DNNs) have produced state-of-the-art systems in speech recognition, image recognition, and natural language processing.

Despite being known for their success, DNNs are notorious for their lengthy periods of training and millions of hyperparameters. DNNs are a composition of successive linear or nonlinear functions or layers. Each layer produces a set of weights. These weights can be interpreted as a filter. The ultimate goal of training deep networks is to produce abstract, useful representations that are generalizable and transferable to different input distributions.

Explicit in its name, Deep Neural Networks tend to draw inspiration from biological systems, leading to architectures such as the Neocognitron and the CNN. Biological inspiration led the community to the representational interpretation of the model parameters—specifically, the com-

position of abstract, reusable features.

While we draw inspiration from biology and neuroscience to develop new networks, the Neuroscience community is conversely drawing inspiration from Deep Learning (*Marblestone et al.*, 2016). This has led to several hypotheses for how the brain optimizes cost functions. The successfulness of Deep Learning and its biological inspiration begs the question: are these features fundamental to human cognition? Is there a stronger relationship to human neural networks than simply the name?

Before we traverse down the path of human cognition, we must consider how representations derived from these methods are evaluated. Are the representations from these systems truly generalizable to inputs from the same or different distributions?

This thesis provides a framework for evaluating the effectiveness of Deep Learning representations. Specifically, experiments are performed to analyze CNNs used for computer-vision. The learned representations are benchmarked against the original image representations on an unbiased SVM. We then examine the trained SVMs and discuss different properties of the models, such as the average error, variance in performance, usage of support vectors, and the selected parameters.

Chapter II: Preliminaries

This chapter starts with a brief history of neural networks used for computer vision. It then follows with background information about CNNs and transfer learning. The presentation and discussion also includes comments on the intuition behind those concepts.

2.1 History of Neural Networks in Computer Vision

Beginning in the 1950s, *Rosenblatt* (1958) designed mathematical models as solutions to computer-vision tasks. Vision systems have tapped into several classical fields of study including computer science, psychology, neuroscience, physics, robotics, and statistics. Since its conception, an intrinsic theme of computer vision has been to detect and extract feature descriptions from images. Several systems were designed to do so, including Scale-Invariant Feature Transform (SIFT) (*Lowe*, 2004).

Neural networks have been intertwined with computer vision since the perceptron's inception. In 1958, *Rosenblatt* (1958) introduced the "Mark 1 Perceptron"—a machine designed for image recognition. The machine consisted of 400 photocells randomly connected to potentiometers (neurons). Electric motors updated the potentiometers (weights) during training. This led to many innovative ideas, which were largely ahead of their time.

During the 1960s, biological vision was also a topic of exploration. *Hubel and Wiesel* (1962) found that within a cat's visual cortex, simple cells and complex cells fired in response to visual input such as the detection and orientation of edges. This abstract idea served as an inspiration for early neural network vision systems, including the *Neocognitron* (*Fukushima*, 1980). The first layer of the Neocognitron is modeled after simple cells, and the second layer is modeled

after complex cells. The Neocognitron had many features analogous to today's best feedforward deep learning vision systems. Rather than using supervised backpropogation, it was trained using unsupervised learning and consisted of downsampling methods such as spatial averaging. The Neocognitron was the predecessor to CNNs.

In 1992, *Geman et al.* showed the inadequacy of feedforward networks for solving difficult problems in machine perception and machine learning, regardless of the serial vs. parallel hardware required. The *bias-variance dilemma* exposed neural models' representational weaknesses due to the bias introduced from tuning several hyper-parameters. If a proper bias was not introduced, the model could result in having high variance.

CNNs were first developed to recognize spatio temporal patterns in 1988 (*Atlas et al.*, 1987). At this time, the multiplication function for neurons was replaced with convolution. CNNs were later improved in 1998 (*Lecun et al.*, 1998), and generalized and simplified in 2003 (*Simard et al.*, 2003).

It is inefficient to fully connect every node between layers. Unlike multi-layer perceptrons, CNNs exploit the stationarity in smaller, spatial regions of an image by locally connecting neurons between adjacent layers (*Lecun et al.*, 1998). The size of the locally-connected region becomes a hyperparameter called the receptive field of a neuron. Since image statistics are generally translation invariant, units are constrained to the same weights in order to reduce the number of parameters in the CNN (*Lecun et al.*, 1998). This is possible under the assumption that if a single feature patch is useful to compute at a position, then it must be useful to compute another position in the image (e.g. edge filters).

Since the 1990s, many learning algorithms have addressed the bias and variance dilemma. Learning algorithms typically have parameters to tune bias and variance. Additionally, learning methods allow for adding regularization terms, ensemble methods, and alternative methods for handling the bias-variance dilemma. As previously discussed, convolutional neural architectures allow for prior information about a general domain of problems to be crafted into the architectures of the networks. However, neural networks as a whole still experience of millions of hyper-

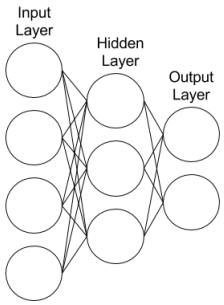

Figure 2.1: A simple MLP with one input layer with four nodes, one hidden layer with three nodes, and finally an output layer with two nodes.

parameters and an infinite number of architectures, making it exceedingly difficult to manage the trade-off between bias and variance.

In recent years, neural networks have adopted several techniques to address the challenges of overfitting, starting with the introduction of unsupervised autoencoders (*Hinton and Zemel*, 1994). Autoencoders learn a compressed encoding of the original data by training the network to reproduce its inputs. They reduce the dimensionality of the input space while extracting features from the image which were captured in the network. Stacking autoencoders as a method of pretraining a neural network can result in better convergence than just randomly initializing weights.

2.2 Traditional Multilayer Perceptrons

Multilayer Perceptrons (MLPs) are ANNs with three or more layers, shown in Figure 2.1. The first layer in the network is the real-valued input. The subsequent layers are called hidden layers. These layers contain hidden nodes have tunable weights and non-linear activation functions. The inputs to the hidden nodes are multiplied by the weights and then passed through an activation function, traditionally a sigmoidal function. The weights are tuned through a method called backpropagation which minimizes the error to the next layer.

All layers in an MLP are typically fully-connected. Since a sigmoidal activation function is used, the decision regions produced by these methods are highly complex. These methods lost their popularity due to their much simpler and related competitor, the SVM.

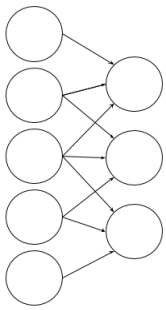

Figure 2.2: A small portion of a CNN showing the local connectivity between layers. This pattern reduces the total number of hyperparameters when compared to a fully-connected network.

2.3 Introduction to Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are very similar to standard MLPs. However, a great bias is introduced due to the assumption that the input to the system is an image. The following sections will discuss this bias and the advancements in architecture. These include local connectivity of nodes, convolutional layers, parameter sharing, and several other architectural modifications.

2.3.1 Local Connectivity

CNNs exploit the problem space by knowing that the input is an image. Therefore, rather than having fully-connected layers, we can take advantage of pixels near one another that are strongly correlated. This is done by locally connecting nodes as shown in Figure 2.2. This local connectivity defines a filter sometimes referred to as a *receptive field*. An image input dimension is represented as a width, height, and depth, where the depth may represent the color channels. The connections are locally-connected across the width and height, but are fully-connected across the depth of the image. For instance, if the image is $28 \times 28 \times 3$ (where 3 represents the RGB color channels) and the size of the filter is 4×4, then every node in the convolutional layer will have $4 \times 4 \times 3$ connections for a total of 48 weights and a bias.

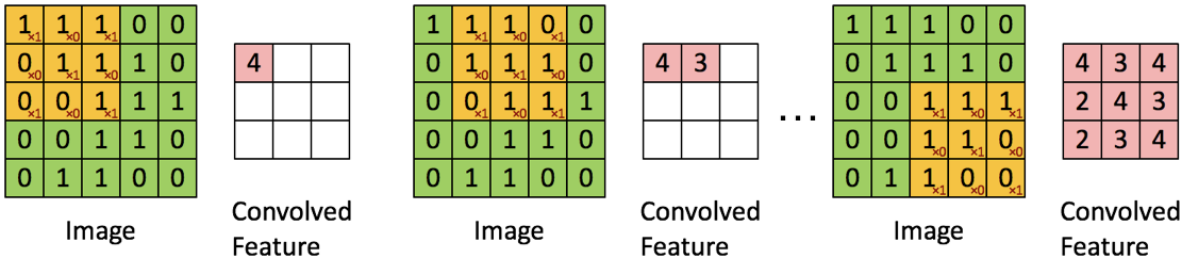

Figure 2.3: This figure demonstrates a convolution within a CNN. The first image shows the initial state of the convolution. The larger numbers in the section labeled "Image" represent the pixel values of the image. The smaller numbers are the weights of the filter. Together these produce a convolved feature. The filter slides over the image producing the final output on the right.

2.3.2 Convolutional Layers

The output dimension of an entire convolutional layer is defined by three parameters. The *depth* is a hyper-parameter corresponding to the number of filters. The *stride* of a filter is responsible for sliding the filter over the image. Finally, the *zero-padding* parameter defines the filter's behavior toward the edges of the image. With knowledge of these three hyperparameters and the input dimension of the layer, the output dimension can be calculated. The convolution of a 5×5 image with a 3×3 filter is shown in Figure 2.3.

2.3.3 Weight Sharing

Colors, usually represented as three channel RGB values from $0 - 255$, are also interrelated. Each color channel is represented by one node (three nodes per color for RGB). For each one of these color values, CNNs tie the weights and biases forming a *feature map*. Similar to the local connectivity, the weight sharing scheme significantly reduces the number of hyperparameters and simultaneously captures the correlated information in the problem space. Parameter sharing is also applied at each layer of the network. For example, if a convolutional layer's output is $28 \times 28 \times 10$, where 10 is the depth of the layer, then it is said to have 10 *slices* of 28×28 images (also called *depth slices*). The parameters are shared across all slices. Because all depth slices use the same

weight vector, then the forward pass of the convolutional layer can be computed as a convolution of the weights along with the input dimension.

Mathematically, the k-th feature map at a hidden layer h is

$$h_{ij}^{k} = f((W^{k} * x)_{ij} + b_k)$$

where the function f can be a linear or nonlinear activation function, and W^k and b_k are the weights and biases at the k-th feature map, respectively.

2.4 Convolutional Neural Network Architecture

This section will discuss components of CNNs in addition to the base architecture described above. These include normalization at hidden layers, different activation functions, and optimizers that will be used in our empirical studies.

2.4.1 Batch Normalization

To make learning smoother, initial values of our parameters are typically normalized. During training, parameters are consistently updated and lose their original normalization effects. Batch Normalization (*Ioffe and Szegedy*, 2015) was introduced to address the issue of hidden layer input distributions that change during training as the previous layers change. Because of this limitation, smaller learning rates were necessary, therefore abating training and proving difficult to train models with saturating nonlinearities. This concept is referred to as *internal covariate shift* (*Ioffe and Szegedy*, 2015). Batch Normalization alleviates the internal covariate shift by introducing normalization as part of the model architecture. As a result, learning rates are reduced, and the normalized layers act as a regularizer, in some cases, eliminating the need for random dropout. Batch normalization should be applied to a layer before any nonlinearities are introduced. In our experiments, batch normalization is used at every convolutional layer before the nonlinear activation functions.

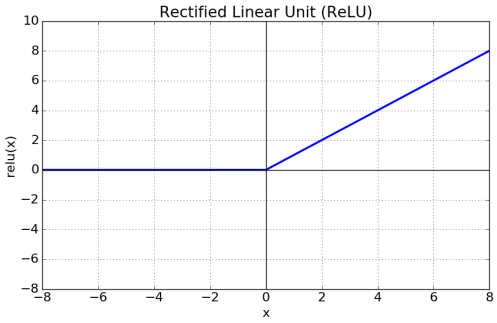

Figure 2.4: A plot of the rectified linear function.

2.4.2 Rectified Linear Units

Rectifier Units (*Nair and Hinton*, 2010) are activation functions given by

$$f(x) = \begin{cases} x, & \text{if } x > 0, \\ 0, & \text{otherwise,} \end{cases}$$

where x is the input to the neuron. The unit with a rectifier activation can be called a *Rectified Linear Units (ReLU)*. These units often perform better compared to its predecessor, the sigmoidal activation. This function can be visualized in Figure 2.4. However, ReLUs introduce a limitation due to the 0 gradient whenever the unit is not activated. This could lead to a dead neuron, where the unit never activates during a gradient-based optimization. To alleviate the 0 activation, *Leaky Rectified Linear Units (LReLU)* (*Nair and Hinton*, 2010) introduce a non-zero gradient and are given by

$$f(x) = \begin{cases} x, & \text{if } x > 0, \\ \varepsilon x, & \text{otherwise,} \end{cases}$$

for some small real-value $\varepsilon > 0$ (e.g., 0.01), causing the gradient to be more resilient during gradient-based optimization and negative input. In the following experiments, LReLU will serve as the activation function in our convolutional model architecture.

2.4.3 Adam Optimizer

Adam (*Kingma and Ba*, 2014) is an algorithm for efficient stochastic optimization using first-order, gradient-based methods. The Adam optimizer computes adaptive learning rates for each parameter. The hyperparameters typically require minor tuning. Empirical studies show the Adam optimizer to be favorable in comparison to other stochastic optimization methods (*Kingma and Ba*, 2014). In the following experiments, the Adam optimizer is used to minimize cross-entropy loss in the CNN architecture.

2.5 Feature Transfer Methods

This section explores different methods for transferring features from a standard CNN. The purpose of transfer learning is to improve the learning of a new task (*Pan and Yang*), called the *target task*, using the knowledge gained from a *source task*.

Generally, the process involves training a CNN and using the weights and biases to learn a solution to a different problem. The Internet contains several deep learning models trained on large datasets such as ImageNet (*Deng et al.*, 2009). The trained models typically require expensive resources and time to build, hoping to capture the bias to exploit it on target tasks.

2.5.1 Fixed-Parameter Transfer

The fixed-parameter transfer method is a way of applying pre-trained CNNs as feature extractors. The process starts with training a CNN on a source task, such as ImageNet, and extracting the weights for all the layers except the last fully-connected layer. This fully-connected layer can be seen as the "classifier" for the network.

Now, using another dataset for training, a forward-pass retrieves the output of the network. The output will be the activations of the hidden layer prior to the classifier. Depending on the architecture, the extracted vector, or representation, can either be smaller or larger than the original input dimension. We can train a new model using this representation as the input. This new

model can be an MLP, thus "fixing" the parameters of the source network and training a new fully-connected portion for the target task.

2.5.2 Fine-Tuning

Similar to the process described above, fine-tuning starts with training a CNN on a source task. However, rather than fixing the parameters and training a new model, we can use the parameters to initialize a network with the same architecture. The parameters are then updated through backpropagation while learning the target task.

The question of whether fine-tuning is appropriate depends on the number of training samples available for the target task. Overfitting occurs when there is a large number of parameters to tune and a small number of training samples. However, if the training dataset is large, then the value of fine-tuning is diminished since the target network can relearn the required features (*Yosinski et al.*, 2014). It is possible to partially fix the network and fine-tune other parts of the network in order to reduce the number of tunable parameters. Choosing which layers to fix and tune is an art which in itself may cause overfitting.

This thesis purely focuses on fixed-parameter feature extraction and analyzing its strengths and weaknesses.

2.6 Related Literature

This section gives a brief overview of the literature regarding regarding Deep Learning within the context of Transfer Learning. The literature covers several topics including successful applications using transfer learning with CNNs, benchmarks for transferability and fine-tuning, and unsupervised transfer learning.

Oquab et al. (2014) used fixed-parameter methods to transfer features learned from a source task (ImageNet) to multiple target tasks with smaller datasets (PASCAL VOC). They reported state-of-the-art performance on both the Pascal VOC 2007 and 2012 datasets.

Oquab et al. (2014) experiment with tuning the number of layers in the final fully-connected classifier, originally two hidden layers, which they call the *adaption layer*. They reported that modifying the adaption layer to one fully-connected layer for the new task classification caused a 1% drop in performance. Modifying it to three hidden layers also resulted in a drop in performance.

In the following experiments, the fixed-parameter transfer methods closely resemble those presented in *Oquab et al.* (2014). However, since modifying the architecture can have a significant effect on performance, the following results are presented using the same CNN architecture.

Razavian et al. (2014) extracts features from a trained network called *OverFeat*. This network was trained on the ImageNet Large Scale Visual Recognition Challenge 2013 (ILSVRC13). A separate linear SVM is trained on the features extracted from the network in order to use the extracted features on several similar datasets and report the results.

Razavian et al. (2014) found that in most cases, features extracted from the CNN outperform features obtained from competing methods such as SIFT. In most cases, however, the transfer methods only achieved state-of-the-art performance when using augmentation methods (e.g. jittering, PCA, normalization, whitening). *Razavian et al.* (2014) recommends that features extracted from CNNs be the primary candidate for most visual recognition tasks.

Their research parallels our experiments in regard to the fixed-parameter transfer and evaluation using an SVM. However, in order to fairly evaluate the features, neither the input or the features extracted were augmented in this research.

Yosinski et al. (2014) investigated the generalization of CNNs at different layers by designing experiments and then attempting to quantify the transferability of certain layers. These transfer experiments involve freezing parameters at certain layers, fine-tuning, training on semantically dissimilar tasks, and benchmarking randomized initialization and the effects with deeper networks.

The reported results showed two main points: (1) challenges in optimization when splitting networks in the middle layers due to *co-adapted layers*, and (2) specialization of higher layers causing a decrease in performance on target tasks. *Yosinski et al.* (2014) also noted that initialization followed by fine-tuning would increase generalization performance.

Yosinski et al. (2014) referenced other research (*Jarrett et al.*, 2009) which showed that initialization with randomized weights and subsequent fine-tuning can perform almost as well as initialization with transfered networks. This is the case with shallower networks (two to three layers) trained with smaller datasets. However, *Yosinski et al.* (2014) stated that when the networks are trained on smaller tasks, such as the Caltech-101 dataset, overfitting may cause a decrease in generalization. *Yosinski et al.* (2014) suggested that if the training dataset is large, then the value of fine-tuning is diminished because the target network can simply relearn the required features.

Transfer methods are typically applied when the target task sample size is small. However, *Yosinski et al.* (2014) hypothesizes that transferring to small datasets and fine-tuning results in overfitting, thus leading to the conclusion that random initialization works as well as transferring. Therefore, it is left unclear at what times fine-tuning is appropriate.

Bengio (2012) studied unsupervised pre-training of representations and transferability using a Transfer Learning Challenge (*Silver et al.*, 2011) for evaluation. In this challenge, the source distribution is very different than the target distribution. The target task does not contain any common labels with the source task. If the representation learning algorithm proves to transfer well to another input distribution, then it will have identified generic features which can be used for other tasks.

Bengio (2012) discusses the intuition behind the depth component of *Deep* Learning. The first referenced inspiration for deeper networks is drawn from biology—simply, since the human brain is not shallow, architectures should have a notion of depth. *Bengio* (2012) also discusses human cognition as a motivation for depth. Representations of concepts at one level of abstraction is construction from a composition of concepts at lower-levels of abstractions. Hence, in Deep Learning, higher-level features are compositions of lower-level features. *Bengio* (2012) suggests it is natural that these algorithms are useful for transfer learning because they contain the idea of "abstract" representations.

This prompts us to revisit the question of whether these features are truly fundamental. Are these features unique and generalizable to any input domain?

Chapter III: Empirical Evaluation

This chapter describes the datasets, tools, and design of the experiments that I conducted for this thesis, followed by an evaluation of the features and a discussion of the results. The experiments that I present in this chapter were specifically designed to assess the effectiveness of CNNs as feature extractors.

3.1 MNIST Dataset

The following experiments use the MNIST dataset (*LeCun and Cortes*, 2010). This dataset is a collection of 70,000, 28×28 handwritten digits (60,000 for training and 10,000 for testing). The MNIST dataset dataset is among the most popular used for evaluating image processing algorithms on real-world handwriting tasks.

Besides popularity, the dataset was chosen for its simplicity in the interest of understanding the transferability of CNNs rather than its ability to navigate the complexity of the input domain. The transfer learning tasks described below only require a single dataset (in our case MNIST), and the experiments are performed on various subsets of the dataset.

In preprocessing, pixel intensities are normalized to values between 0 and 1. To prevent peaking, we first combine the previously divided training and test sets, then randomly resplit them into a training set of $55,000$ samples, a test set of $10,000$ samples, and a validation set of $5,000$ samples.

3.2 Tools

TensorFlow (*Abadi et al.*, 2015) is an open-source library for expressing, implementing, and executing machine-learning algorithms. TensorFlow, developed at Google, is a flexible system

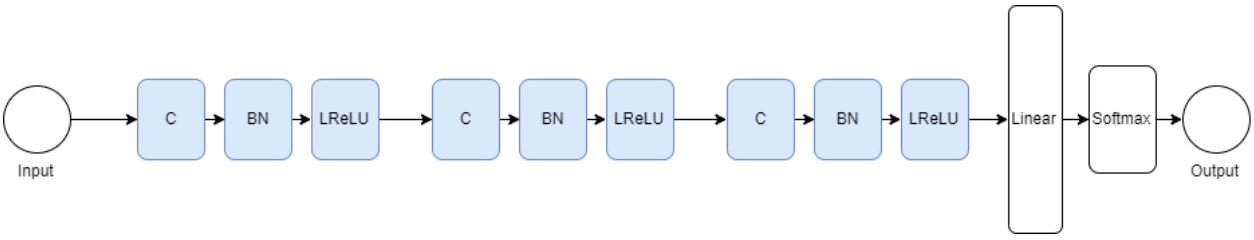

Figure 3.1: The architecture of the convolutional neural network used for the experimentation. The areas in blue represent the convolutional layers and their activations. For feature extraction, a forward pass is used to extract the features from the last shaded node (before the fully-connected linear layer in white).

used to train and make inference algorithms for deep neural-network models. It allows the user to deploy computational needs on several CPUs or GPUs on a desktop, server, or even a mobile application with a single API. TensorFlow has a simple, yet comprehensive set of tools required for building and testing CNNs, which proved suitable for these experiments.

3.3 Network Architecture

The CNN architecture in the following experiments is a combination of the previously discussed modern layers (see Figure 3.1). Each 28×28 image is fed into the network as the input layer. The main network is composed of three successive convolutional layers. Each convolutional layer applies batch normalization, then a LReLU activation function.

The output of these layers is then reshaped into a 1024-length vector and fed into a linear classifier followed by a softmax layer. The network is trained with backpropagation using an Adam optimizer to minimize a cross-entropy loss.

3.4 Experiments

The following sections present the experimental design and transfer learning tasks to evaluate CNNs. It begins with an evaluation of the baseline performance of a CNN and SVM on the original image representation in Section 3.4.1. The representations are then extracted from the trained CNN and compared to representations from the original image using an SVM. The effects of learning

with smaller datasets are then studied. The final experiment concludes with a transfer learning task that divides the class labels into two subsets—the source task and the target task.

3.4.1 Baseline SVM and CNN Performance

This experiment's goal is not to achieve state-of-the-art performance, but to evaluate the representations derived from the CNN. Before we evaluate the features, we start by measuring the baseline performance for both the CNN architecture defined above and the SVM we will used to evaluate the representations.

The CNN is trained on the training dataset for 30 epochs. The accuracy metric is used to measure the baseline performance. In later experiments, the trained network parameters will be saved to allow the use of the CNN as a feature extractor.

A Radial Basis Function (RBF) SVM is a general algorithm used in this experiment to assess the performance of the extracted features (*Cortes and Vapnik*, 1995). The RBF SVM is trained on the same training set used for the CNN. For training, we feed the original 28×28 image as a flattened 784-length vector as input into the SVM. An exhaustive grid search with 10-fold validation is required to tune the penalty term and the kernel coefficient. For performance purposes, the K-fold validation is applied to 10,000 randomly selected samples from the training set.

3.4.2 CNN Extracted Representation vs. Original Representation

The second experiment evaluates the effectiveness of using CNNs as fixed feature extractors. The network is initialized with the parameters learned from the training set in the previous experiment. Using a forward pass, the features are extracted from the output of the layer prior to the linear classification layer. This output will be referred to as the *extracted representation*, with the original image pixel values referred to as the *original representation*. The extracted representation is a vector of 1024-dimensions, a higher dimensional representation than the original image (784-dimensions). Following this method, the original dataset is transformed into a new training, test, and validation dataset with the extracted features.

The effectiveness of the features will be determined by training two models—both RBF SVMs—on the original and extracted representations. Similar to the previous experiment, both SVMs are trained with grid search using 10-fold validation to tune the hyperparameters.

3.4.3 Transfer Learning on Various Sample Sizes

The effectiveness of the extracted features is best demonstrated in experiments where the training sample size is insufficient for generalization. To simulate this, the original training set is sampled to construct smaller training sets of size 50, 100, 500, 1000, and 5000. For each of these sample sizes, a forward pass creates an extracted representation using the trained CNN parameters from the above experiment.

Two RBF SVMs are trained on both the extracted and the original representation for each sample size. Hyperparameters for the SVM are again chosen by performing grid search, however this time using 2-fold validation.

3.4.4 Incremental Transfer to an Unseen Task

To test the incremental transfer between two semantically related datasets, MNIST is divided into two smaller, mutually exclusive datasets composed of five randomly sampled classes. For example, the first set may contain $\{1,2,4,5,9\}$ while the second set may contain $\{0,3,6,7,8\}$. These sets are referred to as set A and set B.

Each subset, A and B, is split into a training set of size 16,000 samples, a test set of size 4,000 samples, and a validation set of 2,000 samples. A CNN with the architecture described in Section 3.3 is trained only on the training set of A. Similar to the previous experiments, the CNN is used to extract a new representation of the data. However, in this case, the representation is only based on the biases learned from set A.

In order to evaluate the features as they are transferring from A to B, a new training set is developed composed of a subset of B with probability p, and a subset of A with probability $1-p$. Initially, the new training set is composed of completely A ($p=0$) and p is incremented by 0.1 for

SVM vs CNN	Average Test Error
RBF Kernel SVM	1.3%
CNN (30 epochs)	**0.8%**

Table 3.1: Comparison of SVM and CNN test error on MNIST.

each following test. A test set is also derived from subsets A and B using the same composition of sets A and B.

Finally, as in the last experiment, the effects of smaller training samples during the transfer are examined. In other words, for each value of p, we also examine the training sample sizes: 50, 100, 500, 1000, 5000, 10000, and 16000. For every experiment, the extracted and original features are evaluated using an SVM with an RBF kernel. The hyperparameters were chosen through an exhaustive grid search method with a 5-fold validation. Each experiment was run 50 times to determine statistical significance.

In addition to studying performance, other metrics such as the number of support vectors and the parameters chosen by the grid-search method are used to evaluate the model.

3.5 Results

This section reports the results from the experiments described above. It starts with an analysis comparing the base performance of CNNs and SVMs on the original pixel image, followed by an empirical examination of the effectiveness of CNNs as feature extractors.

3.5.1 Baseline SVM and CNN Performance

This section examines the average performance of the base SVM and CNN trained on an MNIST training. The test error is shown in Table 3.1. The RBF Kernel SVM with the grid-searched parameters performed worse, with 0.05% higher average test error than the CNN architecture. The CNN parameters learned from the training set are saved and a new dataset is extracted for the next experiment.

Original vs Extracted Features	Average Test Error
Original Representation on RBF SVM	1.3%
Original Representation on CNN	0.8%
Extracted Representation on RBF SVM	**0.6%**

Table 3.2: Comparison of the original features and the features extracted from the CNN.

3.5.2 CNN Extracted Representation vs. Original Representation

This experiment uses a CNN as a feature extractor. A new SVM is then trained using the extracted representation learned in the last experiment. The error on the test set is reported in Table 3.2.

There is a marginal gain in performance by first extracting features from the CNN and then training a non-linear RBF Kernel SVM. The idea of replacing the last layer with a non-linear classifier was demonstrated by *Kontschieder et al.* (2015), who replaced the last layers of a standard CNN with a non-linear tree-based classifier for a reduction in error. The above results suggest this is consistent with another non-linear classifier (the RBF SVM).

3.5.3 Transfer Learning on Various Sample Sizes

The following experiment studies the representations extracted from the CNN by training a separate SVM on various smaller sample sizes. In Figure 3.2, the results show with just 100 samples of the training data, the SVM using the extracted features achieves 96.6% accuracy on a test set, whereas the SVM trained on the original representation achieves 68.9% accuracy.

This result shows that the CNN, as a feature extractor, is capable of capturing bias from the input space, and a separate model is able to leverage the bias for achieving better performance.

3.5.4 Incremental Transfer to an Unseen Task

The results for the experiment described in Section 3.4.4 are reported below. We start with the two subsets, A and B, and train the CNN purely on subset A. As we increment p, a new dataset composed of both A and B is created. This dataset is evaluated using the extracted features (only learned from task A) in addition to the original pixel values.

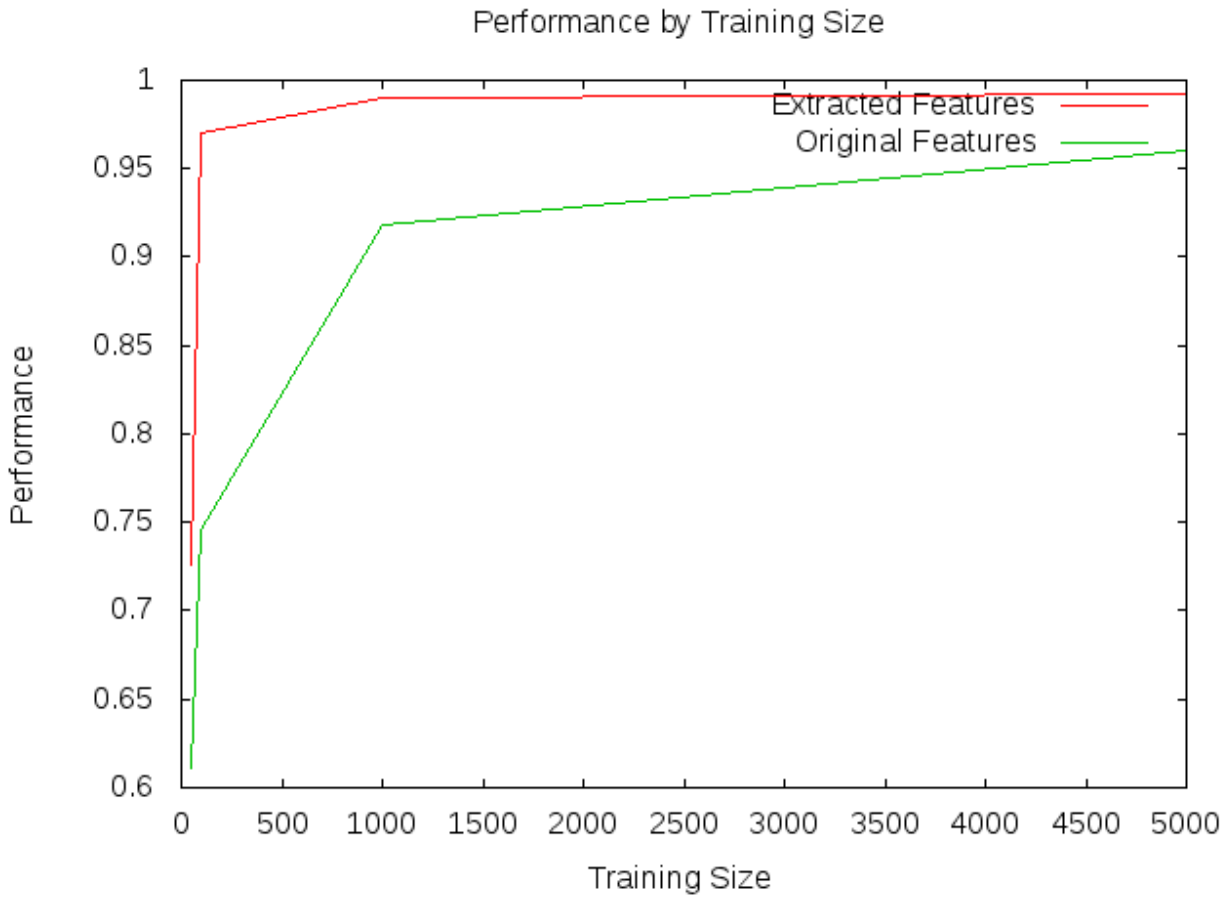

Figure 3.2: A plot showing the extracted features' performance against the original features given the number of training samples.

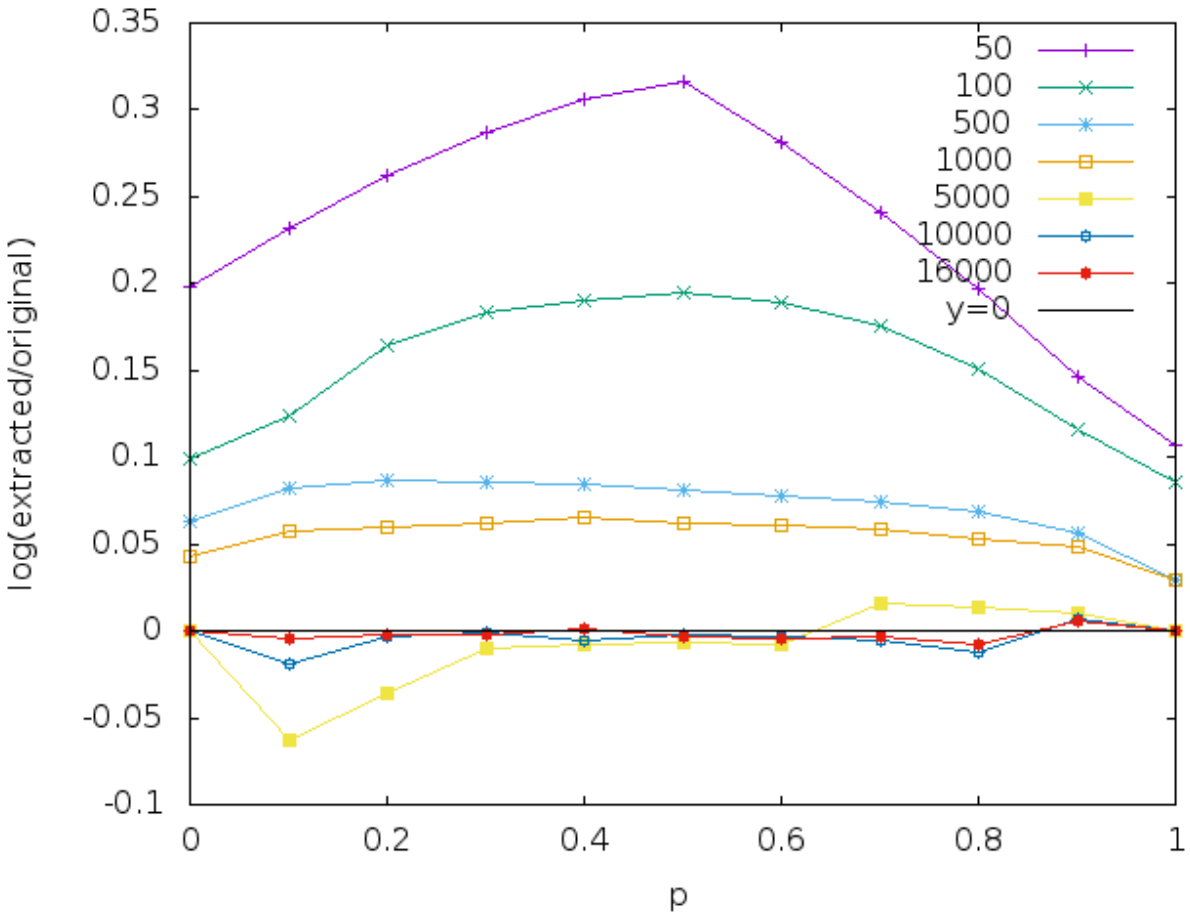

Figure 3.3: The log of the extracted performance over the original performance. When the ratio is positive, the performance of the extracted features are performing better than the original features. It is apparent that as the training sample size increases, the ratio of the performance converges to zero. In other words, the transfer features do not provide additional support when there is enough training data.

Figure 3.3 shows the log ratio of the performance, with the log of the extracted performance divided by the original performance. In many cases, the extracted features consistently outperform than the original features. However, this performance advantage begins to converge to zero as more data is used to train the SVMs used for evaluation. At around 5000 training samples, the original pixel values perform as well as (and sometimes better than) those extracted from the CNN.

If we take a closer look at the performance with 16,000 training samples in Figure 3.4, we can immediately see the high variability in performance when using the extracted representation

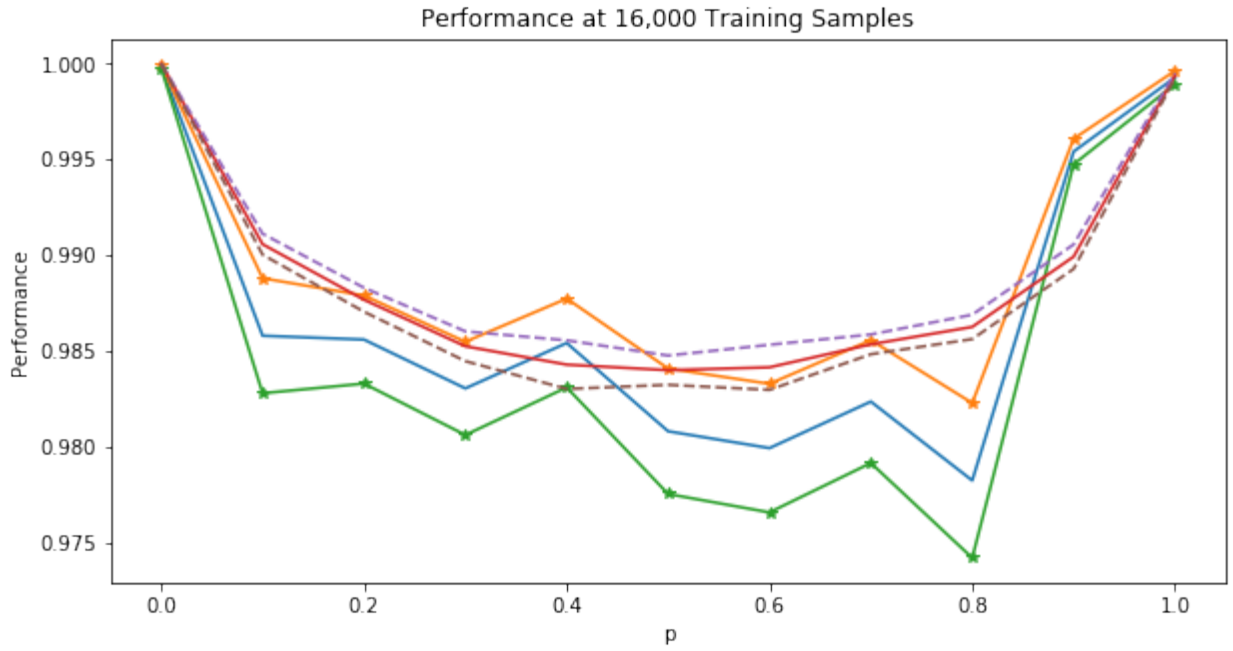

Figure 3.4: The plot of the performance for 16,000 training samples and different transfer sets with 95% confidence intervals over the 50 trials.

over the original representation. And in some cases, using the original pixel values, the mean performance values over 50 trials are significantly larger than the extracted representations. This reveals that even while using the same dataset, a transfer of features could lower performance and cause high variability in the classifier.

Next, the support vectors chosen by the SVMs are assessed. Figure 3.5 shows the number of support vectors chosen by the SVM when trained on the extracted representation against the number of support vectors when using the original representation. When examining the number of support vectors, the number of supports maps to the complexity of the model. As we approach the endpoints of the transfer (where $p = 0$ and $p = 1$), the SVM trained on the extracted features chooses a simpler model relative to the original features. However, in most other cases, the original image representation generates a simpler model (less support vectors). This result maps back to the variability in performance discussed above. The next section evaluates to which degree the chosen supports overlap.

The purpose of the next plot is to determine whether the features learned by the SVM from

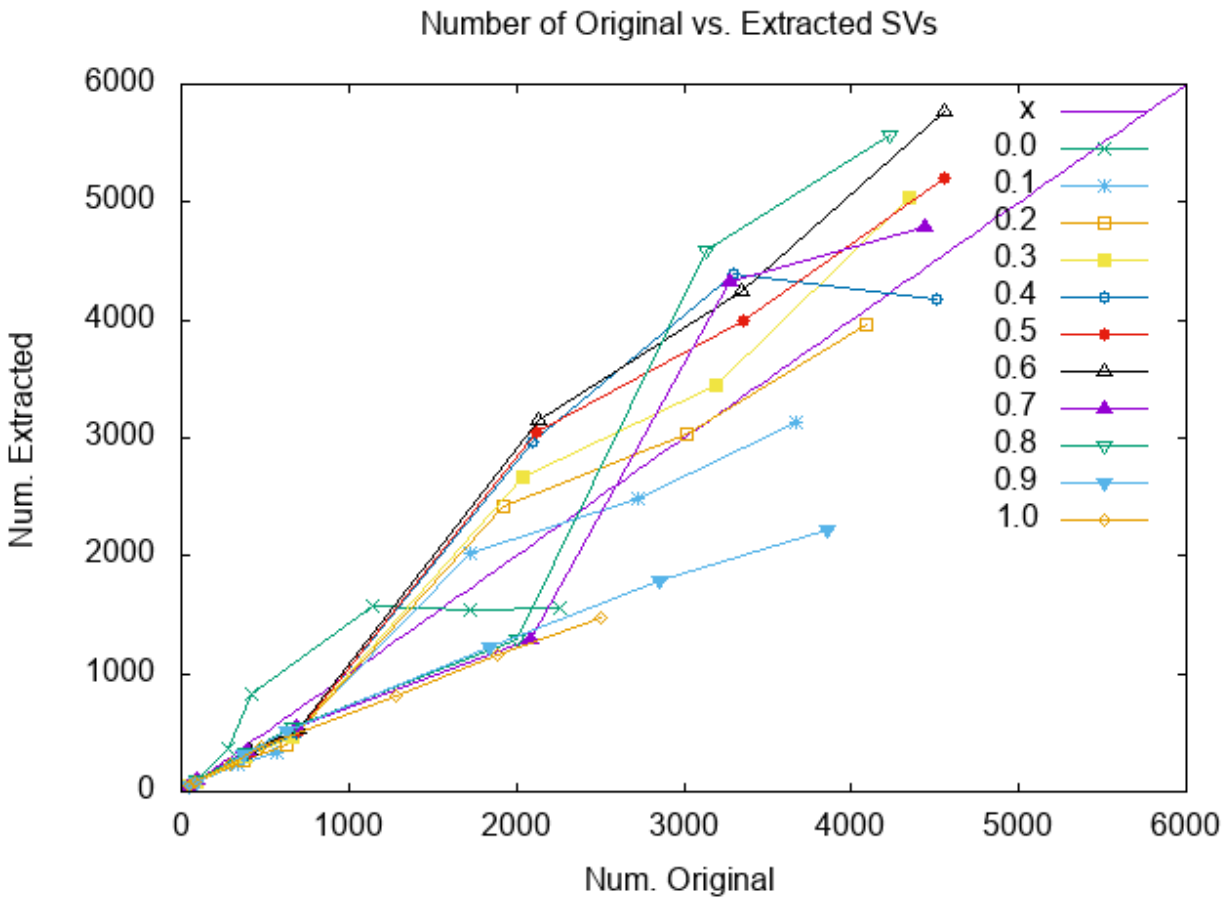

Figure 3.5: The number of support vectors chosen by the SVM when using the extracted representation vs. the number of supports when using the original representation.

the extracted representation are different than those from the original representation. Figure 3.6 reveals the symmetric difference of the selected support vectors. Intuitively, the higher the number, the more different the features. An interesting phenomenon occurs: as the number of training samples increases, the features chosen by the SVMs begin to diverge. In other words, the SVM finds a different set of support vectors that explains the class. As demonstrated in Figure 3.4, the set of supports that are chosen from the original pixel values results in less variability, providing more stable performance.

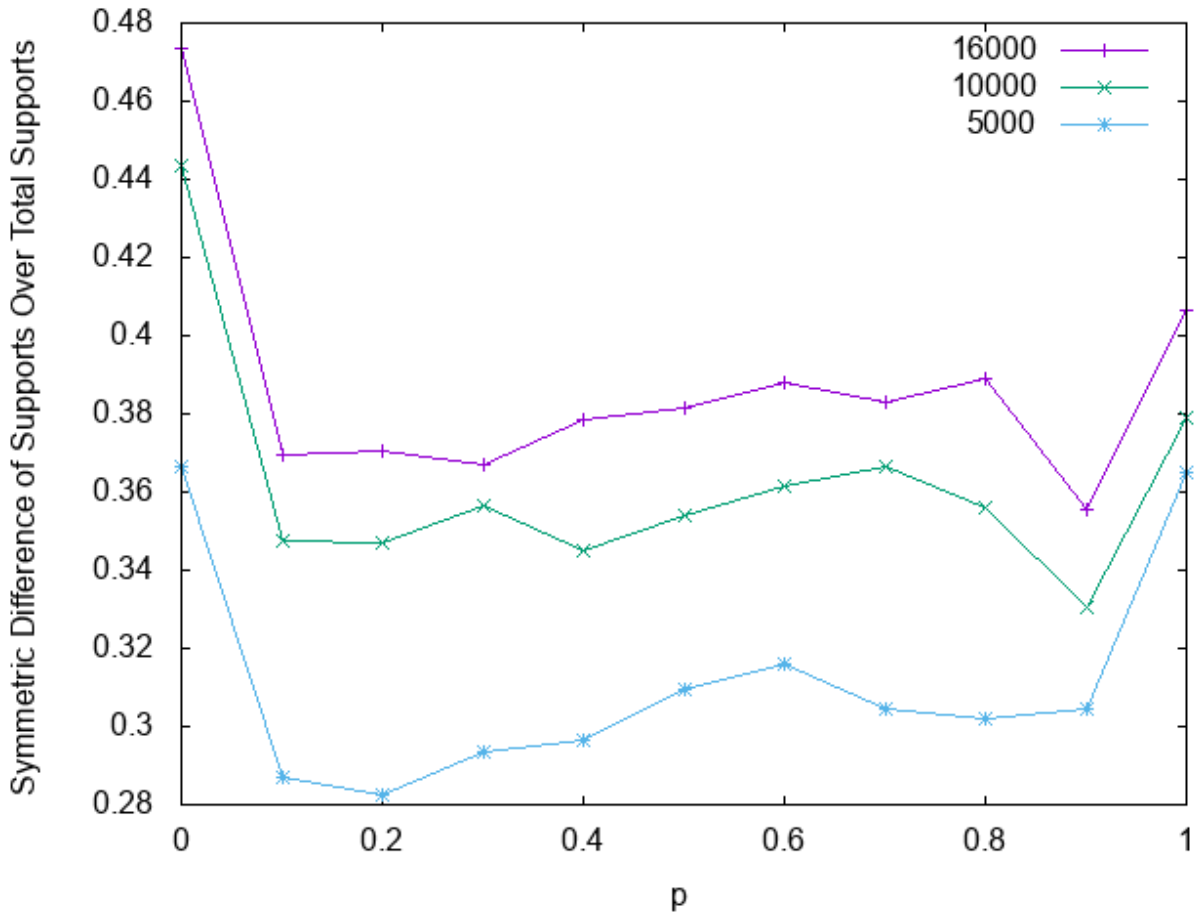

Figure 3.6: Comparison of the overlapping support vectors between the SVMs. The higher the number, the less support vectors the machines share.

Chapter IV: Conclusion

This final chapter revisits the contributions made through this research. We conclude by discussing the opportunity for future work in this area of research.

4.1 Contributions

The research for these experiments began by assessing the previous research conducted on CNN feature extraction. This research revealed the opportunity to expand on the evaluations of the features extracted from CNNs and provide contributions to the field of representation learning.

Initial experiments revealed that training a nonlinear SVM on top of a CNN may provide increased accuracy. This will prove to be useful for improving current applications with CNN vision systems.

The following experiments provided a framework for evaluating CNN features using an incremental transfer learning task. This method allows researchers to use the same dataset as a source and target task without having to quantify the level of semantic similarity between the datasets. By use of this framework, it was revealed that CNN feature extraction is useful for datasets with a low number of training samples. This is consistent with the previous research conducted on CNN transfer learning. However, this task also showed that as the training set sample size increases the models become less stable as opposed to using the original image.

We hope the research and evaluation presented in this thesis sheds light on the strengths and weaknesses of CNNs and paves a way for improving the generality of the features extracted from these systems.

4.2 Future Work

There are several opportunities to extend this research and evaluate other transfer methods, such as fine-tuning. Many of the experiments in this thesis can be used to evaluate other networks and describe their ability to produce generic set of features.

In Figure 3.3, as p is increased from $p = 0$ to $p = 0.1$, the task immediately becomes more difficult; this is because the classification task grows from classifying five labels to ten labels. However, at a low number of training samples, the CNN features prove to be increasingly superior until the distribution of the data contains an equal amount of set A and set B (or $p = 0.5$). At that point, the features' performance declines once again. Future research could pinpoint what exactly is happening when the target task contains a percentage of the source task.

Finally, it would be advantageous to benchmark the effects that different modular components of CNNs have on transferability and generality. These components include different activation functions, optimization routines, architectures, normalization methods, and initialization.

BIBLIOGRAPHY

Abadi, M., et al. (2015), TensorFlow: Large-scale machine learning on heterogeneous systems, software available from tensorflow.org.

Atlas, L. E., T. Homma, and R. J. M. II (1987), An artificial neural network for spatio-temporal bipolar patterns: Application to phoneme classification., in *NIPS*, edited by D. Z. Anderson, pp. 31–40, American Institue of Physics.

Bengio, Y. (2012), Deep learning of representations for unsupervised and transfer learning, in *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, *Proceedings of Machine Learning Research*, vol. 27, edited by I. Guyon, G. Dror, V. Lemaire, G. Taylor, and D. Silver, pp. 17–36, PMLR, Bellevue, Washington, USA.

Cortes, C., and V. Vapnik (1995), Support-vector networks, *Mach. Learn.*, *20*(3), 273–297, doi:10.1023/A:1022627411411.

Deng, J., W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei (2009), ImageNet: A Large-Scale Hierarchical Image Database, in *CVPR09*.

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

Geman, S., E. Bienenstock, and R. Doursat (1992), Neural networks and the bias/variance dilemma, *Neural Comput.*, *4*(1), 1–58, doi:10.1162/neco.1992.4.1.1.

Hinton, G. E., and R. S. Zemel (1994), Autoencoders, minimum description length and helmholtz free energy, in *Advances in Neural Information Processing Systems 6*, edited by J. D. Cowan, G. Tesauro, and J. Alspector, pp. 3–10, Morgan-Kaufmann.

Hubel, D., and T. Wiesel (1962), Receptive fields, binocular interaction, and functional architecture in the cat's visual cortex, *Journal of Physiology*, *160*, 106–154.

Ioffe, S., and C. Szegedy (2015), Batch normalization: Accelerating deep network training by reducing internal covariate shift, *CoRR*, *abs/1502.03167*.

Jarrett, K., K. Kavukcuoglu, M. Ranzato, and Y. LeCun (2009), What is the best multi-stage architecture for object recognition?, in *Proc. International Conference on Computer Vision (ICCV'09)*, IEEE.

Kingma, D. P., and J. Ba (2014), Adam: A method for stochastic optimization, *CoRR*, *abs/1412.6980*.

Kontschieder, P., M. Fiterau, A. Criminisi, and S. R. Bulo' (2015), Deep neural decision forests. [winner of the david marr prize 2015], in *Intl. Conf. on Computer Vision (ICCV), Santiago, Chile*.

LeCun, Y., and C. Cortes (2010), MNIST handwritten digit database.

Lecun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998), Gradient-based learning applied to document recognition, in *Proceedings of the IEEE*, pp. 2278–2324.

Lowe, D. G. (2004), Distinctive image features from scale-invariant keypoints, *Int. J. Comput. Vision*, *60*(2), 91–110, doi:10.1023/B:VISI.0000029664.99615.94.

Marblestone, A. H., G. Wayne, and K. P. Kording (2016), Toward an integration of deep learning and neuroscience, *Frontiers in Computational Neuroscience*, *10*, 94, doi:10.3389/fncom.2016.00094.

Nair, V., and G. E. Hinton (2010), Rectified linear units improve restricted boltzmann machines, in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, edited by J. Frnkranz and T. Joachims, pp. 807–814, Omnipress.

Oquab, M., L. Bottou, I. Laptev, and J. Sivic (2014), Learning and transferring mid-level image representations using convolutional neural networks, in *CVPR*.

Pan, S. J., and Q. Yang (), A survey on transfer learning.

Razavian, A. S., H. Azizpour, J. Sullivan, and S. Carlsson (2014), Cnn features off-the-shelf: An astounding baseline for recognition, in *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, CVPRW '14, pp. 512–519, IEEE Computer Society, Washington, DC, USA, doi:10.1109/CVPRW.2014.131.

Rosenblatt, F. (1958), The perceptron: A probabilistic model for information storage and organization in the brain, *Psychological Review*, *65*(6), 386–408, doi:10.1037/h0042519.

Silver, D. L., I. Guyon, G. Taylor, G. Dror, and V. Lemaire (2011), Icml2011 unsupervised and transfer learning workshop, in *Proceedings of the 2011 International Conference on Unsupervised and Transfer Learning Workshop - Volume 27*, UTLW'11, pp. 1–16, JMLR.org.

Simard, P. Y., D. Steinkraus, and J. C. Platt (2003), Best practices for convolutional neural networks applied to visual document analysis, Institute of Electrical and Electronics Engineers, Inc.

Yosinski, J., J. Clune, Y. Bengio, and H. Lipson (2014), How transferable are features in deep neural networks?, in *Advances in Neural Information Processing Systems 27*, edited by Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, pp. 3320–3328, Curran Associates, Inc.