TRANSFER LEARNING IN CONVOLUTIONAL NEURAL NETWORKS FOR FINE-GRAINED IMAGE CLASSIFICATION

THESIS

Nicholas C. Becherer, 2nd Lieutenant, USAF

AFIT-ENG-MS-17-M-005

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-MS-17-M-005

TRANSFER LEARNING IN CONVOLUTIONAL NEURAL NETWORKS FOR FINE-GRAINED IMAGE CLASSIFICATION

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Computer Engineering

Nicholas C. Becherer, B.S.E.E.

2nd Lieutenant, USAF

March 2017

AFIT-ENG-MS-17-M-005

TRANSFER LEARNING IN CONVOLUTIONAL NEURAL NETWORKS FOR FINE-GRAINED IMAGE CLASSIFICATION

THESIS

Nicholas C. Becherer, B.S.E.E.
2nd Lieutenant, USAF

Committee Membership:

Lt Col J. M. Pecarina, PhD
Chair

Dr. S. L. Nykl
Member

Dr. K. M. Hopkinson
Member

AFIT-ENG-MS-17-M-005

Abstract

In recent years, convolutional neural networks have achieved state of the art performance in a number of computer vision problems such as image classification. Prior research has shown that a transfer learning technique known as parameter fine-tuning wherein a network is pre-trained on different datasets can boost the performance of these networks. However, the topic of identifying the best source dataset and learning strategy for a given target domain is largely unexplored. Thus, this research presents and evaluates various transfer learning methods for fine-grained image classification as well as the effect on ensemble networks. The main contributions are a framework to evaluate the effectiveness of transfer learning, an optimal strategy for parameter fine-tuning, and a thorough demonstration of its effectiveness. The experimental framework and findings will help to train models in reduced time and with improved accuracy for target recognition and automated aerial refueling.

Acknowledgements

First and foremost, I would like to thank my advisor Lieutenant Colonel John Pecarina for providing the guidance and direction I needed to start and finish this document. I would like to thank my teachers for giving me the background that was necessary to even attempt this. Lastly, I would like to thank all my friends and classmates without whom I probably wouldn't have been able to finish this program.

Nicholas C. Becherer

Table of Contents

Page

List of Figures

List of Tables

TRANSFER LEARNING IN CONVOLUTIONAL NEURAL NETWORKS FOR FINE-GRAINED IMAGE CLASSIFICATION

I. Introduction

1.1 Background and Motivation

In recent years, Convolutional Neural Networks (ConvNets) have become increasingly popular for a number of computer vision problems such as image classification [31], image localization, and image segmentation [22]. The research presented is of interest to anyone who is in the computer vision field and autonomy. Target recognition is an important task for computer vision. In particular, fine-grained classification is an important problem for fields such as surveillance. While it is simple to identify something like a tank, identifying a friendly or enemy tank is more difficult. This is an important goal for automating surveillance. Another application would be automated aerial refueling. Before a refueling aircraft could begin doing any sort of modeling of the receiver aircraft, it would need to first correctly identify the receiver aircraft. This makes image classification a critical step. This research also has application to civilian fields such as search and rescue as well as robotics.

In order to be effective, ConvNets require a large amount of labeled data and a massive number of supervised training iterations. One technique that has been shown to improve the performance of a ConvNet is to use a ConvNet trained on a different dataset and use it as the initialization for the current problem. This technique is known as parameter fine-tuning. While it has been shown that this generally leads to better performance than random initialization [52], research in this area is relatively

scarce and is far from complete.

1.2 Problem Statement and Research Goals

Research has shown that parameter fine-tuning is an effective form of transfer learning for ConvNets [22][52]. This suggests that ConvNets learn some set of features from the source dataset that is universal for many or all datasets. This thesis investigates the universality of features, especially when taken from general or fine-grained classifiers for use on fine-grained classification. More specifically, the questions this thesis seeks to answer are:

1. When transferring learned layers, how much learning is necessary in the pre-trained layers?

2. For fine-grained classification, what type of dataset serves as the best source task?

3. Do ensembles of fine-tuned ConvNets outperform ensembles of randomly initialized ConvNets?

This thesis executes an experimental approach to analyze concepts such as transfer learning in ConvNets. In so doing a set of experiments to evaluate different transfer learning strategies, we can answer the above questions and present best practices for transfer learning applied to fine-grained image classification.

1.3 Approach

The methodology is fully presented in chapter 3, but this section provides a brief overview of the methodology. Before the experiment can be fully described, the underlying dataset must be explained. This thesis aggregates seven fine-grained datasets that are freely available. These datasets are then given a major classification. The

major classifications are dog, plane, sign, bird, flower, cat, and vegetable. Within each major classification is a number of minor classifications. For example, dog has 111 minor classes corresponding to different breeds of dogs such as Border Collie and Rottweiler. No major class has the same number of minor classes or images. For example, dog has 71947 images across 111 minor classes while cat has 2392 images across 12 categories.

From this, there are three different types of networks trained. The first two are called superset networks and are trained on data from all or almost all of the major classes. If the superset network does coarse-grained classification at the major class level (e.g. dog vs. plane), it is called a generalist superset network. If the superset network does fine-grained classification (e.g. Border Collie vs. 737), it is called a high-fan out or just high-fan superset network. The last type of network only performs fine-grained classification on a single major class and is called a specialist network.

To answer research question one, an experiment is presented to show how ConvNets require learning in transferred layers. By progressively transferring and freezing layers, we show a decrease in accuracy. A followup experiment shows that by adjusting the learn rate in the transferred layers accuracy increases. Taken together, these experiments help to empirically identify the optimal learn rate for transfer learning.

Next, this thesis addresses question two proposed in the problem statement based on the results of five different initializations of each specialist network. First, the specialist network is trained from random initialization to serve as a control. Next, it is trained using the superset high-fan and generalist networks as the initialization. This allows us to determine whether fine-grained or coarse-grained networks serve as a better source for fine-grained classification. Next, we use the superset high-fan and generalist networks trained on the dataset without the major class that specialist is training on. This source is more distant from the superset networks trained on the full

dataset, allowing us to measure the effect of distance on the target dataset. Lastly, since each of the major classes has a different amount of training data available, we'll be able to measure the effect of training data on the target task because there are seven different target tasks.

Lastly, an ensemble of fine-tuned ConvNets generated by the previous experiment are compared to an ensemble of randomly initialized ConvNets. This allows comparison of transfer learning and ensemble classifiers.

1.4 Assumptions and Limitations

The network architecture used for this thesis is AlexNet [31]. As explained in chapter 2, it is no longer the leading architecture in the field. However, due to computational time restraints, it was used as the model for this experiment. It is assumed that the results would scale up to more advanced architectures.

The assumption that features learned by ConvNets are somewhat universal is based on the observation that parameter fine-tuning has been effective. Since training a ConvNet is really just an optimization problem approximated by gradient descent, it is possible that using a random initialization can achieve the same or better results. However, the fact that parameter fine-tuning has been shown to be effective suggests that the local minima found in a previous problem is closer to the local minima of most problems. This is the basis for the assumption that weights learned in ConvNets are somewhat universal.

Several related questions are beyond the scope of this thesis and are not answered by it. The target task in this research is fine-grained classification. The effect on other target tasks is not examined. Often times, a network trained on the ImageNet dataset is used as the source task [52]. This dataset is a mixture of fine- and coarse-grained data. This sort of network is not examined and would require a different dataset.

A different type of transfer learning uses a ConvNet trained on a dataset (again, usually ImageNet) as a fixed feature extractor for use in algorithms designed to solve other visual problems [22][4][44]. This sort of transfer learning is also beyond the scope of this thesis, although the framework established here could be applied to this technique as well.

1.5 Thesis Organization

This thesis is organized into several chapters as shown. The first chapter serves as an introduction to the thesis as a whole. It explains the purpose of the problem, the scope of this research, and the goals of the research. The overall format is shown in figure 1.

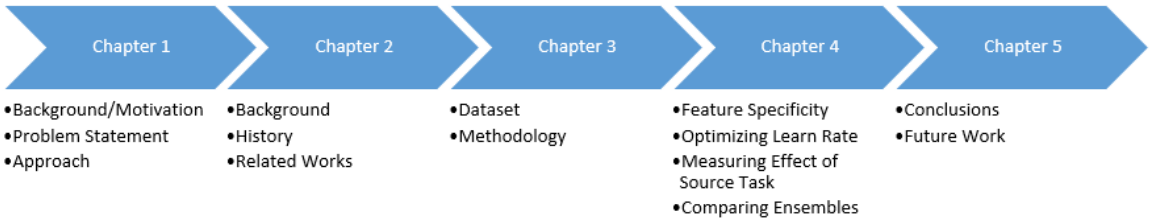

Figure 1. The organization of the thesis.

Chapter 2 begins with an in depth explanation of convolutional neural networks, explaining the algorithms and technical details required to understand them. It will then present a brief history explaining how convolutional neural networks came to be the leading technique for image classification. Then, it explores current state of the art implementations and research in the area of ConvNets. This particular section will especially focus on transfer learning for image classification and other visual problems. This helps us identify gaps in the current understanding that this thesis will in part seek to fill.

Chapter 3 serves to provide a methodology to fill these gaps. Specifically, we will present a method for evaluating the effects of transfer learning on a target dataset.

The goal is to provide insight on whether ConvNets can learn general features that can be applied universally to all datasets or whether a network learns features specific to that dataset. Several networks will be trained with varying degrees of finely-grained data. Then a specialist network will be trained for a particular data subset using the learned weights from the previous networks as initialization. They will also be initialized randomly to serve as a control group. This will give us insight into whether or not the features that a convolutional neural network learns are general to all imagery or specific to the dataset in question.

Chapter 4 then details the implementation of the methodology and the results. It will present a dataset of annotated images for the evaluated networks to classify. These annotations will serve as the ground truth on which to judge the accuracy of the networks. The accuracy between the networks trained on specific subset of data but with a different initialization will be compared. This should give insight into whether or not the features are general or specific to the dataset and therefore whether or not features are general.

Lastly, Chapter 5 will then summarize the findings of this research. It will present the conclusions that may be drawn from the results in chapter 4. It will also offer areas of interest for future research and how they may be addressed.

II. Background and Related Works

2.1 Background

Convolutional Neural Networks are not a new topic, but they have recently risen to prominence after years of relative neglect. ConvNets are a machine learning technique that have proven to be useful in a number of areas from image classification [48] to speech recognition [14] to drug discovery [3]. They can trace their lineage back to Fukushima's Neocognitron from the '80s [20]. With the introduction of training through stochastic gradient descent, they were able to solve a number of problems. However, difficulty in getting them to scale to larger problems led to a loss of interest after the 1990s. In recent years, several researchers have successfully demonstrated their ability to scale and to achieve state of the art results in open computer vision problems [31].

2.1.1 Artificial Neural Networks.

As the name implies, Convolutional Neural Networks are an extension on artificial neural networks. Both are supervised machine learning techniques. They are represented as a directed, acyclic graph and provide a rough approximation of a biological neural network. A basic artificial neural network is shown in figure 2.

The circles represent neurons and the lines connecting them represent weights. A vertical column of neurons is referred to as a layer. The leftmost layer is known as the input layer or layer 0 and the rightmost layer is known as the output layer. All layers in between are referred to as hidden layers. The lines connecting layers are collectively referred to as a weight matrix.

The elements of the input to a layer is referred to as ${}^{l}x_i$ where i is between 1 and the number of inputs and l is the layer. An activation function is usually applied to

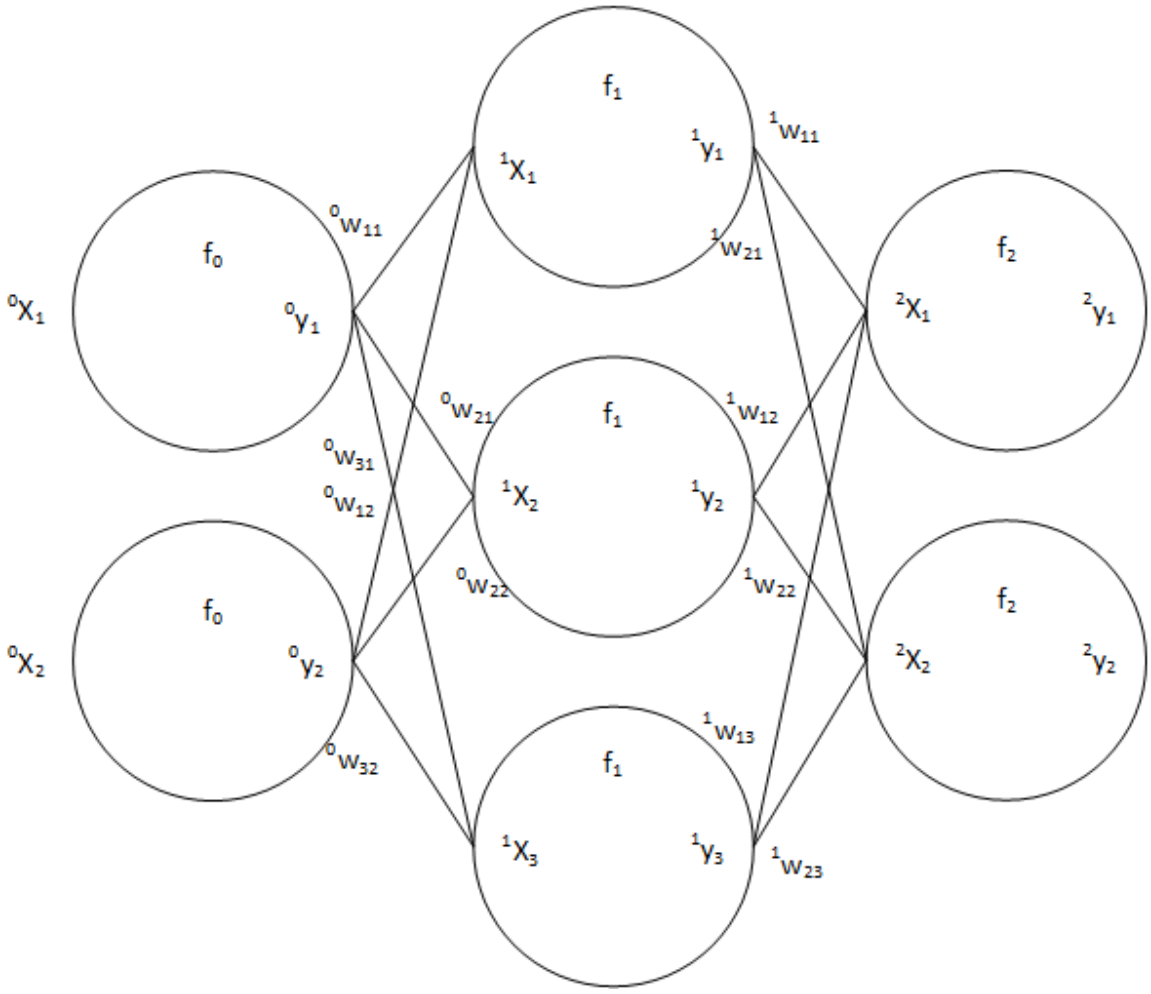

Figure 2. A basic neural network with 2 inputs and outputs and one hidden layer.

the input of the neuron such that ${}^{l}y_i = f_l({}^{l}x_i)$, where f_l is the activation function for that particular layer. The individual weights connecting neurons are are referred to as ${}^{l}w_{ij}$ where l is the previous layer, i is the neuron in the next layer to which the weight connects, and j is the neuron in the previous layer from which the weight connects. So ${}^{1}w_{21}$ is the weight that connects the value from the layer 1 neuron 1 to layer 2 neuron 2. The value of ${}^{l}x_i$, where x is the input of neuron i of layer l, is

$$ {}^{l}x_i = \sum_{j=1}^{J} ({}^{l-1}y_j)({}^{l-1}w_{ij}) \tag{1} $$

Activation functions usually have several important properties. First, they need to be nonlinear. If a linear function is chosen as the activation function, then the neural network can only learn linear functions. Typical choices for activation functions include hyperbolic tangent and the sigmoid function. By applying a linear combination of nonlinear activation functions, any arbitrary function can be approximated[16].

Two special cases should be noted. First, there is usually no activation function applied to the input layer. Second, in the case of classifier networks, the softmax function is often used as the activation on the output layer. This translates a set of real numbers into a probability distribution between 0 and 1 exclusive. Since it is a probability distribution, the outputs sum to one. The softmax function is defined as

$$ \sigma(x_i) = \frac{e^{x_i}}{\sum_{i=1}^{I} e^{x_i}} \tag{2} $$

By using exponentials, the softmax function forces all probabilities to be positive and highlights the maximum value while suppressing values less than the maximum.

2.1.2 Backprop.

Neural networks can approximate a function given the right set of weights. However, it is extremely difficult to know the optimal set of weights a priori or even what the function to approximate is. Neural networks rely on an algorithm known as backpropagation to find these weights. The basic overview of the process begins with a sample input and a known output for that sample. An error function is then used to compare the expected result with the actual result. After the error is calculated, a credit assignment process determines the error caused by each individual weight and adjusts them proportionally. This algorithm is known as backpropagation or backprop. The initial error is determined by the loss function of the network. As we move backwards across the layers, gradient descent is used to calculate the error in earlier layers. Hence the error at the output of the network propogates backwards through the entire network.

The multinomial logistic loss or cross entropy loss function is usually chosen as the error function for classifiers. For each output, the error is referred to as E_i, the expected output is Y_i, and the actual output is y_i.

$$E_i = -Y_i * log(y_i) \tag{3}$$

For each neuron, there is an associated error ${}^{l}\delta_i$. For the output layer ${}^{l}\delta_i$ is the output of the multinomial logistic loss function times the gradient of the activation function, or ${}^{l}\delta_i = f'({}^{l}x_i) * E_i$. For every other layer, it is the same times the weights connected to that neuron. In general the delta for a layer is

$${}^{l-1}\delta_i = (f_l')({}^{l}x_i)({}^{l}\delta_i)\sum_{j=1}^{J} {}^{l-1}w_{ji} \tag{4}$$

The output layer is just a special case where there is only one weight equal to one

and δ is equal to E. The amount to adjusts the weights is then

$$\Delta^l w_{ij} = (\lambda)({}^l\delta_i)({}^l y_j) \tag{5}$$

where λ is the learning rate hyper-parameter. After this has been calculated for every layer and weight in the network, the weights are updated such that

$${}^l w_{ij} =^l w_{ij} - \Delta^l w_{ij} \tag{6}$$

This entire process represents one training iteration of the backprop algorithm [21].

Performing backprop on one example at a time is referred to as online learning. Performing on multiple examples at a time can be done by averaging the gradient from each example. When the entire training set is used during backprop, the process is referred to as batch learning. Anywhere between batch and online learning is referred to as minibatch learning. The backprop algorithm is only guaranteed to converge if batch learning is used [7]. However, due to computational restraints, minibatch is usually used. This process is often performed stochastically by choosing examples in a random order. With an appropriate learning rate, stochastic gradient descent is almost guaranteed to converge to a local minima [10].

2.1.3 Early History of ConvNets.

Aside from neural networks, ConvNets owe some credit to the early work in understanding mammalian vision systems [25][26]. Hubel and Weiss discovered that mammalian visual systems consisted of a light exciting the optical system and transmitting a signal through a series of dense nerves to the primary visual cortex of the brain. Their experiments lead to the discovery that mammalian eyes were especially

sensitive to edges and blobs of color [40]. This biological system served as the inspiration for Fukushima's neocognitron architecture in the early 1980s [20]. These architectures introduced some of the ideas that would later make ConvNets successful, such as invariance to translation. However, Fukushima lacked both an effective method for training these systems and the computational power that would later be needed for a better method.

By 1989, LeCun *et al* presented the first successful convolutional neural network [34]. Initially built as a system to automatically recognize digits, LeCun was able to advance his network to read machine- and handwritten checks by the end of the 1990s [32][36]. LeCun named his architecture LeNet-5, and it demonstrates all of the important concepts present in ConvNets, shown in figure 3.

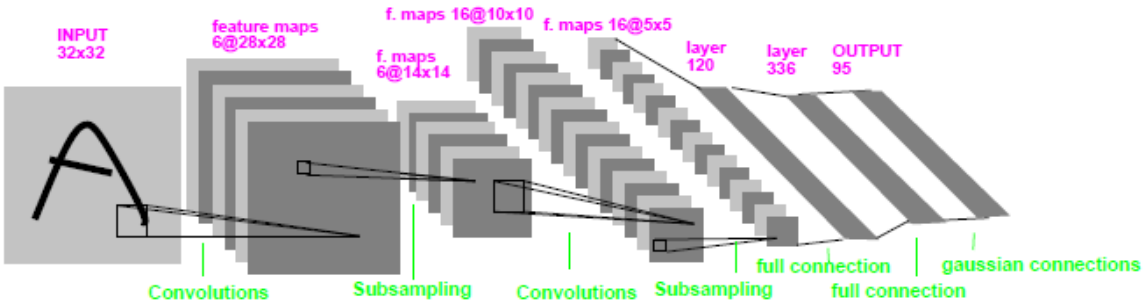

Figure 3. The architecture of LeNet-5 [36]

LeNet-5 accepts a greyscale 32x32 image as the input. The first convolutional layer contains 6 5x5 convolutional kernels which generate 6 28x28 feature maps (unlike most ConvNets, LeNet-5 does not pad inputs to convolutional layers, causing the feature map to have reduced dimensionality compared to the input). Each feature map is downsampled by using a 2x2 average pooling filter with a stride of 2, reducing the dimensionality by three quarters. 16 5x5 convolutions are applied to generate 16 10x10 feature maps. Another 2x2 average pooling filter is applied to reduce the feature maps to 16 5x5 feature maps. These feature maps are then used as the input to a fully-connected neural network. The first layer has 120 hidden neurons. The

second has 336 hidden neurons. The output layer has 95 neurons, corresponding to 95 different characters. Hyperbolic tangent is used as the activation function in the fully connected layers.

2.1.4 Applying Convolution to Computer Vision Problems.

Artificial Neural Networks are ill suited for problems with inexact inputs. For example, speech recognition is difficult for neural networks because the important part of an audio clip may appear anywhere in the clip. Neural networks depend on the same type of input features being passed to the input neurons. To overcome this, some sort of preprocessing needs to be applied to the input to remove this variance [32]. The convolution operation can help overcome this. Although convolution is effective for other problems such as speech recognition, it will be discussed in the context of visual problems because that is the focus of this thesis.

Visual problems tend to be inexact due to changes in position, lighting, size, and orientation. To overcome this, a convolutional front end is added to the network [32]. Convolution is a slide and shift operation that combines two arbitrary functions. If the functions are finite and discrete (like an image), then the operation can be thought of as a window being applied to some portion of the input and pulling some information from that localized window and representing it as a single value. Mathematically, the function is

$$y[m,n] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} h[m-k, n-l]x[k,l] \tag{7}$$

As the window slides along both dimensions of the image, a matrix is created containing feature information about the original image. The second function or matrix is called a convolutional kernel. The output of the convolution is called a feature map. Figure 4 shows a simple example. The 2x2 kernel h slides over the

input x. The sum of the element-wise multiplication is the value generated for that location. The input x is typically padded with some value for the case where the h does not perfectly align. If it were 0-padded, the value of $y[1,1]$ would be $-.5$.

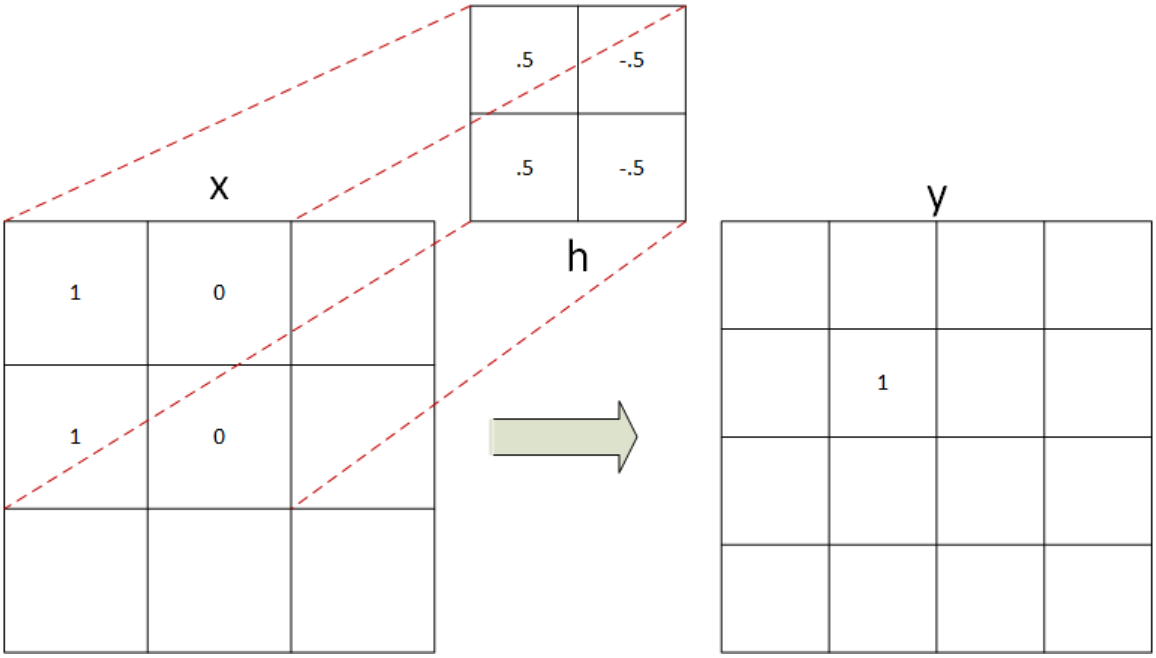

Figure 4. An example of the convolution operator in action. The sum of the products of each pairwise value of the input x and the kernel h generates one value in the feature map. As the kernel slides, it generates the entire feature map y.

In practice, convolution can be implemented as a modified version of a fully connected neural network layer [35][15]. Since the same value is multiplied across several input values, this can be implemented as forcing multiple weights to share a value. Inputs not covered by the current window are connected by weights that are forced to 0. These shared weights are then trainable using the same backprop algorithm. The only difference is that the weight update is averaged across all neurons that are forced to share the same value. This prevents them from diverging from their shared values. The result is an efficient, parallel implementation of convolution where the kernels are learnable.

The practical effect of convolution is to extract patterns or features from the

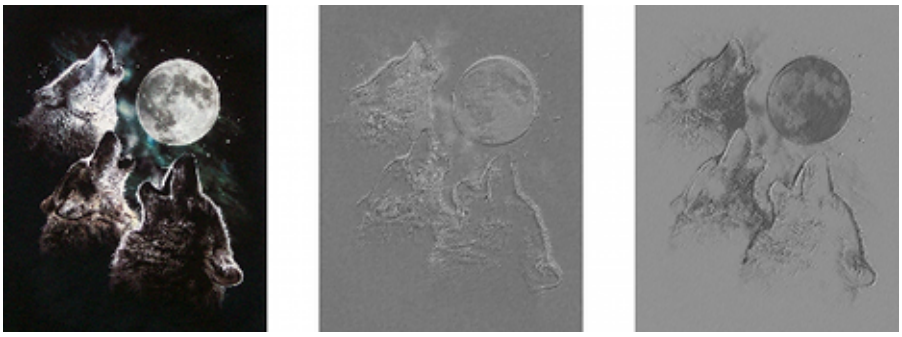

Figure 5. An example of a two random kernels applied to an image. Random kernels tend to approximate edge detectors [1].

image regardless of position in the image. By applying it uniformly across the entire image, a certain pattern will be extracted regardless of its position. It has long been noted that kernels initialized with Gaussian noise tend to approximate Gabor filters or edge detectors [18][23][32]. This suggests there may be some universality of features learned by ConvNets. By having multiple convolutional layers in series, the network can identify low level feaures, combinations of low-level features, and combinations of combinations of low-level features and so on. This allows the network to detect high-level features present in the original image [33]. After several convolutional layers, these features are passed to the fully connected layers of the network and the system is is able to classify them. Rather than classify the image based on the pixels, the network classifies the image based on the patterns present.

Pooling. One downside to the addition of a convolutional layer is the explosion in the dimensionality for the entire network. In the example shown in figure 4, only one feature map is generated. In practice, most networks have dozens or hundreds of kernels per convolutional layer. If a network had k layers with n convolutional kernels each, the resulting convolutional layers would increase the dimensionality by a factor of n^k. Suppose the input of a network was a 228x228 image. The input would have 51,984 features. After applying 3 convolutional layers with 96 kernels each, the

resulting output would be almost 46 billion features. If a 32-bit floating point number was used to represent each feature, then the network would need 171GB to represent the output. Training would require performing backprop on an amount of data that is computationally unfeasible. This is obviously untenable and requires a solution.

It is then necessary to implement some form of downsampling or pooling to reduce the dimensionality. Pooling operations typically have a fixed size window and certain stride. The window applies some function (e.g. average or maximum) to the values in it and reduces them to a single value. The window then slides across the feature map by its stride. The stride is set up such that the pools may overlap, align on the boundary, or not at all. For example, if a 2x2 pooling window with a stride of 2 were applied to the feature map shown in figure 4, the output would be a 2x2 matrix. A 1x1 pool with a stride of 3 would also achieve the same reduction in dimensionality, although it would not necessarily generate the same feature map. In general, overlapping or boundary-aligned strides are desirable because they preserve the locality of information. Non-overlapping strides leads to some information not be considered in the downsampling process. This technique allows many convolutions to be used without having to deal with the resulting explosion of dimensionality.

By the late 1990s, progress and research in neural networks began to stagnate for several reasons [33]. The first was that computational cost greatly increases with input size, which made training larger networks difficult and time consuming. The second was the fear that gradient descent may be insufficient for large scale networks. It was thought that this technique would be prone to getting stuck in local minima. Lastly, larger networks were prone to overfitting and required large amounts of training data to proper train. Prior to the growth of the Internet, this was difficult to acquire [40].

This stagnation continued through the early 2000s. Work on ConvNets continued but rarely scaled beyond smaller problems. Most work focused on smaller datasets

such as the MNIST (handwritten digits), NORB (Toys), or CIFAR10 (10 random objects) [19][37][29]. All of these datasets are either low resolution, greyscale, or both. Instead, focus shifted towards new algorithms such as SIFT and SURF [38][6]. These algorithms are designed to extract features from images. Unlike convolutional layers, however, these features extractors are engineered and do no learning. Importantly, it was shown that ConvNets were highly parallel and benefited from being trained on GPUs; however, GPU processing required writing custom pixel shaders in graphics libraries and was a significant barrier to research [11]. With these barriers, researchers were unable to significantly advance ConvNets until the breakthrough work demonstrated in AlexNet [31].

2.1.5 ImageNet Competitions and AlexNet.

In recent years, ConvNets have come back to the forefront of computer vision after a number of breakthroughs. In particular, the work of Krizhevsky *et al* in the 2012 ImageNet Large Scale Visual Recognition Challenge (ILSVRC) demonstrated the viability of ConvNets for much more difficult problems. ILSVRC presented approximately 1 million images across 1000 classes. Krizhevsky *et al* managed to leverage a number of recent ideas to scale a ConvNet. By utilizing these recent advances, they managed to achieve state of the art results in accuracy, reducing the error rate of the next best entry by almost half [31]. Before their submission, it was thought that ConvNets of such scale would be prone to overfitting and would generalize poorly. Krizhevsky *et al* demonstrated that this was not the case when a few new techniques were utilized.

AlexNet used the Rectified Linear Unit (ReLU) as the activation function instead of the more common hyperbolic tangent or the sigmoid function. The ReLU function is defined as

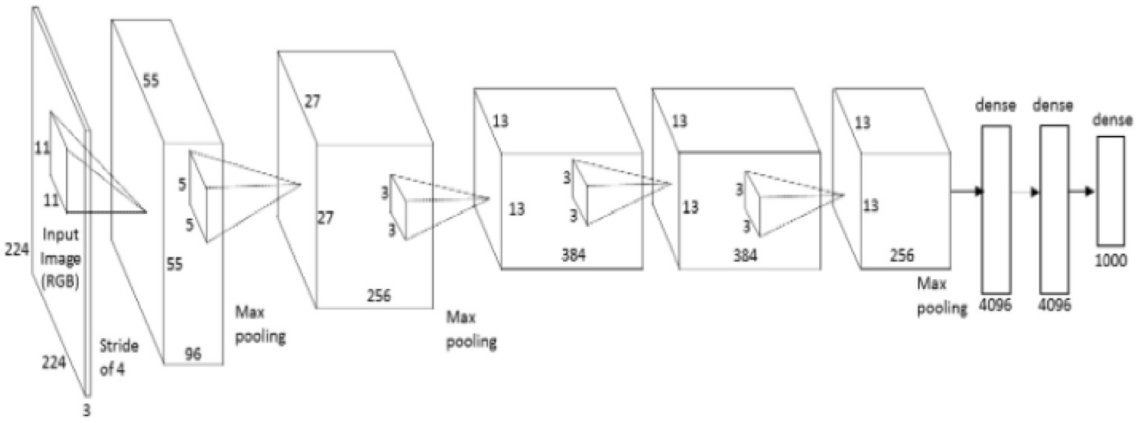

Figure 6. The architecture of AlexNet [31]

$$ReLU(x) = argmax(0, x) \tag{8}$$

ReLUs are non-linear, which means a two layer neural network composed of ReLUs can approximate any arbitrary function. They are also differentiable, which means they are an acceptable function for use in the backprop algorithm. Krizhevsky *et al* chose the ReLU based on prior work showing that it would likely lead to faster convergence over other activation functions [41].

AlexNet also used overlapping pooling boundaries, whereas LeNet-5 had used boundary aligned pools. The authors stated that this reduces error rates and reduces the propensity to overfit. To further reduce overfitting, the authors also introduced a new technique known as dropout regularization [24]. Inspired by the stochastic decision tree technique known as bagging, neurons in the fully connected layer are selected for dropout with a probability p during each training iteration. Connections to selected neurons are then ignored during forward- and back-propagation. At test time, dropout is removed and the entire network is used for forward propagation. This prevents neurons from becoming tied to any given input features and forces the network as a whole to be able to generalize. Additionally, the authors also introduced

several data augmentation techniques to artificially expand their training data. First, an image had a 50% chance of being selected for horizontal mirroring. A dog facing left is still a dog even if it is facing right. Second, random cropping of the images provided many variations of the same image. The input of the network was 228x228 pixels, and the dataset provided images that were 256x256. Rather than scaling down images during training, the image was randomly cropped under the assumption that any particular window was likely to still demonstrate that particular object. During testing, 5 patches and their mirrors were extracted and the average result was used to determine the network's decision. The final data augmentation technique the authors applied was random perturbations proportional to the eigenvalues of the RGB channels of the image. This had the effect of changing the illumination of the image without changing the subject of the image. The combination of these overlapping pooling boundaries, dropout, and data augmentation techniques are credited with reducing overfitting by a "substantial" amount, although dropout is said to have doubled the number of training iterations before convergence [31].

Lastly, the authors also managed to implement their network and train it on a pair of GPUs using Nvidia's CUDA library. Neural networks are highly parallel because the result at any particular neuron and its activation only depeneds on the previous layer having been calculated. This means an entire layer can be calculated in parallel so long as the previous layers have already been calculated. GPUs are well suited to this task because they have a large number of cores and high memory bandwidth. Research has shown that a parallel GPU implementations can be as much as two orders of magnitude quicker to train in terms of wall clock time [47][12][30]. Given that it took Krizhevsky *et al* almost a week to train the network, training would have likely taken months on a CPU-only implementation.

The success of AlexNet has lead to increased interest in ConvNets. Coupled

with the introduction of Nvidia's CUDA library for general purpose programming on GPUs, ConvNets have advanced significantly in the past few years. Research in this area has also been aided by the development of several new libraries designed specifically for deep learning, each designed with their own philosophies and programming languages. Berkeley Caffe was designed to separate architecture specification from implementation [27]. Torch7 sought to implement a MATLAB-like interface with GPU support in order to achieve performance while maintaining familiarity for many academics [13]. Theano is Python-based compiler for mathematical expressions that performs automatic symbolic differentiation with a focus on performance [9]. Google's TensorFlow was designed to scale across distributed and heterogeneous systems [2]. A comparison of the libraries shows that not all implementations have the same performance; however they all benefit significantly from using a GPU [5]. These libraries have enabled researchers to rapidly prototype and experiment with ConvNets and deep learning in general.

Advances to AlexNet. Subsequent ILSCVRC competitions have lead to advancements in the basic architecture of ConvNets. In 2013, Sermanet *et al* demonstrated that ConvNets could be efficiently extended to perform localization (image classification with a bounding box) and detection (localization applied to multiple unique objects in a image) [45]. They dubbed their network OverFeat and released it for use as a generic feature extractor (similar to SIFT or SURF features). Later research has shown that these features are at least as effective as SIFT features when used in a number of computer vision algorithms [44].

Other networks worth noting include VGG (from Oxford's Visual Geometry Group) [46]. VGG was the name of the model developed by Simonyan and Zisserman for the 2014 ILSCVRC competition. Their contribution was to measure the effect of adding more convolutional layers. Whereas AlexNet had 8 weight layers, they tested models

that had between 11 and 19 layers. They also exclusively used 3x3 convolutional kernels. In contrast, AlexNet used different sized kernels for each layer. Simonyan and Zisserman demonstrated that adding depth to the model increased its performance as the 19 layer model achieved the best overall accuracy.

The current leading model is known as GoogLeNet. It is a 22 layer model developed by Szegedy *et al* [48]. One key contribution Szegedy's team made was the use of the "inception module". At each module, a 1x1, 3x3, and 5x5 convolution is applied as well as a 3x3 max pooling filter. These feature maps are then concatenated, preserving local information at a range of resolutions. The other contribution was their use of additional classifiers midway through the network during training. The authors were concerned that the earlier layers of the network would train very slowly due to the vanishing gradient problem [8]. In order to magnify the gradient, they introduced two additional classifiers during training. When the training data reaches this part of the network, it is passed forward to the next layer and the fully-connected classifier portion. The error for that layer is then the sum of the error in the next layer and the error from the classifier portion. During testing, these extra fully-connected classifiers are dropped and only the result in the final softmax layer is considered. By using an ensemble committee of GoogLeNet models, Szegedy *et al* achieved the best result in the 2014 ILSCVRC competition.

2.2 Related Works in Transfer Learning for ConvNets

Transfer learning is the study of using data gained from one problem in machine learning and applying it to another related, yet different, problem. With ConvNets, there are two main ways to apply transfer learning. The first is to remove the softmax layer of a trained network and use the raw output of the previous fully-connected layer as a generic feature vector that describes a particular image. These features are then

used in a number of algorithms for visual problems. The second is to use the weights learned in a network as the initialization for the same network to be trained on a different dataset. This technique is sometimes known as parameter fine-tuning or network surgery. The last weight layer is removed from the network (due to the fact that the source and target datasets likely had a different number of output classes) and reinitialized randomly. The network is then trained on the target dataset with the hope that the weights learned in the earlier layers are still relevant for the target task.

The next section discusses several papers that have made contributions in this field. The first is "Rich feature hierarchies for accurate object detection and semantic segmentation" which introduces the Regions with CNN Features algorithm and discusses both parameter fine-tuning and using ConvNets as a fixed feature extractor [22]. "CNN Features off-the-shelf: an Outstanding Baseline for Recognition" and "From Generic to Specific Deep Representations for Visual Recognition" discuss using ConvNets as a fixed feature extractor and apply it to a number of computer vision problems [44][4]. Finally, "How Transferable are Features in Deep Neural Networks" exclusively discusses parameter fine-tuning and is most relevant to this thesis.

2.2.1 Regions With CNN features.

A number of computer vision algorithms rely on the input of engineered features to perform their tasks. ConvNets are inherently designed to perform image classification and have to be adapated to work in these tasks. One such task is object detection. In this task, multiple objects may be present in the image and the algorithm is responsible for identifying them and providing bounding boxes. Girshick *et al* note that after the demonstration of AlexNet, many academics doubted that it would be possible to extend ConvNets to other tasks such as object detection. Girshick *et al*

set out to test this hypothesis by applying ConvNets to the 2012 Pattern Analysis, Statistical Modeling and Computational Learning Visual Object Classes (PASCAL VOC) benchmark.

The authors proposed a three step system they dubbed Regions with CNN Features (R-CNN): first, regions are proposed for analysis; second, features are generated from each region using a ConvNet without the SoftMax classifier; third, these features are used passed into class-specific Support Vector Machines (SVM) to label the region. The authors note that R-CNN can work with a number of region proposal methods, but they chose selective search to stay in line with previous work. The ConvNet they used was the same architecture as AlexNet; however, they trained it slightly differently. It was initially trained on the ImageNet dataset. After achieving approximately the same results as Krizhevsky *et al*, they then applied parameter fine-tuning to retrain the network on the 20 classes in PASCAL VOC. After fine-tuning the ConvNet, they remove the SoftMax and use it to generate a 4096-length generic feature vector. Lastly, they trained 21 SVM classifiers (20 classes + 1 rejection class) to classify regions based on the ConvNet-generated features. Any overlapping regions of the same class are then combined.

Girschick *et al* compare their results to the then-leading results on the benchmark. The authors note that their algorithm is similar to the one presented by Uijlings *et al*, which uses selective search to identify regions [50]. Uijlings *et al* then choose one of four different algorithms that are all based on SIFT to generate features in these regions. They then use SVMs to classify the region. This algorithm is identical to R-CNN except for its method of generating features. This makes R-CNN an excellent algorithm for comparing ConvNets with engineered features such as SIFT. The metric used for measuring performance on object detection is commonly mean average precision (mAP), which takes into account label and bounding box distance

from ground truth. In order to improve the mAP, the algorithm needs to correctly label objects and provide a bounding box that matches the ground truth to some degree. As the bounding box more closely matches the ground truth, the mAP increases. Girshick *et al* achieve state of the art results of a mAP of 53.7%. This is better than the results of Uijlings *et al* and the previous best mAP achieved of 35.1% and 40.4%, respectively [50].

R-CNN demonstrated that ConvNets could successfully be used for other visual problems outside simple image classification. The authors also claim to show that parameter fine-tuning is an effective method for training large ConvNets on small datasets. They are the first to note the value of fine-tuning for ConvNets. However, they don't rigorously test this claim. They merely report that by using ImageNet as a supervised pretraining set, they were able to successfully fine-tune to the target task of classifying PASCAL VOC images. It is particular noteworthy that they reduced the learning rate during fine-tuning to prevent "clobbering the initialization" [22]. They do not report hard numbers or even state that they compare to any other methods. The authors also fail to provide any reasoning as to why this is important or necessary. The authors implicitly assume that features generated from a PASCAL VOC classifier are more useful than features generated by an ImageNet classifier. Though this seems to be an intuitive assumption given that the task is based on PASCAL VOC classes and not ImageNet classes, they provide no rationale as to why this may be true. This thesis explicitly evaluates the claim about clobbering the initialization and identifying the best source task.

2.2.2 CNN Features off-the-shelf: an Astounding Baseline for Recognition.

In recognition of the limited work done in transferring learning along the lines of Girshick *et al*, Razavian *et al* sought to apply ConvNet features to a number of vision problems [44]. Rather than focus on creating highly optimized algorithms, the goal was to create the simplest algorithm possible to establish a baseline performance. In general, the authors used the OverFeat network to generate features and then simply passed these features to a linear SVM classifier [45]. The results showed that these features were often competitive with the state of the art algorithms in that particular dataset. The authors also showed that in many cases, they were able to surpass state of the art results by augmenting this simple approach. The tasks they looked at were image classification, fine-grained image classification, attribute detection, and instance retrieval.

For image classification, the authors take the output of the first fully-connected layer of OverFeat and use this as a generic 4096-length feature vector. They then train a set of linear SVMs on these vectors. They also train a set of linear SVMs based on the training data and augmented data gathered from cropping and rotating the original training set. Using the 2007 PASCAL VOC image classification benchmark and the MIT-67 scenes benchmark, they test their model. They also apply the same methodology to the fine-grained imagery from the Caltech-UCSD Birds dataset and the Oxford 102 flowers dataset. Both the simple model and the augmented model beat the existing state of the art results for the PASCAL VOC benchmark. For the MIT-67 benchmark, the simple model is competitive while the augmented model beats the state of the art result. This result is repeated on the Oxford flowers dataset. This result is almost repeated on the Caltech-UCSD Birds dataset, but the authors also compare their results to that of a highly optimized algorithm for the dataset that

also used generic ConvNet features which achieves a better accuracy [17].

For attribute detection, the goal is to identify which nonexclusive attributes are present in an image, e.g. whether the subject is male and/or is wearing glasses. The authors use the same ConvNet and linear SVM architecture. The datasets are the Humans in 3D (H3D) and University of Illinois at Urbana-Champaign 64 attribute (UIUC64). For both datasets, the simple model achieved better than state of the art results while the augmented model did even better.

Lastly, the authors applied instance retrieval. The objective is to take a given image and find a set of similar images from a labeled set. To determine the similarity, the authors use the Euclidean distance between the image in question and the known images. They also augment their data by applying principal component analysis (PCA) to the features. This technique is applied to 5 different datasets. They find that at worst, their method achieves comparable results to the state of the art methods. At best, they slightly outperform it.

The major conclusion Razavian *et al* draw is that ConvNets are a powerful tool that can be applied to many visual tools. For many tasks, a simple SVM classifier is sufficient to match the performance of current leading algorithms. This suggests that further specialization in each domain would likely lead to even better results.

There are three ways to extend the work done. The first is to identify additional domains to which ConvNets could be applied. The second is to dive deeper into individual domains and study ways to further specialize ConvNets for that particular domain. Lastly, the authors can study the effect of the different source networks applied to each task. The last extension is examined in this thesis.

2.2.3 From Generic to Specific Deep Representations for Visual Recognition.

The researchers of the previous paper continued their work in a later paper on the same subject. This time they systematically identify the factors that can affect the transferability and then measure them against the tasks listed above. The factors they identify are early stopping, network depth, network width, fine-tuning, dimension reduction, and number of layers as shown in figure 7. They then exhaustively test these factors against the same benchmarks from [44].

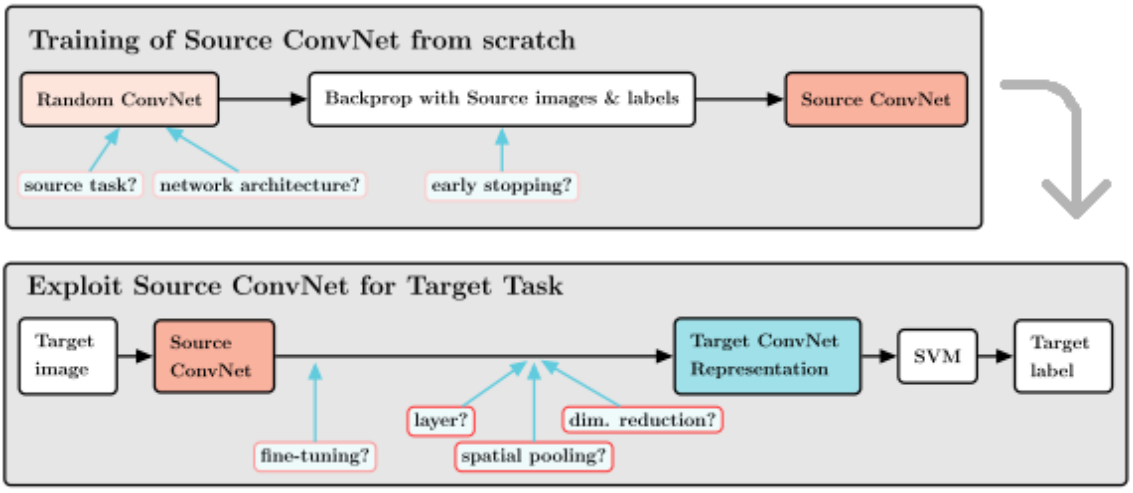

Figure 7. Factors that must be considered for transfer learning according to [4]

Network width refers to the number of neurons in a given layer. They use number of free parameters as a proxy for network width, since different layers in ConvNets have different numbers of neurons. Their results show networks smaller than AlexNet (60 million parameters) transfer poorly. In general, OverFeat (150 million parameters) transfers roughly as well as AlexNet. The main exception is in the case of image retrieval, where OverFeat performs drastically worse on all benchmarks compared to AlexNet.

Network depth refers to the number of layers in the network. In line with the results demonstrated by VGG and GoogLeNet, the authors suspect that deeper net-

works are better able to extract information from the original image. They vary the number of convolutional layers from 6 to 14 and test the results. The conclusion is that as this number is increased, the performance on nearly all the datasets increases" [4]. The only exception is one image retrieval task, which suffers from a minor reduction in performance when depth is increased.

Early stopping is a method used to prevent a model from overfitting. Generally, a validation set is used to test the performance. If performance against the validation set begins to fall, the training process is stopped under the assumption that the model is just overfitting. Contrary to what one might expect, early stopping is not beneficial in any case. Every benchmark shows improvement with more training, though this effect seems to stop after 200,000 training iterations.

The authors also examine the effect of the source task on the target. They do this by also training a variant against the MIT Places dataset as well as a variant trained on both datasets. The results show that the best result is generally comes from the fourth model which concatenates the output of the two individual networks. The only exception is fine-grained recognition does better when using only ImageNet as the source. They suggest that this is because ImageNet has more labels which makes the network more diverse. MIT Places has fewer classes and they are more visually similar than some of the varieties found in ImageNet.

Lastly, they performed an experiment that extended on the R-CNN paper by Girshick *et al* [22]. They found that adding additional images to the existing ImageNet training set managed to provide modest gains over the results achieved by Girshick *et al*. They conclude that even the million images in ImageNet aren't enough to saturate the learning capability of a ConvNet.

The work done is fairly exhaustive in most of the cases studied by the authors. However, it still stands to be extended in a few ways. Although they did vary net-

work architecture parameters, they did not test the leading architecture GoogLeNet [48]. Training GoogLeNet is notably different because it is exceptionally deep and calculates the error at multiple points rather than just the end of the network. Secondly, the research done on source task could be expanded. The authors test two different sources. ImageNet is a hodgepodge of classes, mixing fine-grained (e.g. over 100 breeds of dogs) and general classes (e.g. warplane is a single class). MIT Places is all scenery data that has some degree of clutter depending on the place. Testing source datasets is much harder to do exhaustively because there are so many different datasets and factors that go into creating a dataset. This thesis provides research into this area of research.

2.2.4 How Transferable are Features in Deep Neural Networks?

In parallel with the work done by Azizpour *et al* is research done by Yosinski *et al*. Noting that the filters learned in the first layer of different ConvNets always tend to approximate Gabor filters and color blobs, Yosinski *et al* sought to understand the transferability of layers as network depth increased [52]. They hypothesized that earlier layers were more general and as depth increased later layers were more attuned to specific features particular to the dataset. To measure this, they divided the 1000 classes of ImageNet data into two smaller datasets of 500 classes each. They trained on one dataset and transferred to the other to measure the effect. They measured two variables: the number of layers transferred and the effect of freezing learning in the transferred layers.

The results reveal two interesting behaviors. The first is that they demonstrate the effectiveness of transfer learning in ConvNets and show that transferring more layers is more effective than fewer. However, even transferring one layer shows some improvement. They also demonstrated the importance of continuing learning even

after transferring. The results show that freezing learning in transferred layers consistently causes performance to decline. Most notably, the authors find that freezing learning in earlier layers and re-initializing the later layers causes a tremendous drop in performance (the case of transferring from a source task to a target task that is the same). The authors dub this effect "fragile co-adaptation". Backprop assumes simultaneous changes to the entire network during updates. When this assumption, is invalid, backprop is unable to recover the features that made it work in the first place. This was an unexpected result.

Yosinski *et al* also devised an experiment to measure the effect of transfer learning to a more distant task. ImageNet contains over 100 breeds of dogs, and the first experiment split them roughly equally into the two datasets. In order to create a split with more "distance", the authors split the ImageNet dataset into two datasets based on whether the class was man-made or natural. They state that more distance decreases the less beneficial transfer learning is.

The biggest contribution from Yosinski *et al* was showing that transfer learning is beneficial in general. It was previously assumed that it would only be beneficial in the case of small target datasets, which ConvNets are prone to overfit. The data strongly suggests that random initialization is a poor choice when transfer learning is available.

The experiment measuring the effect of distance on transferability leave something to be desired. ConvNets work based on visual information. A more distant task should be more visually dissimilar. However, the authors present two datasets that are categorically dissimilar, but not necessarily visually dissimilar. Consider the case of lemon, tennis ball, and microwave, all of which are labels in the ImageNet dataset. Visually, tennis ball is closer to lemon than microwave because of similar shape and colors. However, Yosinski's method considers microwaves to be more closer. At

best, it provides a rough approximation for distance. We improve on this work by providing a dataset that groups imagery based on visual similarity rather than an abstract categorization.

There exists a good deal of literature demonstrating the effectiveness of parameter fine-tuning. However, there are still gaps in our understanding. One issue is that the ImageNet dataset, which mixes both fine- and coarse-grained categories together is usually used as the source dataset. Very little research has been done using anything else as the source. This thesis seeks to gain insight into three questions regarding parameter fine-tuning and fine-grained classification. How many layers should be transferred, and how much learning should occur in the transferred layers? What type of source task transfers the best? And finally, what effect does transfer learning have on ensembles of classifiers?

III. Methodology

Chapter 1 of this thesis presented three research questions. Based on the literature review, we expand some of the questions here. For reference, they are:

1. When transferring layers...
   (a) How many layers should be transferred?
   (b) What is the optimal learning strategy in transferred layers?
2. For fine-grained classification...
   (a) Is learning a fine-grained dataset a better source task or a coarse-grained dataset a better source task?
   (b) Is learning a more distant dataset or a more visually similar dataset a better source task?
   (c) Does a larger training dataset reduce the effectiveness of parameter fine-tuning?
3. Do ensembles of fine-tuned ConvNets outperform ensembles of randomly initialized ConvNets?

With these goals in mind, a dataset must be specified that has certain characteristics in order to answer the above questions.

Dataset requirements. First, fine-grained classification must be defined, especially with regards to coarse-grained classification. Fine-grained images are images that are of the same class, but have some degree of differentiation. All images in a fine-grained dataset should be visually similar. Ultimately, the separation of classes within the fine-grained dataset are arbitrary. Consider the major class of

airplanes. This could be divided into fine-grained classifications in multiple ways. One such division could be the name of plane, such as Boeing 737 vs. F-16. Another possible division could be based on a characteristic of the plane, such as biplane vs. monoplane. The important characteristic of fine-grained classification is that the dataset is visually similar. The separates it from coarse-grained classification, which compares two very different objects such as plane vs. flower.

Since the target task under consideration is fine-grained image classification, a collection of labeled finely-grained images is necessary. In order to verify the effect under study, multiple finely-grained datasets should be used to ensure that the effect does not happen to be particular to a certain dataset. By aggregating different sets, it is also possible to create a dataset of coarse-grained dataset. Every image is given a major label and a minor label. The major label is the dataset to which it belongs; the minor label is the fine-grained class in that particular class. For example, an image could have a major label of plane and a minor label of F-16. The coarse-grained dataset is images labeled by their major label. A fine-grained dataset is all images that share a common major class but are labeled by their minor class. This allows the dataset to address the question posed by 2(a).

By aggregating multiple fine-grained datasets, it is also possible to address the question of distance. By training on different subsets of the aggregated dataset, a measure of distance can be obtained. When the target task is fine-grained classification of planes, for example, a network trained on planes and other objects is a closer task than a network trained only on the other objects. This addresses question 2(b).

Lastly, there is some question on how the size of the available training data affects the usefulness of parameter fine-tuning. There is speculation that large datasets do not benefit from parameter fine-tuning because they have enough training data to successfully learn a useful representation. If the aggregated datasets have a nonuni-

form amount of training data, it is possible to measure the usefulness of parameter fine-tuning on larger datasets vs. smaller datasets. This is necessary for addressing question 2(c).

The requirements for answering questions 1 and 3 are much less stringent. So if a dataset meets the above requirements, it necessarily is suitable for answering all of the questions posed earlier.

3.1 Fine-Grained Imagery Datasets

Overview. Data was gathered from a number of fine-grained datasets available online. Table 1 shows an overview of the data gathered. There are 7 major classes. They are vegetable, cat, flower, bird, sign, dog, and plane. The data is not uniformly distributed. Every major class has a different number of images and minor classes, and the imagery is not uniformly distributed among the minor classes. The dataset is further broken down into mutually exclusive training and testing sets. The split is 75%/25% training/testing. Training images are used to fit the models to. Testing images are used to measure the Each major class if further divided into a number of finely-grained minor classes. There are between 12 and 111 minor classes in each major class and between 2392 images and 71947 images in each major class.

Table 1. A high-level overview of the dataset

Major Class	Images	Train	Test	Categories	Mean	Median
Vegetable	17562	13243	4409	24	735.5	800
Cat	2392	1795	597	12	199.3	200
Flower	8189	6180	2009	102	80.3	66
Bird	11788	8872	2916	200	58.9	60
Sign	5886	4424	1462	24	245.3	189
Dog	71947	53982	17965	111	648.2	800
Plane	17800	13350	4450	78	228.2	100
Total	135654	101846	33808	551	246.2	91

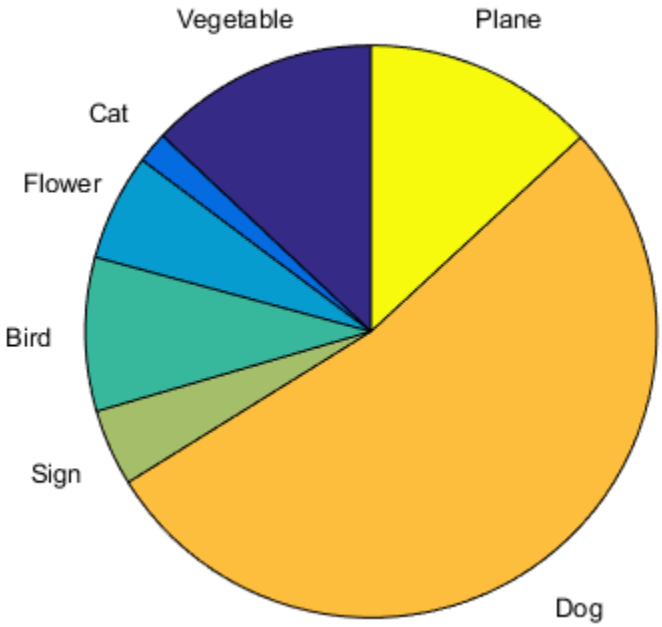

Figure 8. The distribution of the data presented by major class.

Figure 9. A sample of the signs used in the dataset. Plain English labels were not provided with the dataset. From left to right: sign 021, sign 038, and sign 061.

Table 2. A comprehensive list of all the minor classes in the sign class.

Minor	Images	Train	Test
sign_00001	137	103	34
sign_00007	247	186	61
sign_00013	129	97	32
sign_00017	262	197	65
sign_00018	200	150	50
sign_00019	394	296	98
sign_00021	88	66	22
sign_00022	436	327	109
sign_00028	176	132	44
sign_00031	149	112	37
sign_00032	738	554	184
sign_00035	214	161	53
sign_00037	129	97	32
sign_00038	495	372	123
sign_00039	295	222	73
sign_00040	290	218	72
sign_00041	159	120	39
sign_00045	158	119	39
sign_00047	178	134	44
sign_00053	223	168	55
sign_00054	166	125	41
sign_00056	128	96	32
sign_00057	119	90	29
sign_00061	376	282	94

Signs. Table 2 shows a breakdown of all the minor classes within the major class of sign, and figure 9 shows a sample of some of the images. There are 5886 images spread across 24 minor classes with a mean of 245.25 images in each class and a median of 189 images in each class. The data was taken from the KUL Belgium Traffic Sign Classification benchmark [49].

The KUL Belgium Traffic Sign Classification is a collection of images of signs taken around Belgium. Each image is labeled with the type of sign and the specific location. The location data was stripped out and all signs of the same type were combined. The images were provided only with a numerical label for the type rather than an English description. Only signs with more than 80 images were kept. In the end, this lead to 24 minor classes.

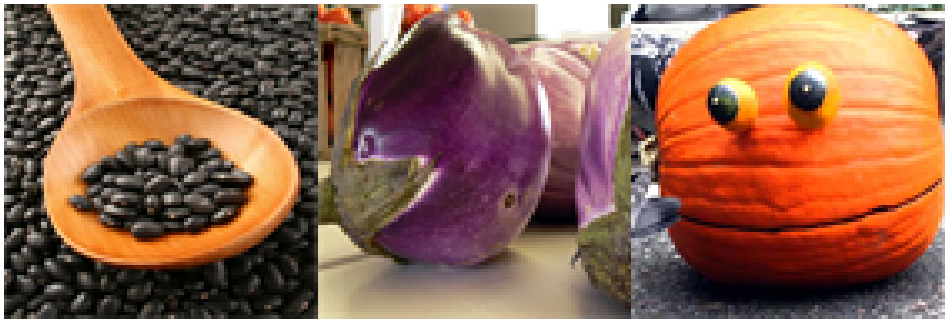

Figure 10. A sample of the vegetables used in the dataset. From left to right: black beans, eggplant, and pumpkin.

Vegetables. Table 3 shows a breakdown of all the minor classes within the major class of vegetable, and figure 10 shows a sample of some of the images. There are 24 minor classes with a mean of 735.5 images in each class and a median of 800 images in each class. The data was gathered from the ImageNet database [49]. The ImageNet database uses automated methods for gathering imagery across the internet in hundreds of thousands of categories. These images are then screened to participants through Amazon Mechanical Turk to eliminate false positives. The

Table 3. A comprehensive list of the images in the vegetable class.

Minor	Images	Train	Test
cayenne	730	548	182
cucumber	800	600	200
tomato	800	600	200
radish	800	600	200
carrot	800	600	200
fava	800	600	200
shallot	686	515	171
broccoli	800	600	200
pumpkin	800	600	200
black	515	387	128
asparagus	800	600	200
brussel_sprouts	800	600	200
artichoke	771	579	192
cauliflower	800	600	200
spinach	748	561	187
bell	800	600	200
pinto	223	168	55
snow_pea	800	600	200
kidney	726	545	181
mushroom	800	600	200
okra	800	600	200
leek	536	402	134
plantain	717	538	179
eggplant	800	600	200

URLs of the remaining images are made available for academic use. This served as the source of images for the vegetable class.

Figure 11. A sample of the dogs used in the dataset. From left to right: Beagle, Dingo, and Border Collie.

Dogs. Table 4 shows a breakdown of all the minor classes within the major class of dog, and figure 11 shows a sample of some of the images. There are 111 minor classes with a mean of 648.2 images in each class and a median of 800 images in each class. It is the largest major class with a total of 71947 images and has the most images per class.

Part of the dataset originated with the Stanford Dogs dataset for fine-grained visual classification [28]. Originally, 20,000 images were presented with this dataset. The dataset was then added to the ImageNet database and combined with images using ImageNet's standard methodology. ImageNet was the ultimate source for this research.

Cats. Table 5 shows a breakdown of all the minor classes within the major class of cat, and figure 12 shows a sample of some of the images. There are 12 minor classes with a mean of 199.3 and a median of 200 images per class. It is the smallest major class with a total of 2392 images.

The dataset was provided as part of the Oxford Pets dataset from the Oxford Visual Geometry Group [43]. The dataset provides a fine-grained set of cats and dogs

Table 4. A subset of the minor classes in the dog class. See Appendix A for the full table.

Minor	Images	Train	Test
Airedale	745	559	186
toy_poodle	800	600	200
border_terrier	800	600	200
tibetan_terrier	460	345	115
soft-coated_terrier	800	600	200
curly-coated_retriever	238	179	59
australian_terrier	650	488	162
griffon	122	92	30
bullterrier	800	600	200
standard_poodle	800	600	200

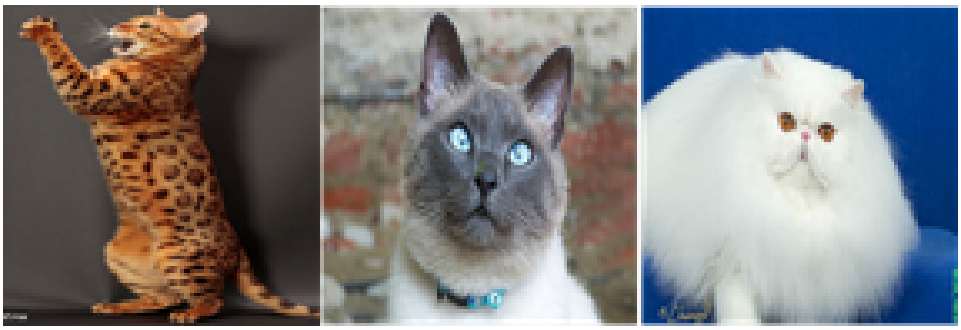

Figure 12. A sample of the cats used in the dataset. From left to right: Bengal, Siamese, and Persian.

Table 5. A comprehensive list of the minor classes in the cat class.

Minor	Images	Train	Test
Siamese	200	150	50
Egyptian_Mau	194	146	48
Birman	200	150	50
Russian_Blue	200	150	50
Ragdoll	200	150	50
Bombay	200	150	50
Sphynx	200	150	50
British_Shorthair	200	150	50
Maine_Coon	200	150	50
Persian	200	150	50
Abyssinian	198	149	49
Bengal	200	150	50

gather from a social network for pet owners and Flickr image groups. The images were then checked manually by human experts.

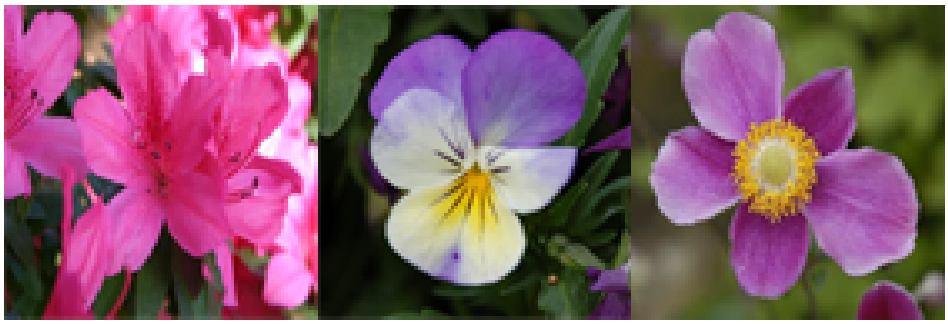

Figure 13. A sample of the flowers used in the dataset. From left to right: Azalea, Wild Pansy, and Japanese Anemone.

Flowers. Table 6 shows a breakdown of all the minor classes within the major class of flower, and figure 13 shows a sample of some of the images. There are 102 minor classes with a mean of 80.3 and a median of 66 images per class. In total, the major class has 8189 images.

The data was provided by the Visual Geometry Group as well. It is known as the Oxford Flowers dataset [42]. The dataset is particularly difficult because multiple characteristics must be taken into account to fully identify the flower. For example, some flowers have identical shapes but different colors, while others have identical colors but identical shapes.

Birds. Table 7 shows a breakdown of all the minor classes within the major class of flower, and figure 14 shows a sample of some of the images. There are 200 minor classes with a mean of 58.9 and a median of 60 images per class. There are a total of 11788 images

The dataset was generated collaboratively between Cal-Tech and the University of California San Diego and is known as the Caltech-UCSD Birds-200 or CUBS-200

Table 6. A subset of the minor classes in the flower class. See Appendix A for the full table.

Minor	Images	Train	Test
bearded_iris	54	41	13
hard-leaved_pocket_orchid	60	45	15
purple_coneflower	85	64	21
bee_balm	66	50	16
foxglove	162	122	40
poinsettia	93	70	23
carnation	52	39	13
japanese_anemone	55	42	13
moon_orchid	40	30	10
alpine_sea_holly	43	33	10

Figure 14. A sample of the birds used in the dataset. From left to right: Cardinal, American crow, and Carolina Wren.

Table 7. A subset of the minor classes in the bird class. See Appendix A for the full table.

Minor	Images	Train	Test
Lazuli_Bunting	58	44	14
Green_tailed_Towhee	60	45	15
Heermann_Gull	60	45	15
Ovenbird	60	45	15
Yellow_headed_Blackbird	56	42	14
Crested_Auklet	44	33	11
Horned_Lark	60	45	15
Groove_billed_Ani	60	45	15
Red_breasted_Merganser	60	45	15
Barn_Swallow	60	45	15

dataset [51]. It is one of the more difficult datasets because there is less training data per class.

Figure 15. A sample of the planes used in the dataset. From left to right: A-10, Spitfire, and Boeing 737.

Planes. Table 8 shows a breakdown of all the minor classes within the major class of flower, and figure 15 shows a sample of some of the images. There are 78 minor classes with a mean of 228.2 and a median of 100 images per class. There are a total of 17800 images.

The dataset was aggregated from two other datasets. The first is another fine-grained dataset from the Oxford Visual Geometry Group. It is known as the Fine-grained Visual Classification of Aircraft dataset (FGVC-Aircraft) [39]. A number of images are provided and labeled with the manufacturer, family, variant, and model. For example, the Boeing 737-200J has a manufacturer of Boeing, a family of 737, a variant of 200 and model of J. Different models of the same variant are often visually indistinguishable. For the purpose of this research, all variants of a family are lumped together at the family level in order to prevent the creation of extremely small classes. It was supplemented with data from a non-public dataset of 10 Air Force aircraft created by Robert Mash [40]. Some minor classes of aircraft have data from both Mash and FGVC-Aircraft.

Table 8. A subset of the minor classes in the plane class. See Appendix A for the full table.

Minor	Images	Train	Test
CV2	800	600	200
DC-6	100	75	25
C130	800	600	200
CRJ-200	100	75	25
F22	800	600	200
A300	100	75	25
BAE-146	200	150	50
Boeing_757	200	150	50
Boeing_717	100	75	25
CRJ-700	200	150	50

3.2 Experimental Design

Before the setup of the experiment is described, some terminology must be defined. There are three types of ConvNets defined based on the training data used and the output. The first is a generalist network. It classifies images based on their major class. The second is referred to as a high-fan network. It performs fine-grained classification on all of the datasets. Since it necessarily has more outputs than the output of the generalist network, it has a much higher fan-out; hence the name "high-fan". Since generalist and high-fan networks train on data from multiple fine-grained datasets, they are also called superset networks. Lastly, a specialist network is a ConvNet that only trains on one major class and outputs the minor classes for that particular major class.

For the sake of an example, consider the variations with respect to the planes dataset. If the dataset is the entirety of the data and is classified at the major class level (7 labels), it is referred to as the generalist network. If the dataset is the entirety of the data and is classified at the minor class level (551 labels), it is referred to as the high-fan network. If the dataset is the entirety of the data except for the planes subset and is classified at the major level (6 labels), it is referred to as the generalist

without planes network. If the dataset is the entirety of the data except for the planes subset and is classified at the minor level (473 labels), it is referred to as the high-fan without planes network. Lastly, there is a planes specialist network which only uses the planes dataset and classifies it at the minor level (78 labels). The planes specialist network can have one of 5 initializations for its weights. The first is random Gaussian noise and is referred to as a "scratch" initialization. The next four come from the aforementioned generalist and high-fan networks and are referred to by the name of the network it comes from. For example, a plane specialist network may be referred to as the "high-fan without planes" specialist network, meaning it was initialized from the weights of the high-fan network that was trained on the full dataset minus the planes subset.

With the goals of this research in mind, the dataset described, and the terminology used throughout defined, the design of the experiment may be explained. In total, four experiments regarding parameter fine-tuning are performed. The first attempts to recreate some of the work done by Yosinski *et al* and demonstrate the specificity of features in learned layers [52]. Next, we examine the effect of varying the learning rate in the learned layers between 0 and 1. Third, we measure the transferability of different sources of data to each individual fine-grained dataset. Lastly, we compare an ensemble of fine-tuned networks to an ensemble of randomly initialized networks.

3.2.1 Constant Parameters.

There are many parameters that can be adjusted when it comes to training ConvNets. Unless otherwise noted, the parameters described below are used across all experiments.

The network architecture to be used for this experiment is the same one as in AlexNet. While other architectures have been used to achieve better results on the

ILVSRC challenge (most notably GoogLeNet [48]), AlexNet is used for several reasons. First, it is a well known architecture and has become a standard architecture for experiments on ConvNets themselves [52]. Second, even with high-powered GPUs, training a ConvNet takes days, and more complicated models take longer (the authors of VGG reported that training took 2-3 weeks on similar hardware [46]). Given the number of ConvNets that need to be trained for this experiment, time is a nontrivial factor.

AlexNet takes a 227x227 RGB image as its input. The first layer contains 96 11x11 convolutional kernels with a stride of 4. The output is then passed through a ReLU activation. Max pooling with a 3x3 window and a stride of 2 is then used to downsample. The next convolutional layer uses 256 5x5 kernels with a stride of 1. This is also followed by a ReLU activation and another identical pooling layer. The third convolutional layer uses 384 3x3 kernels. It is followed by another ReLU but no pooling. The fourth convolutional layer also contains 384 3x3 kernels and is followed by a ReLU activation and no pooling. The final convolutional layer contains 256 3x3 kernels. It is followed by a ReLU activation and 3x3 max pooling with a stride of 2. The output is then fed into the fully-connected artificial neural network. The first hidden layer has 4096 neurons and also uses the ReLU activation. The second hidden layer is identical. Lastly, the output layer has an output equal to the number of labels for the particular dataset in question (1000 in the case of the original AlexNet [31]). A softmax function is applied to this function in order to generate the probability distribution of all possible labels. During random initialization, weights in the fully connected and convolutional layers are sampled from a Gaussian distribution with a mean of 0 and a standard deviation of .05. Dropout with a probability of .5 is applied to the fully-connected layers.

There still remains a number of network hyperparameters to be defined. Many of

these hyperparameters were chosen by experimenting until the network began to train effectively. Large minibatches can lead to slow convergence while smaller minibatches lead to suboptimal convergence. Minibatch size was set to 256. The learn rate was initially set to .001. It was reduced by half every 25,000 training iterations. This learning rate schedule was used for every network and is global unless otherwise noted. Every network is trained for 200,000 iterations. Most networks converged to an optimal solution in less than 100,000 training iterations and then began to overfit; however, for the sake of consistency and thoroughness, all networks were trained for 200,000 iterations. Training consisted of 100 iterations followed by a complete test against the entire validation set. The overall accuracy during each test was saved in order to find the "best" model. When a network's accuracy is reported, it is the model that had the best accuracy on the validation set. However, the best model was never used for transferring. The model saved at 200,000 iterations was always the source model. This is in line with the research done by Azizpour *et al*, which found that early stopping was less beneficial than overfitting [4]

All experiments were done using Berkeley Caffe. Caffe was chosen for several reasons. First, it supports GPU-accelerated training which makes research on this scale feasible. It also makes it simple to transfer learned weights. Lastly, it is very popular and well-supported by the community. It is expected that the same results would be achieved regardless of deep learning framework used. The main difference is interface and underlying implementation, not algorithm.

A sample of the code is used shown in appendix B. The code shown is not exhaustive and is specific to our system due to its use of absolute file paths. Every network trained requires a number of minor changes to the basic template provided. However, anyone familiar with Caffe and Python should be able to recreate our work on a dataset of their choosing with little difficulty.

3.2.2 Determining Specificity of Features.

The goal is to determine how transferable the knowledge of a ConvNet is. This experiment recreates some of the work done by Yosinski to validate his results and our approach. To do this, we transfer the layers one by one to a new task but do not allow learning in the transferred layers. This allows us to isolate the previous knowledge and analyze how effective it is at a different task.

The generalist superset network is used as the source task. Then a variable number of layers between 1 and 7 from the high-fan network are to a new network. The layers not transferred are initialized randomly. We effectively halting learning in transferred layers by setting the learning rate to 0 in these layers only. The remaining weight layers are trained as usual. To serve as a baseline comparison, we also include a scratch network with no transferred layers. According to Yosinski *et al*, transferring the first and second layers has little effect compared to an entirely random initialization. However, as the number of layers transferred increases, the accuracy begins to drop due to the specificity of features learned on the earlier dataset. These features are only useful on the target dataset and the inability to adapt them due to the frozen learning rate has a negative effect on the target task. The target task in this case is the plane subset. Due to time constraints, it was not possible to test this exhaustively across all subsets.

3.2.3 Optimizing Learning Rate.

Having learned the effect of the weights with no adjustment, we seek to study the effect of the rate of learning in transferred layers. It has been suggested that the learning rate in transferred layers should be reduced in order to not "clobber the learning" done previously [22]. However, this has never been tested empirically and is just an intuitive assumption. The next experiment aims to study this.

All seven layers are transferred from the generalist network and then retrained using a different learning rate multiplier. The final eighth layer always has a learning rate multiplier of 1 because it was not transferred. This multiplier ranges from 0 to 1 in increments of .2. A multiplier of 0 represents a frozen transferred layer and a multiplier of 1 represents transferring with full "clobbering" enabled. This will allow the effect of learning rate in transferred layers to be measured. Once again, the source task is the generalist superset network and the target task is the specialist plane network. The scratch network also serves as the baseline comparison. Due to time constraints, it was not possible to test this exhaustively across all subsets.

3.2.4 Measuring Effect of Source Task on Target Task.

The optimal learn rate for transferring learned weights having been established, we next move on to analyzing the effect of different source tasks for the target task of fine-grained image classification. Unlike the previous experiments, this one is exhaustive across the entire dataset.

To measure the effect of transferring from different source tasks, we train a specialist network on each fine-grained dataset. However, this is done with 5 different initializations. The first is random initialization. This represents no transfer learning and serves as a control. The next two initializations used the weights from the high-fan and generalist superset networks as the initialization. This allows us to compare whether features learned in general datasets are more transferable than features learned in more specific datasets. Lastly, there are two more initializations. They come from high-fan and generalist networks trained on the entire superset *except for the major class targeted by the specialist network in question.* These are more distant source tasks compared to the high-fan and generalist network trained on the entire dataset. This allows us to measure the effect of distance on transferability. This

experiment is performed across all seven major classes. In total, 51 networks are trained for this experiment.

3.2.5 Comparing Ensemble Networks.

For the final experiment, we analyze the effect of transfer learning on ensemble networks. Fine-tuned ConvNets tend to outperform randomly initialized ConvNets. Ensembles of ConvNets tend to outperform individual ConvNets. This begs the question: do ensembles of fine-tuned ConvNets outperform ensembles of randomly initialized ConvNets?

For this, we compare the accuracy of an ensemble of four networks initialized from the different source tasks listed in the previous experiment to an ensemble of four networks initialized randomly. Since fine-tuned networks tend to outperform randomly initialized networks, we expect an ensemble of fine-tuned networks to outperform an ensemble of randomly initialized networks. Due to time constraints, the only dataset tested is the planes dataset. Four separate ConvNets are randomly initialized and trained per usual to serve as the first ensemble. The second ensemble is made up of the different initializations.

Evaluation Methods. The most important metric is overall accuracy. This is the number of correctly identified images divided by the total number of images. The top-1 error rate is also presented. This is the error rate in the class with the lowest overall accuracy. It is common to report this metric as well [31][46][48].

For each network, the confusion matrix of the best model trained (as determined by the accuracy on the validation set) is shown alongside a graph of the accuracy for each training iteration. This provides a visual way to compare networks and can sometimes reveal trends that other metrics don't. Each row of pixels in the confusion matrix shows how often a particular class appeared. Each column shows how often

a particular class was guessed by the network. Correct guesses are located on the diagonal. A dark blue pixel indicates that a guess of that type was made 0% of the time for the class represented by that row. A yellow pixel indicates a rate of 50%. A dark red pixel indicates a rate of 100%. An ideal network that correctly classifies every image would be dark red along the diagonal and dark blue everywhere else.

A graph showing the overall accuracy for every 100 training iterations is also shown. This is to give the reader a sense of how transfer learning affects the convergence of a network with transfer learning.

In the third experiment, we also examine the overall error reduction rate as a way to directly compare networks trained on different datasets. For example, consider a fine-tuned network with an overall accuracy of 75% and a scratch network with an overall accuracy of 50%. The error reduction rate would 50% since it eliminated half the errors in the scratch network.

In order to get a consolidated view of the results in the third experiment, two approaches were taken. The first is to look at the number of "winners" for each type of network. Table 11 shows this result. A winner is considered the top-performing network for that particular dataset. For example, the highest-performing source for the planes dataset is the generalist network. However, it is not a statistically significant win for the generalist network, so it is not considered a win in the second row. The third row tallies a win for all networks that were within the margin of error of the winning network. So for the planes dataset, both the generalist and the generalist-without networks earn a win in the third row.

The second approach is to measure the error reduction rate and use it as a normalizing factor to compare the results from different datasets. This is defined as

$$ReductionRate = \frac{1 - (ScratchErrors - TransferErrors)}{ScratchErrors} \tag{9}$$

where scratch errors refers to the number of errors made by the scratch network and transfer errors refers to the number of errors made by each of the fine-tuned networks. Consider a network that achieves an accuracy of 75%, and a second that achieves an accuracy of 80%. The second network eliminated one fifth of the errors from the first network and therefore achieved a 20% reduction in error rate. We apply this method to the entire dataset to normalize the results against the scratch network.

Taken together, these experiments should answer the questions presented at the beginning of this chapter. The first two experiments attempt to answer the first question. The next experiment addresses questions two, three, and four by varying the source task on a set of target tasks. Lastly, the final experiment directly addresses the last question. The results of these experiments are presented in the next chapter.

IV. Results

This chapter will now present the results of the experiments described in chapter three. The specificity of features experiment is performed first, followed by the attempt to find the optimal learn rate. Next, the experiment to identify the best source task is performed. Lastly, the effect on ensembles is studied.

4.1 Determining Specificity of Features

The first set of networks tests the effect of fine-tuning as the number of layers transferred increases. The source network was the the superset generalist network. The target dataset is the planes subset. Learning is frozen in the transferred layers.

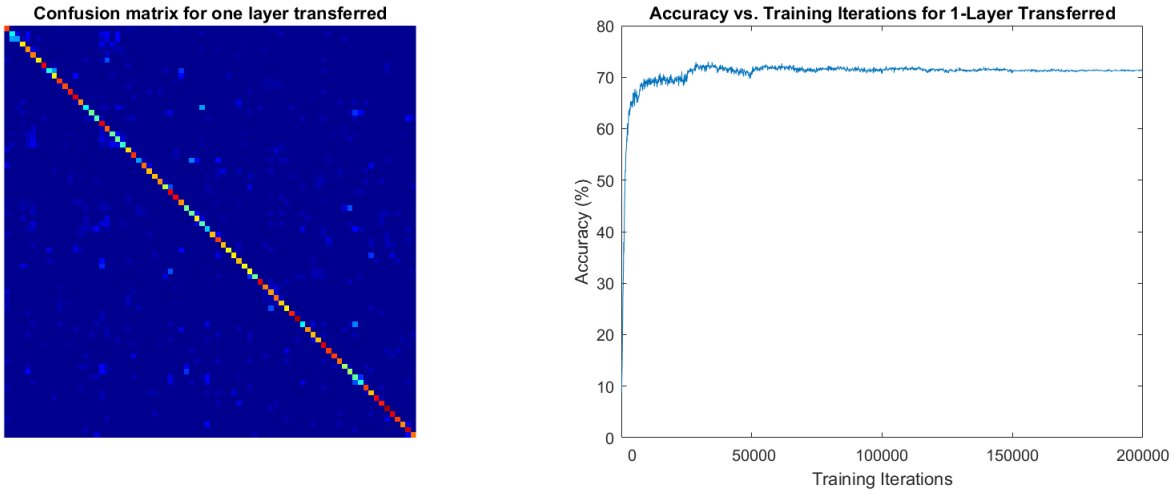

Figure 16. The results of the 1-layer network. Left: The confusion matrix generated by the best model with only 1-layer transferred. Right: The accuracy on the test set over training.

1-Layer. Figure 16 shows the test set accuracy vs. training iterations. The highest overall accuracy achieved was 71.326% after 33300 training iterations. The top-1 error rate of this model was 72%. Figure 16 also shows the confusion matrix.

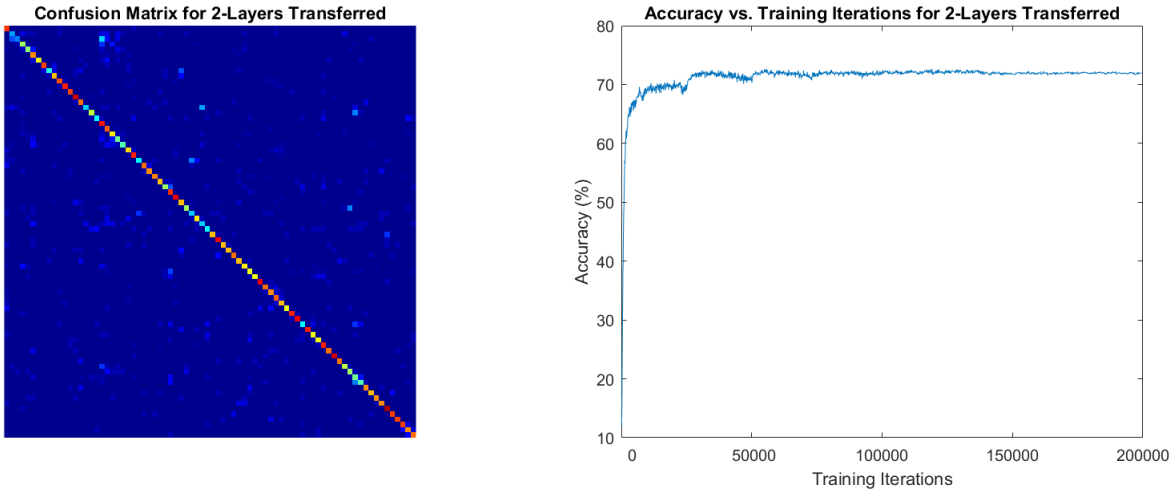

Figure 17. The results of the 2-layer network. Left: The confusion matrix generated by the best model with only 2-layers transferred. Right: The accuracy on the test set over training.

2-Layer. Figure 17 shows the test set accuracy vs. training iterations. The highest overall accuracy achieved was 71.371% after 119100 training iterations. The top-1 error rate of this model was 76%. Figure 17 also shows the confusion matrix.

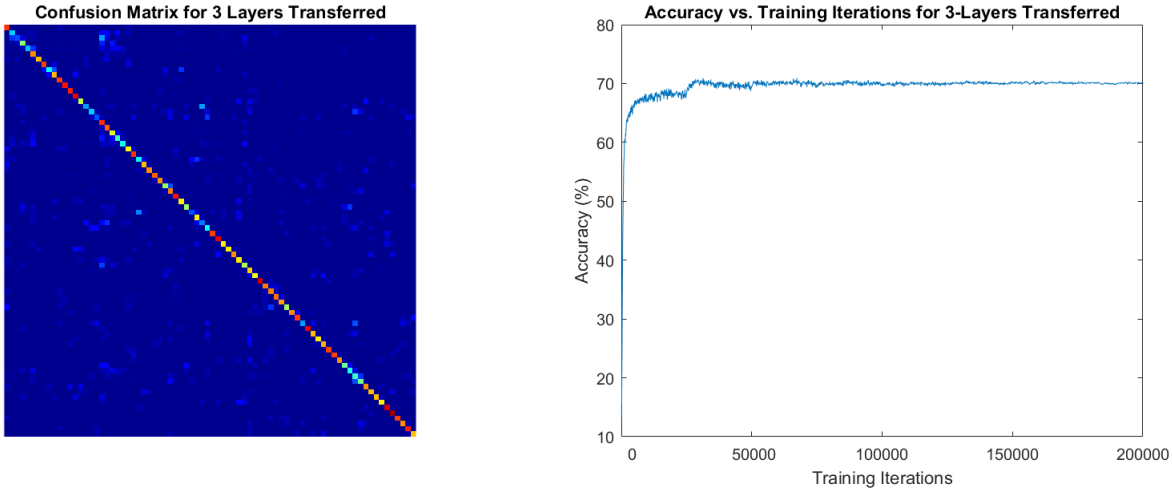

Figure 18. The results of the 3-layer network. Left: The confusion matrix generated by the best model with only 3-layers transferred. Right: The accuracy on the test set over training.

3-Layer. Figure 18 shows the test set accuracy vs. training iterations. The highest overall accuracy achieved was 69.191% after 67300 training iterations. The top-1 error rate of this model was 80%. Figure 18 also shows the confusion matrix.

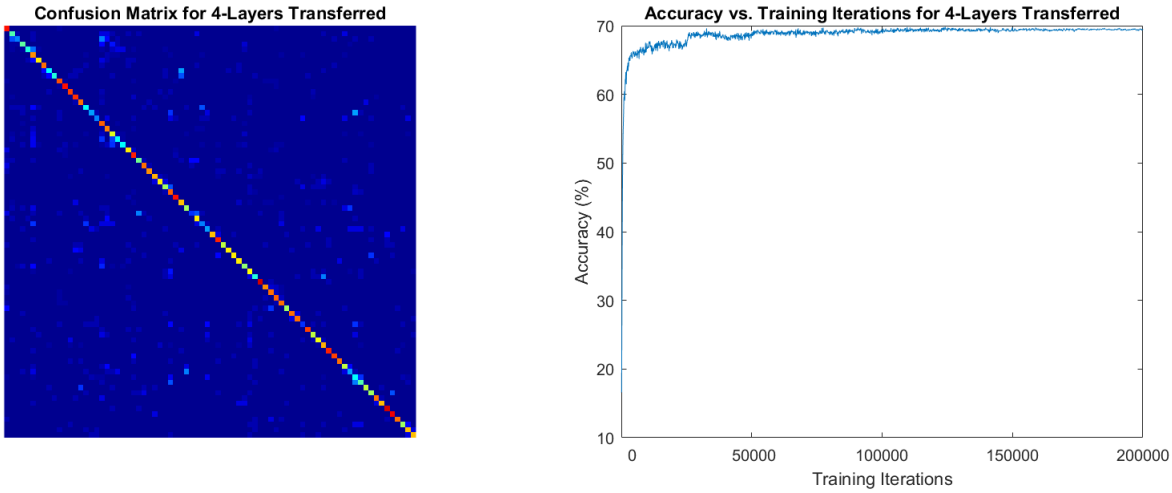

Figure 19. The results of the 4-layer network. Left: The confusion matrix generated by the best model with only 4-layers transferred. Right: The accuracy on the test set over training.

4-Layer. Figure 19 shows the test set accuracy vs. training iterations. The highest overall accuracy achieved was 68.202% after 109600 training iterations. The top-1 error rate of this model was 84%. Figure 19 also shows the confusion matrix.

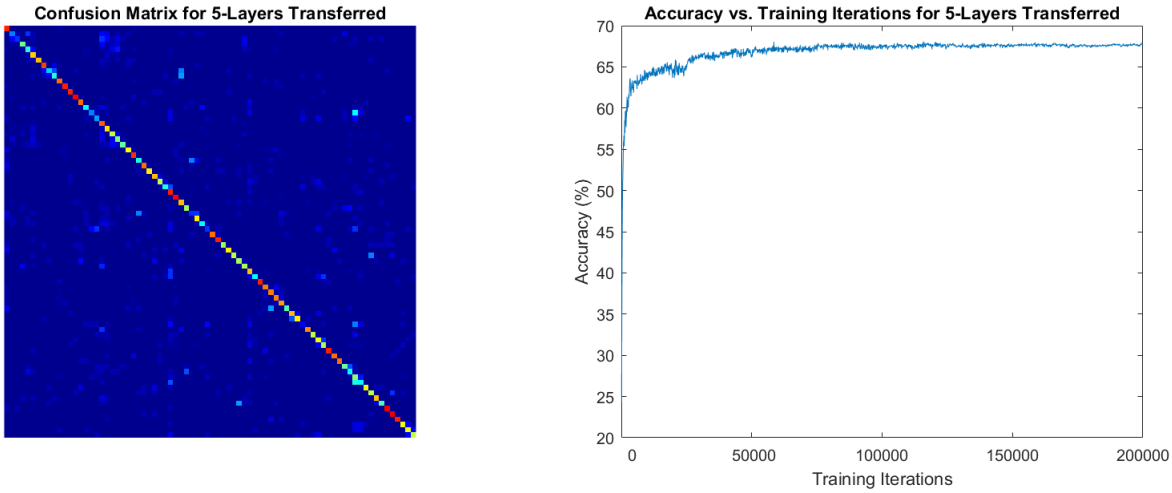

Figure 20. The results of the 5-layer network. Left: The confusion matrix generated by the best model with only 5-layers transferred. Right: The accuracy on the test set over training.

5-Layer. Figure 20 shows the test set accuracy vs. training iterations. The highest overall accuracy achieved was 66.719% after 115800 training iterations. The top-1 error rate of this model was 88%. Figure 20 also shows the confusion matrix.

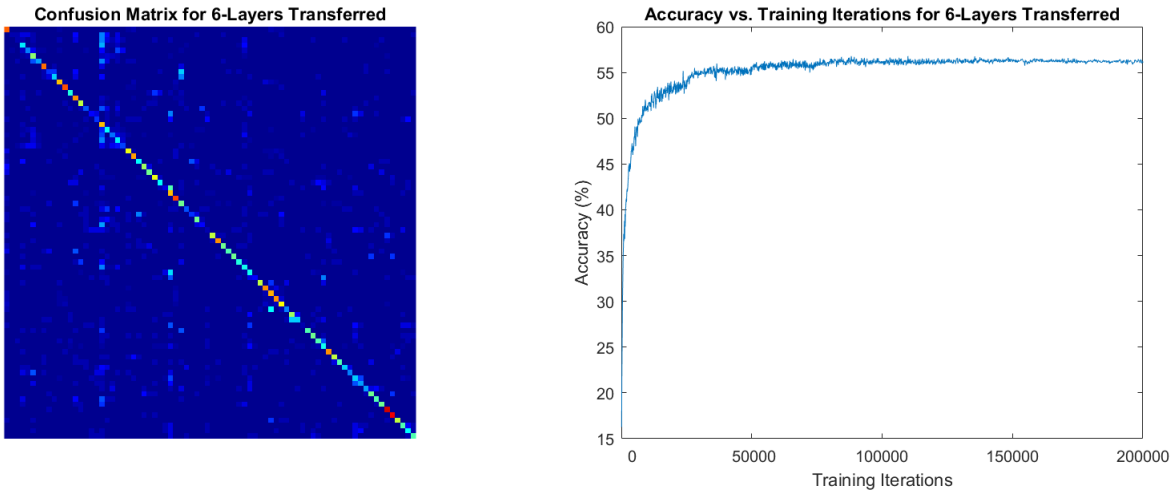

Figure 21. The results of the 6-layer network. Left: The confusion matrix generated by the best model with only 6-layers transferred. Right: The accuracy on the test set over training.

6-Layer. Figure 21 shows the test set accuracy vs. training iterations. The highest overall accuracy achieved was 55.281% after 88200 training iterations. The top-1 error rate of this model was 96%. Figure 21 also shows the confusion matrix.

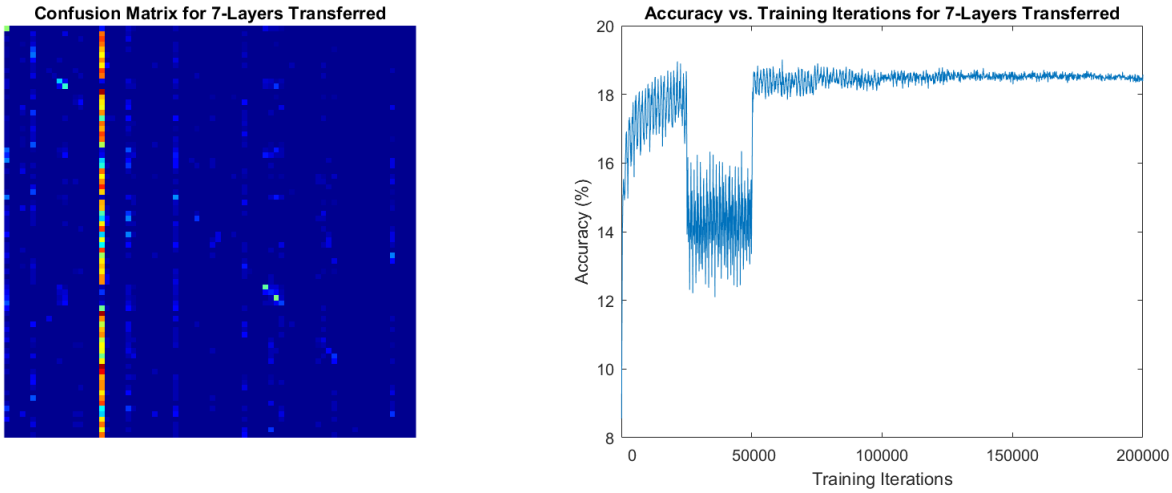

Figure 22. The results of the 7-layer network. Left: The confusion matrix generated by the best model with only 7-layers transferred. Right: The accuracy on the test set over training.

7-Layer. Figure 22 shows the test set accuracy vs. training iterations. The highest overall accuracy achieved was 18.584% after 125000 training iterations. The top-1 error rate of this model was 100%. Figure 22 also shows the confusion matrix. The confusion matrix shows several vertical lines, including one very prominent one. These lines correspond to classes that have the most training data relative to the other classes in the dataset.

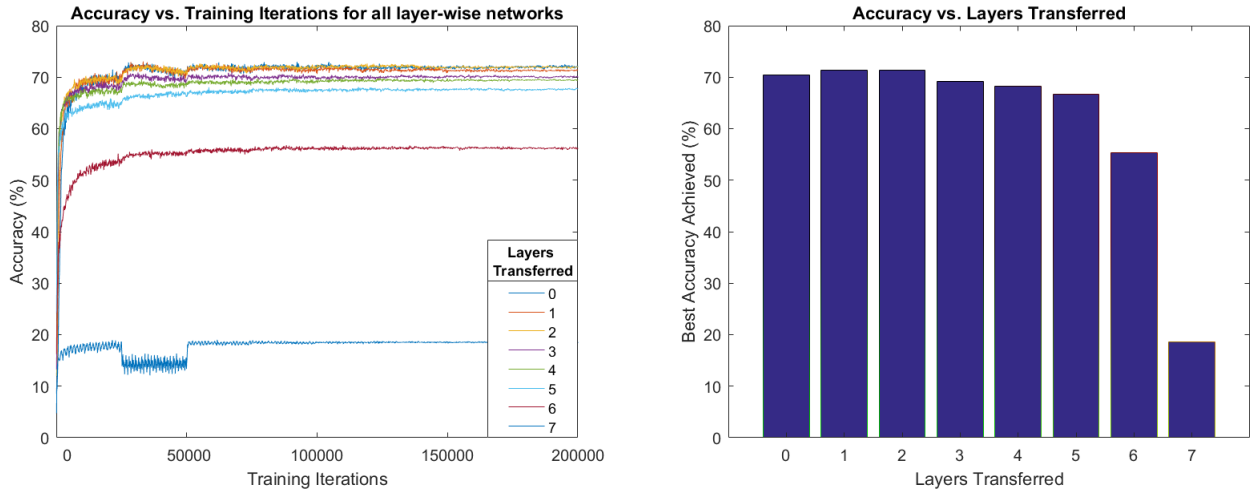

Figure 23. The results of all the layer-wise networks. Left: The performance of all the networks on one figure. Right: The best accuracy of each network.

Figure 82 shows the test results of all of the above networks on the same plot. To serve as a control, a network with no learned layers is shown for reference as well. See 72 for a more detailed view of it. The best network is 2-layers transferred, which performs marginally better than 1-layer transferred, although it had a worse top-1 error rate. Transferring up to two layers provided The figure clearly shows that as the number of layers transferred increases, the performance drops. There is little difference between each layer until the 6th layer and a drastic drop between the 6th and 7th layer. It is worth noting that the 5th layer is the last convolutional layer and that the 6th and 7th layers are fully-connected layers.

A difference of .61% accuracy is is considered statistically significant with 95% confidence for this dataset. Some of these layers have overlapping interval. The scratch network overlaps with the first three layers. Layers one and two over lap with each other. However, layers one and two do not overlap with layer three. Layer three and four overlap. Beyond this, all other results are considered statistically significant. This trend matches the results found by Yosinksi *et al* [52].

4.2 Optimizing Learn Rate

This experiment tests the effect of varying the learning rate in the transferred layers. Normally, all layers are trained at the same rate governed by a global learning rate. For this experiment, a multiplier ranging from 0 to 1 in increments of .2 was applied to the global learning rate in the transferred layers. All 7 layers were transferred from the generalist superset network and the target task was the planes subset.

Frozen learning. A new network was not trained for this parameter. It is the same network as the 7-layer network discussed above. For reference, the results are repeated here. Figure 22 shows the test set accuracy vs. training iterations. The highest overall accuracy achieved was 18.584% after 125000 training iterations. The top-1 error rate of this model was 100%. Figure 22 also shows the confusion matrix.

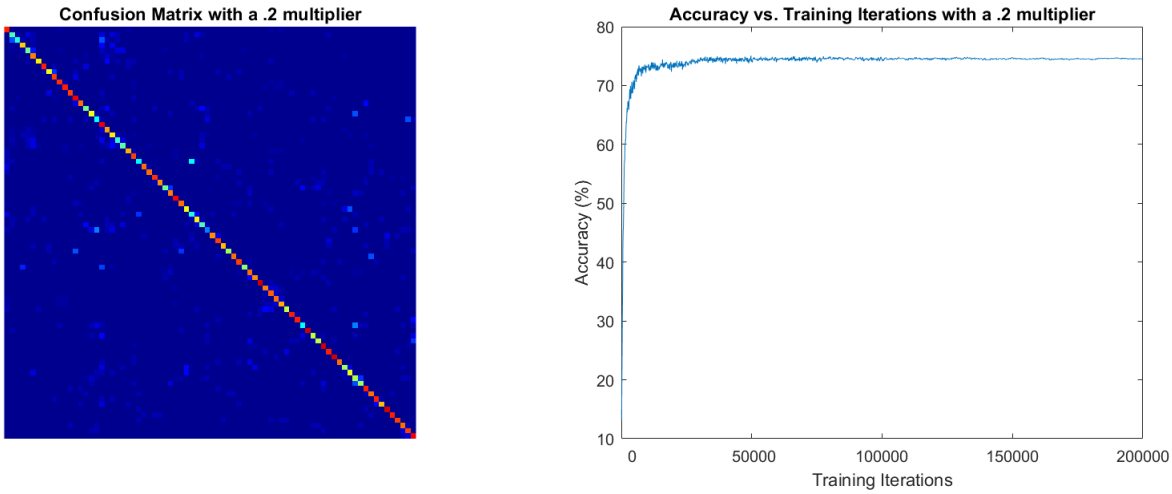

Figure 24. The result of the network with a .2 multiplier. Left: The performance of all the layer-wise networks on one figure based on their learn rate multiplier. Right: The best accuracy of each network.

.2 multiplier. Figure 24 shows the test set accuracy vs. training iterations. The highest overall accuracy achieved was 73.034% after 49700 training iterations. The top-1 error rate of this model was 68%. Figure 24 also shows the confusion matrix.

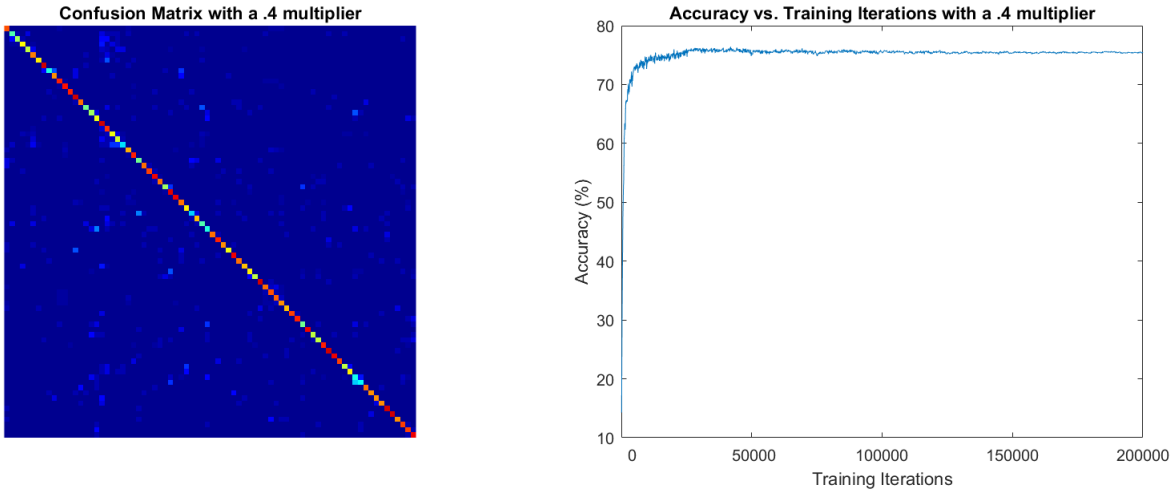

Figure 25. The result of the network with a .4 multiplier. Left: The confusion matrix generated by the best model trained with a .4 multiplier in the learned layers. Right: The accuracy on the test set over training.

.4 multiplier. Figure 25 shows the test set accuracy vs. training iterations. The highest overall accuracy achieved was 74.405% after 41600 training iterations. The top-1 error rate of this model was 68%. Figure 25 also shows the confusion matrix.

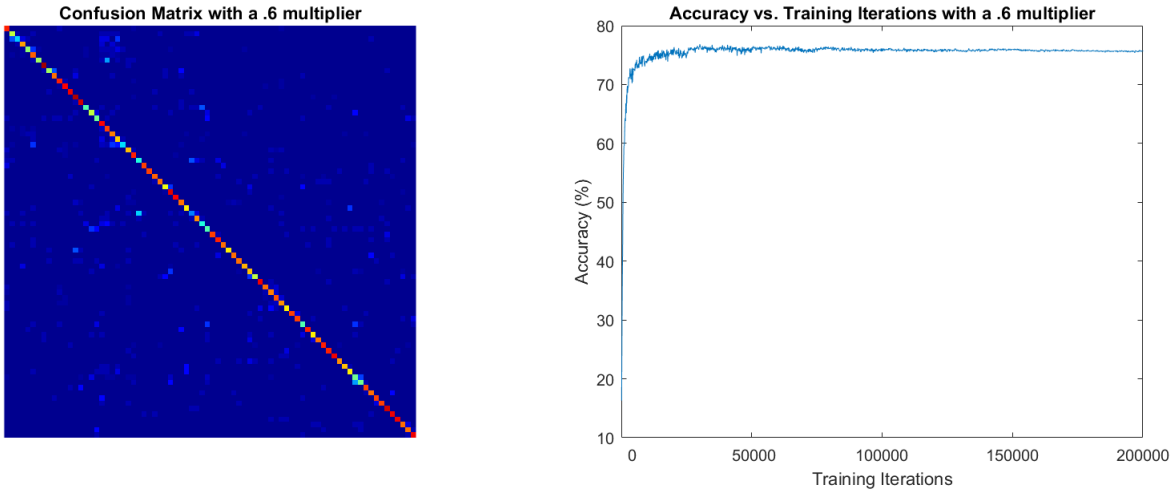

Figure 26. The result of the network with a .6 multiplier. Left: The confusion matrix generated by the best model trained with a .6 multiplier in the learned layers. Right: The best accuracy of each network.

.6 multiplier. Figure 25 shows the test set accuracy vs. training iterations. The highest overall accuracy achieved was 75.056% after 30100 training iterations. The top-1 error rate of this model was 76%. Figure 26 also shows the confusion matrix.

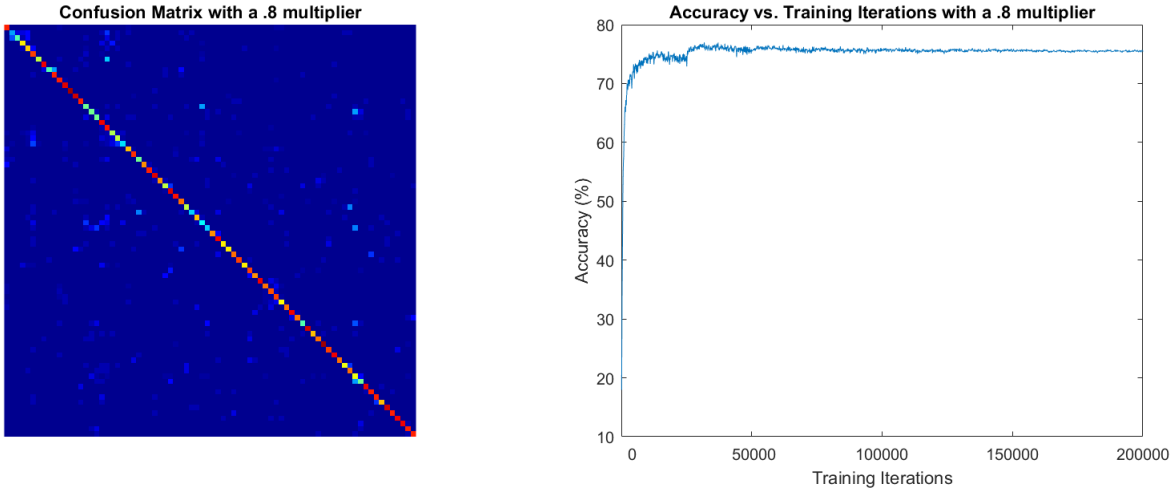

Figure 27. The result of the network with a .8 multiplier. Left: The confusion matrix generated by the best model trained with a .8 multiplier in the learned layers. Right: The best accuracy of each network.

.8 multiplier. Figure 27 shows the test set accuracy vs. training iterations. The highest overall accuracy achieved was 75.146% after 31200 training iterations. The top-1 error rate of this model was 72%. Figure 27 also shows the confusion matrix.

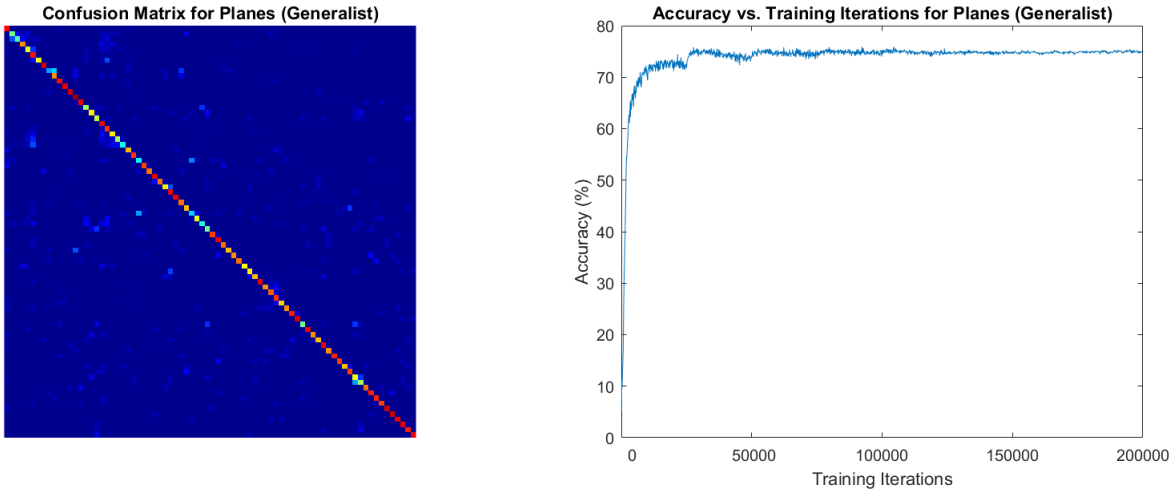

Figure 28. Results from the plane generalist network. Left: The confusion matrix generated by the best model for plane with initialization from the generalist superset network. Right: The accuracy on the test set over training.

Global learning rate. A network with no multiplier in the learned layers is one that uses the global learning rate throughout. This is the same as the generalist plane network in third experiment. Figure 73 shows the test set accuracy vs. training iterations. The highest overall accuracy achieved was 75.933% after 30600 training iterations. The top-1 error rate of this model was 72%. Figure 73 also shows the confusion matrix.

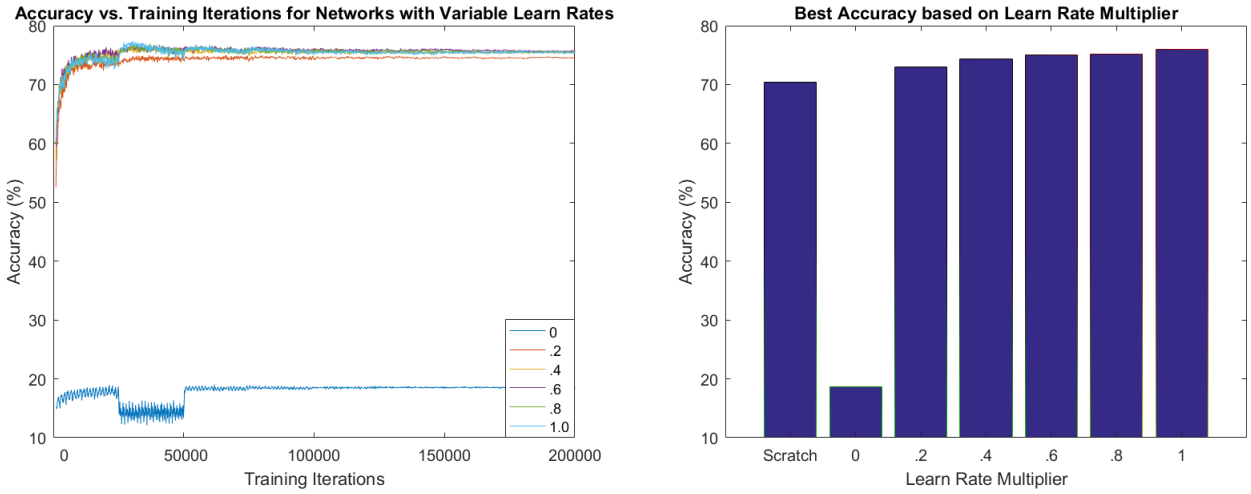

Figure 29. The combined results of all the networks with different learning rate multipliers. Left: The performance of all the networks on one figure based on their learn rate multiplier. Right: The best accuracy of each network.

Figure 83 shows the test set accuracy vs. training iterations for all of the above but only shows the first 100000 training iterations for clarity. The results show that as the learn rate multiplier approaches 1, the performance improves. A difference of .61% accuracy is considered statistically significant with 95% confidence. The difference between a .6 and .8 multiplier was not statistically significant; however, the difference between all other increments was statistically significant.

4.3 Measuring Effect of Source Task on Target Task

Next, we will evaluate the effect of different source tasks on the target task of fine-grained classification for each dataset detailed in chapter three. For each category in the dataset, five different initializations are tested and evaluated. First, the random or "scratch" initialization of a particular specialist network is shown. It serves as the control and baseline for that category. Then the results from the generalist and high-fan specialist networks are shown. Lastly, the results from the generalist and high-fan superset specialist networks without the category in question are shown. Lastly, the

combined results of all the different initializations are shown for easy comparison. The results are presented in the same format as before. For each network trained, a confusion matrix is presented from the best model as well as a figure showing the accuracy over time. Some figures use "-W" as a shorthand for the superset network trained on the entire dataset without the particular subset in question. For example, the plane specialist network initialized from the weights of the generalist superset network without planes would be referred to as "generalist-without" or "generalist-w".

4.3.1 Signs.

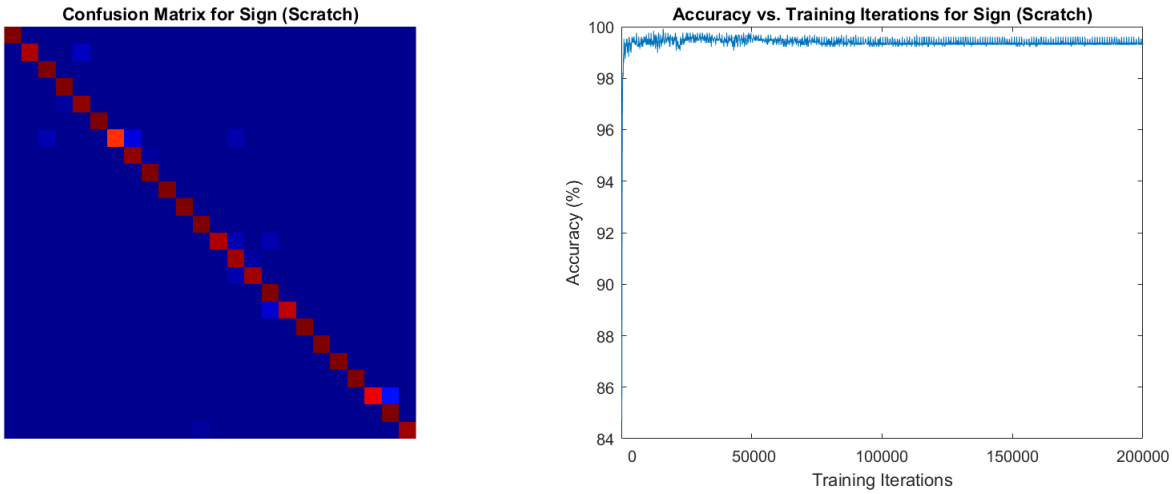

Figure 30. Results from the sign scratch network. Left: The confusion matrix generated by the best model for sign with random initialization. Right: The accuracy on the test set over training.

Scratch initialization. Figure 30 shows the test set accuracy vs. training iterations with random initialization. The highest overall accuracy achieved was 97.743% after 16000 training iterations. The top-1 error rate of this model was 20.513%.

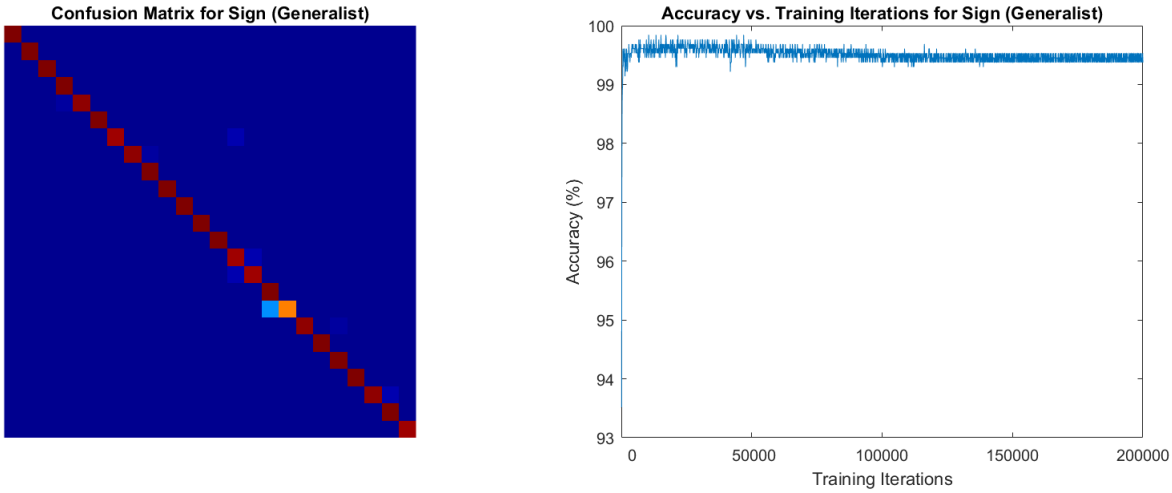

Figure 31. Results from the sign generalist network. Left: The confusion matrix generated by the best model for sign with initialization from the generalist superset network. Right: The accuracy on the test set over training.

Generalist initialization. Figure 31 shows the test set accuracy vs. training iterations with initialization from the generalist network. The highest overall accuracy achieved was 98.153% after 44200 training iterations. The top-1 error rate of this model was 25.641%. Compared to random initialization, the overall accuracy is higher by .41% and the top-1 error rate is higher by 5.128. As can be seen in the confusion matrix in figure 31, almost all of the errors are located in a single class

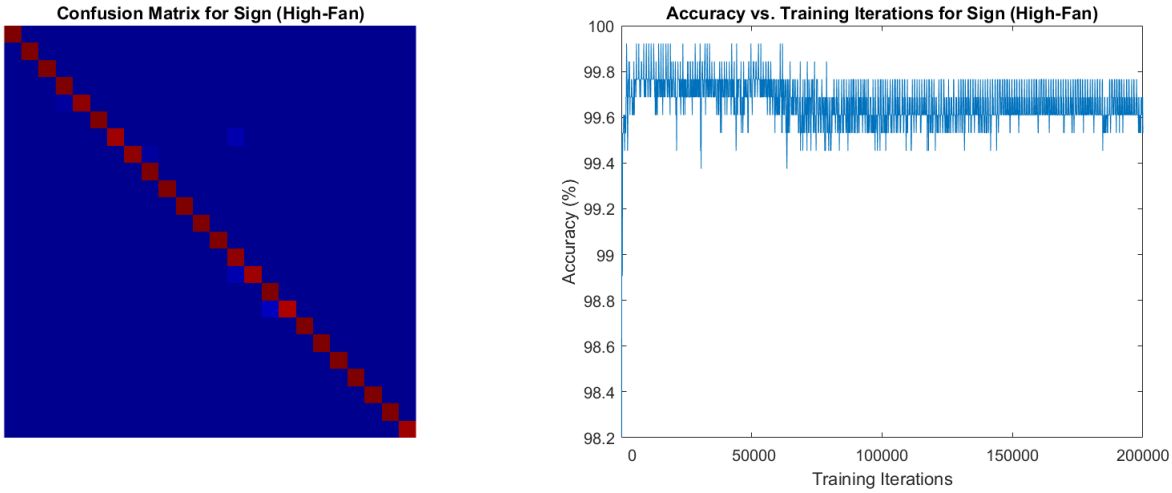

Figure 32. Results from the sign high-fan network. Left: The confusion matrix generated by the best model for sign with initialization from the high-fan superset network. Right: The accuracy on the test set over training.

High-Fan initialization. Figure 32 shows the test set accuracy vs. training iterations with initialization from the high-fan network. The highest overall accuracy achieved was 98.974% after 61800 training iterations. The top-1 error rate of this model was 5.128%. Compared to random initialization, the overall accuracy is 1.231% high and the top-1 error rate is 15.89% lower.

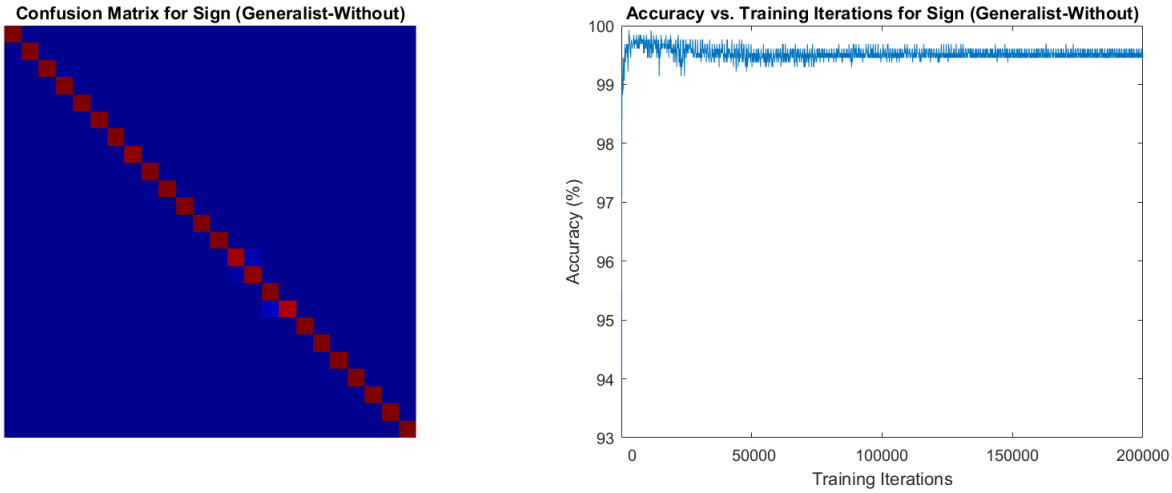

Figure 33. Results from the sign generalist-without network. Left: The confusion matrix generated by the best model for sign with initialization from the generalist superset network without sign data. Right: The accuracy on the test set over training.

Generalist Without Signs Initialization. Figure 33 shows the test set accuracy vs. training iterations with initialization from the more distant generalist without signs network. The highest overall accuracy achieved was 99.111% after 11400 training iterations. The top-1 error rate of this model was 5.128%. Compared to random initialization, the overall accuracy increased by 1.368% and the top-1 error rate is lower by 15.89%.

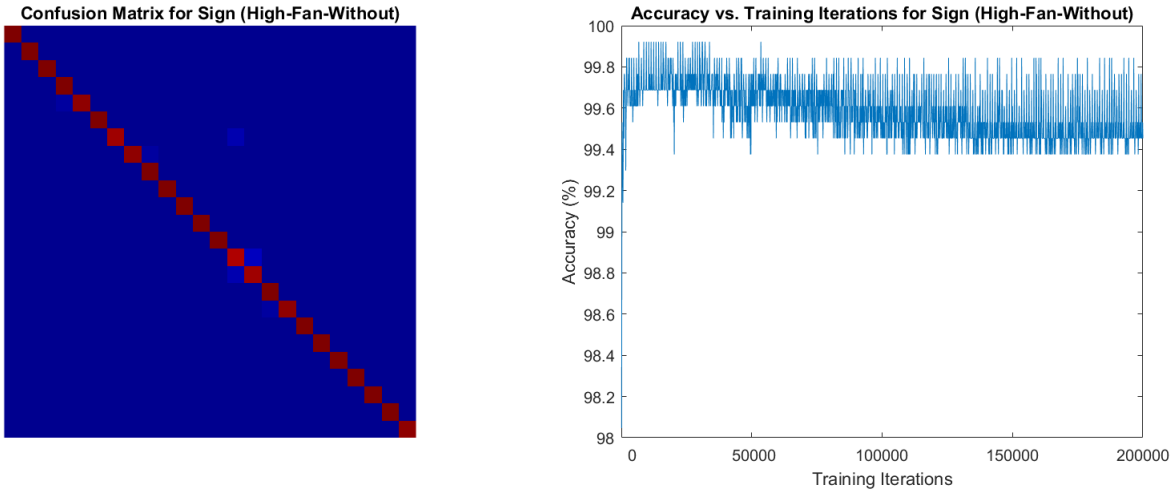

Figure 34. Results from the sign high-fan-without network. Left: The confusion matrix generated by the best model for sign with initialization from the high-fan superset network without sign data. Right: The accuracy on the test set over training.

High-Fan Without Signs Initialization. Figure 34 shows the test set accuracy vs. training iterations with initialization from the more distant high-fan without signs network. The highest overall accuracy achieved was 98.974% after 31000 training iterations. The top-1 error rate of this model was 4.545%. Compared to random initialization, the overall accuracy increased by 1.231% and the top-1 error rate is lower by 15.968%.

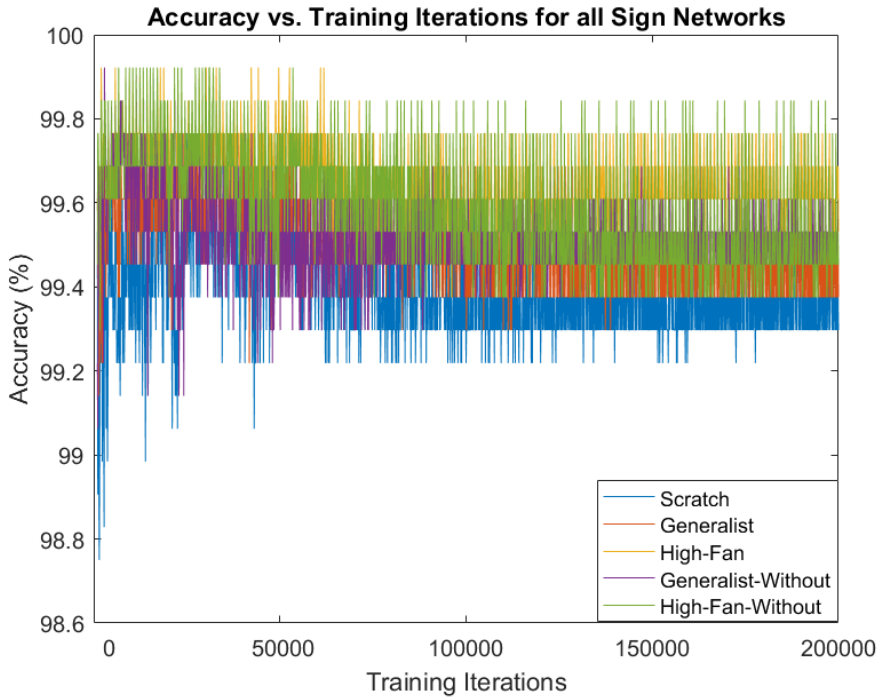

Figure 35. The performance of all sign networks on one figure.

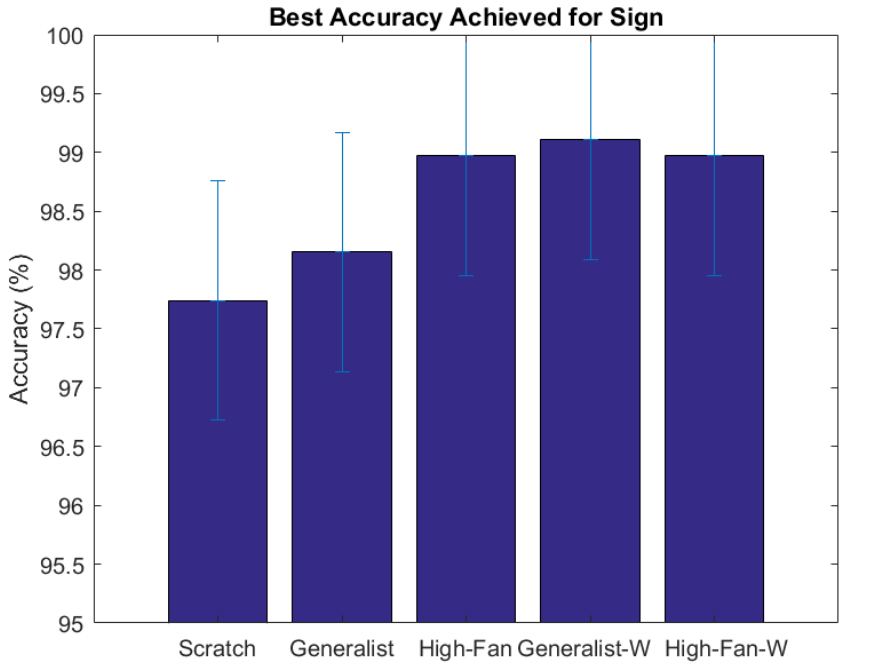

Figure 36. The best accuracy of all sign networks on one figure.

Figures 35 and 36 show the overall results for the sign dataset. In all cases, transfer learning improved the performance over the random initialization. The best source was the more distant generalist without signs superset network, which had the greatest increase in overall accuracy and the greatest reduction in the top-1 error rate of 99.111% and 5.128%, respectively. It also converged the fastest in only 11400 training iterations. All other transferred networks took longer to converge than the scratch network which took 16000 training iterations. It is also worth noting that the class of the top-1 error rate was the same for all networks except for the high-fan without signs initialization

It is worth noting that 95% confidence in statistical significance requires a differences of 2.4%. All of the models fall within this margin. A larger dataset is needed to overcome this uncertainty.

4.3.2 Vegetables.

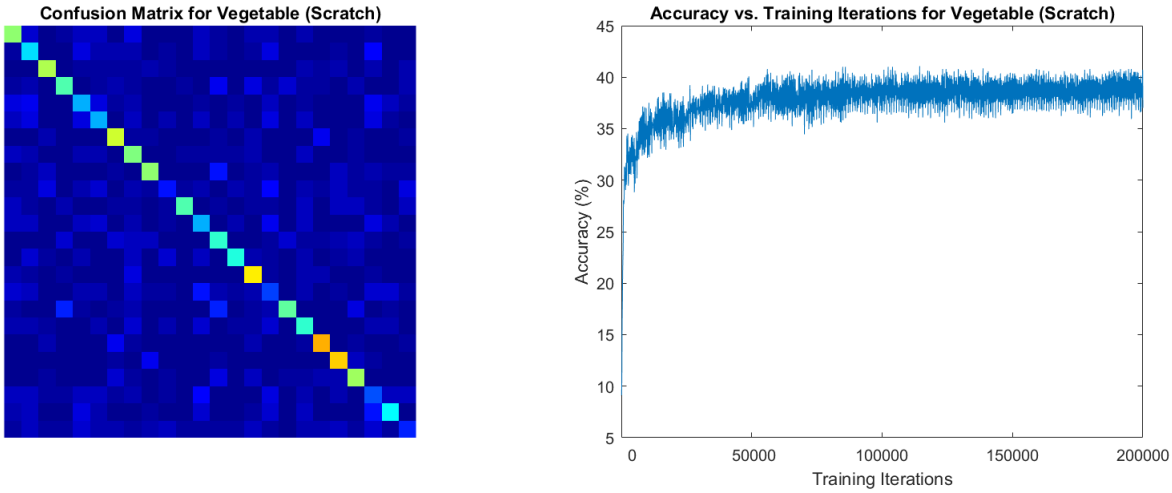

Figure 37. Results from the vegetable scratch network. Left: The confusion matrix generated by the best model for vegetable with random initialization. Right: The accuracy on the test set over training.

Scratch Initialization. Figure 37 shows the test set accuracy vs. training iterations with random initialization. The highest overall accuracy achieved was 41.071% after 114400 training iterations. The top-1 error rate of this model was 86%.

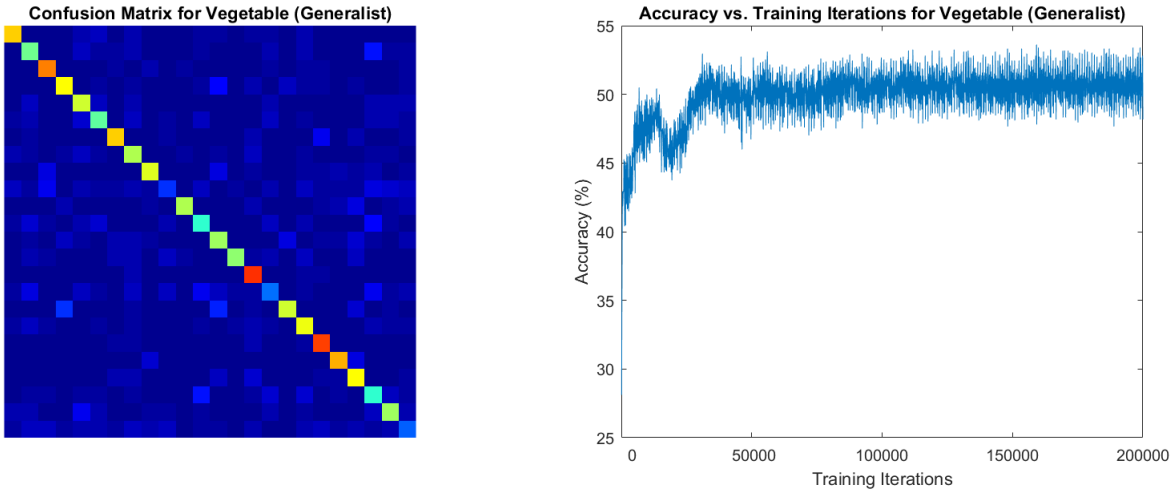

Figure 38. Results from the vegetable generalist network. Left: The confusion matrix generated by the best model for vegetable with initialization from the generalist superset network. Right: The accuracy on the test set over training.

Generalist Initialization. Figure 38 shows the test set accuracy vs. training iterations with initialization from the generalist network. The highest overall accuracy achieved was 53.621% after 159400 training iterations. The top-1 error rate of this model was 83.5%. Compared to random initialization, the overall accuracy increased by 12.551% and the top-1 error rate was reduced by 2.5%.

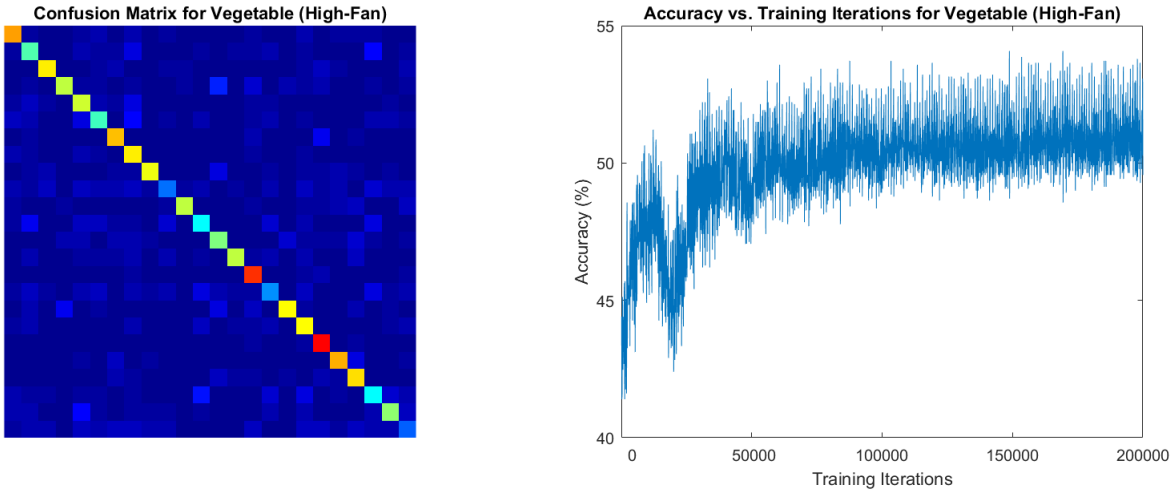

Figure 39. Results from the vegetable high-fan network. Left: The confusion matrix generated by the best model for vegetable with initialization from the high-fan superset network. Right: The accuracy on the test set over training.

High-Fan Initialization. Figure 39 shows the test set accuracy vs. training iterations with initialization from the high-fan network. The highest overall accuracy achieved was 54.077% after 169400 training iterations. The top-1 error rate of this model was 77.665%. Compared to random initialization, the overall accuracy increased by 13.007% and the top-1 error rate was reduced by 8.335%

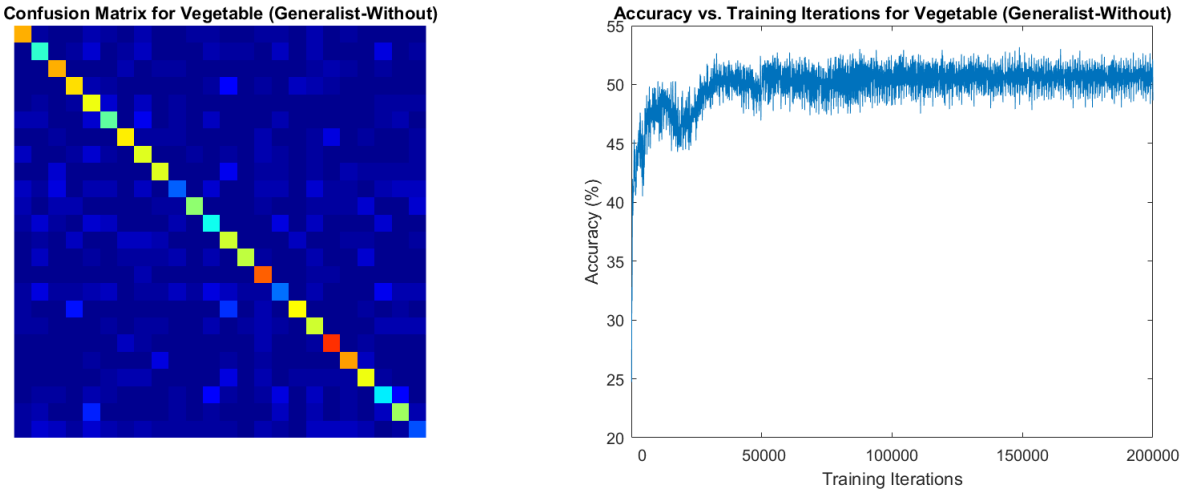

Figure 40. Results from the vegetable generalist-without network. Left: The confusion matrix generated by the best model for vegetable with initialization from the generalist superset network without vegetable data. Right: The accuracy on the test set over training.

Generalist Without Vegetable Initialization. Figure 40 shows the test set accuracy vs. training iterations with initialization from the generalist without vegetables network. The highest overall accuracy achieved was 53.166% after 149000 training iterations. The top-1 error rate of this model was 80.203%. Compared to random initialization, the overall accuracy increased by 12.096% and the top-1 error rate was reduced by 5.797%.

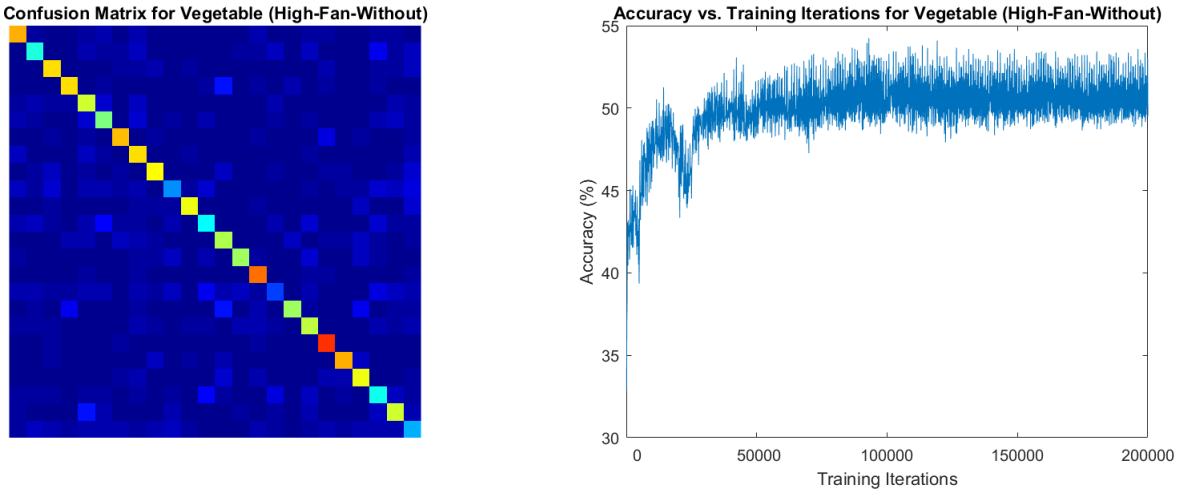

Figure 41. Results from the vegetable high-fan-without network. Left: The confusion matrix generated by the best model for vegetable with initialization from the high-fan superset network without vegetable data. Right: The accuracy on the test set over training.

High-Fan Without Vegetable Initialization. Figure 41 shows the test set accuracy vs. training iterations with initialization from the high-fan without vegetables network. The highest overall accuracy achieved was 54.237% after 93000 training iterations. The top-1 error rate of this model was 81.5%. Compared to random initialization, the overall accuracy increased by 13.167% and the top-1 error rate was reduced by 4.5%.

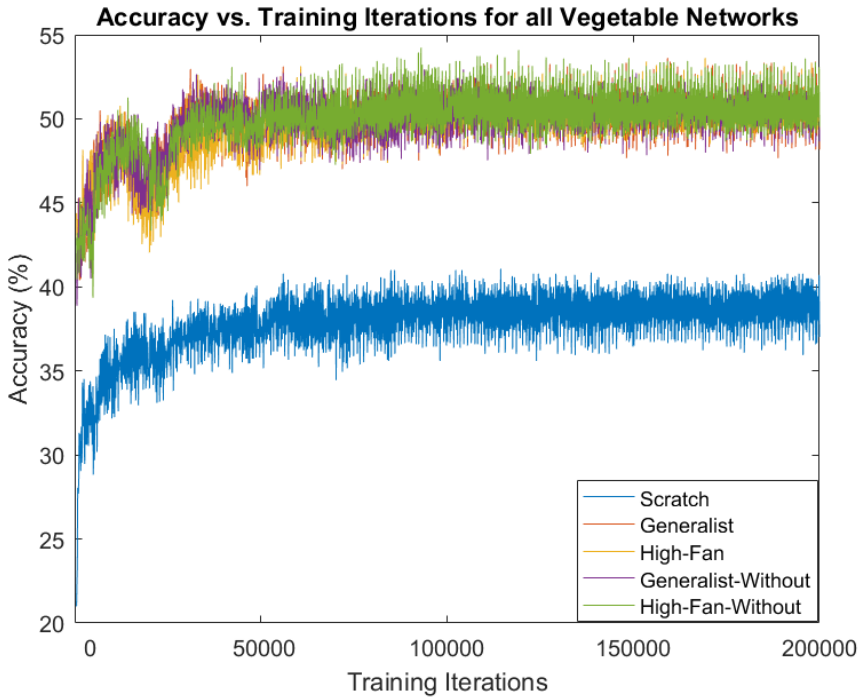

Figure 42. The performance of all vegetable networks on one figure.

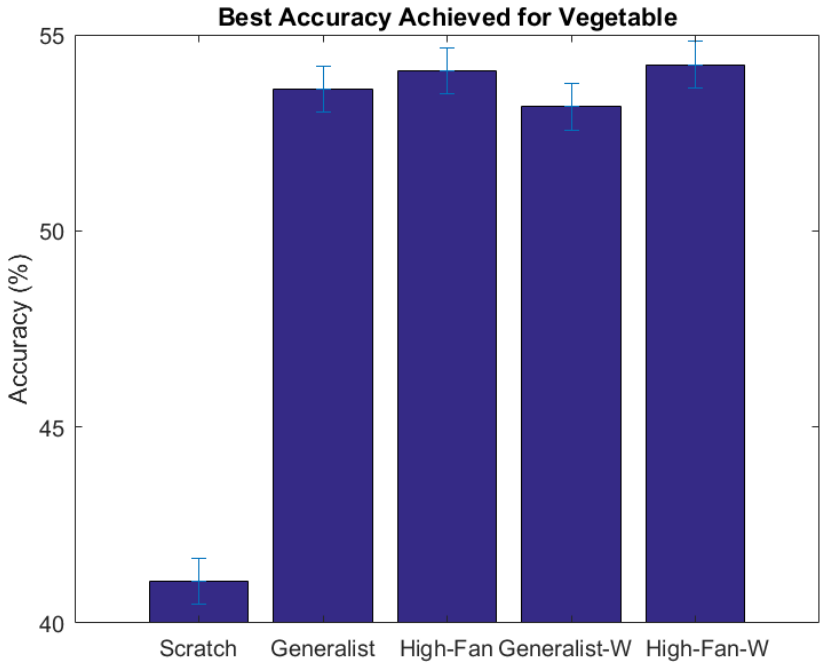

Figure 43. The best accuracy of all vegetable networks on one figure.

Figures 42 and 43 show the overall results for the vegetable dataset. In all cases, transfer learning improved the performance over the random initialization. The best source was the more distant high-fan without vegetables superset network, which had the greatest increase in overall accuracy to 54.237%. However, the high-fan superset network was the best source for reducing the top-1 error rate. It had a top-1 error rate of 77.650% compared to the scratch initialization network's top-1 error rate of 86%. The high-fan without vegetable network also converged the fastest at 93000 training iterations. All other transferred networks took longer to converge than the scratch network which took 114400 training iterations. The class responsible for the top-1 error rate was different for each network.

A difference of .61% is considered statistically significant with 95% confidence. All of the transferred networks are within the margin of error except for the scratch network.

4.3.3 Dogs.

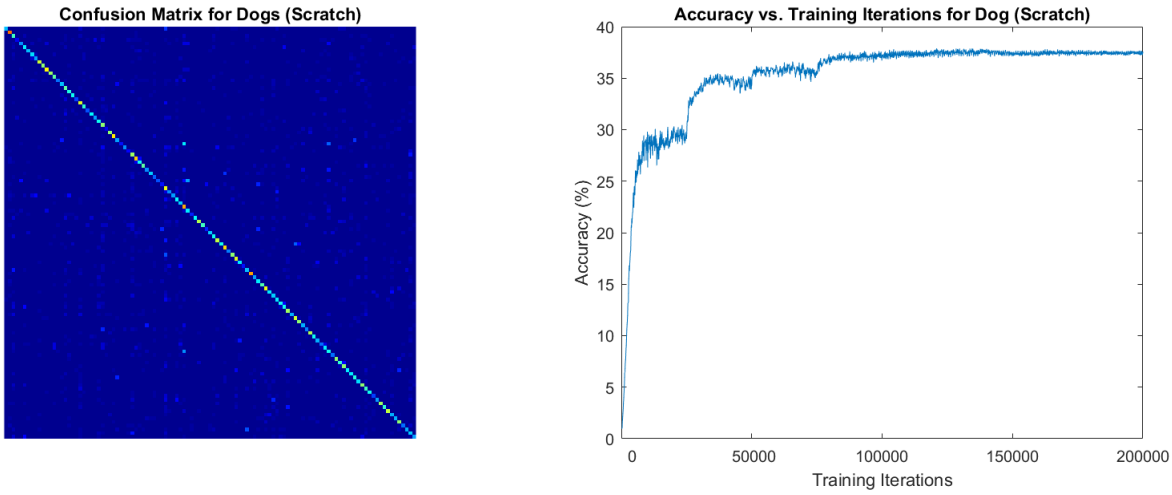

Figure 44. Results from the dog scratch network. Left: The confusion matrix generated by the best model for dog with random initialization. Right: The accuracy on the test set over training.

Scratch Initialization. Figure 44 shows the test set accuracy vs. training iterations with random initialization. The highest overall accuracy achieved was 37.885% after 134200 training iterations. The top-1 error rate of this model was 90%.

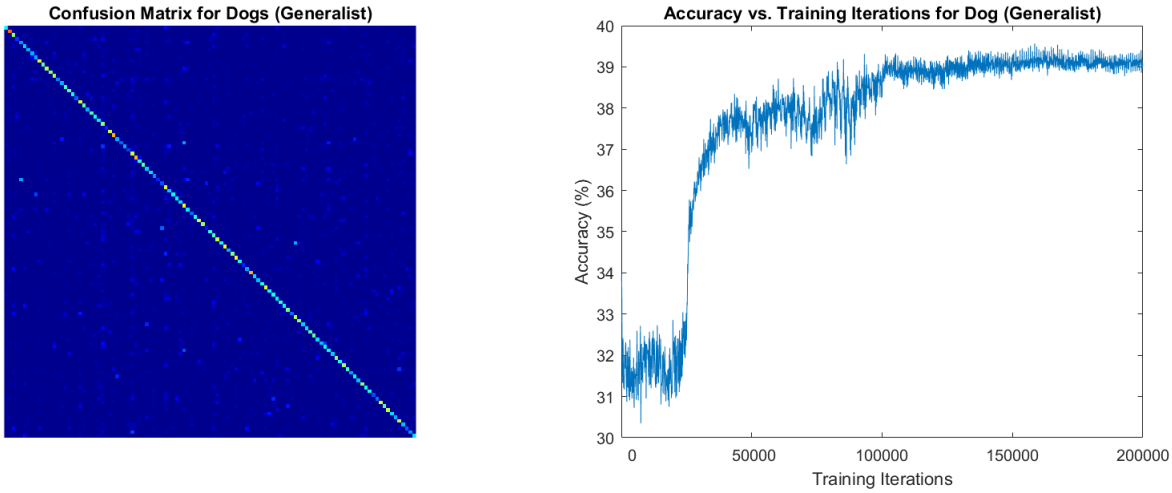

Figure 45. Results from the dog generalist network. Left: The confusion matrix generated by the best model for dog with initialization from the generalist superset network. Right: The accuracy on the test set over training.

Generalist Initialization. Figure 45 shows the test set accuracy vs. training iterations with initialization from the generalist network. The highest overall accuracy achieved was 39.568% after 158400 training iterations. The top-1 error rate of this model was 86%. Compared to random initialization, the overall accuracy increased by 1.684% and the top-1 error rate was reduced by 4%.

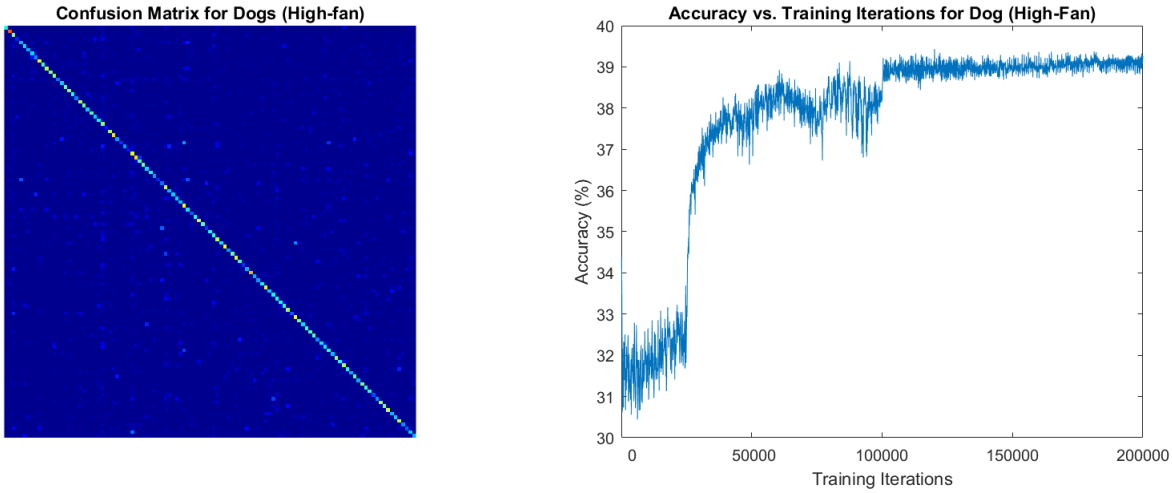

Figure 46. Results from the dog high-fan network. Left: The confusion matrix generated by the best model for dog with initialization from the high-fan superset network. Right: The accuracy on the test set over training.

High-Fan Initialization. Figure 46 shows the test set accuracy vs. training iterations with initialization from the high-fan network. The highest overall accuracy achieved was 39.434% after 120000 training iterations. The top-1 error rate of this model was 87%. Compared to random initialization, the overall accuracy increased by 1.55% and the top-1 error rate was reduced by 3%.

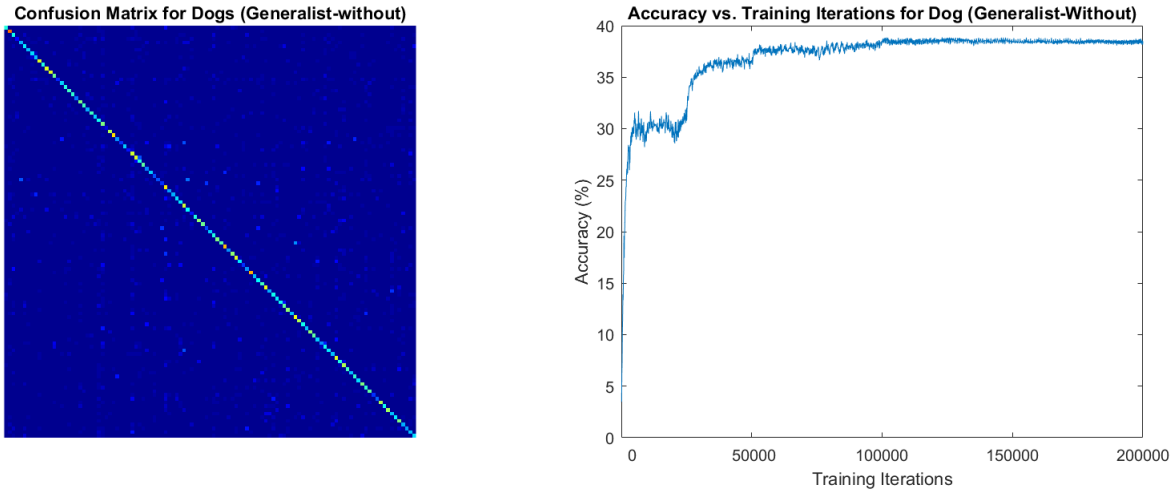

Figure 47. Results from the dog generalist-without network. Left: The confusion matrix generated by the best model for dog with initialization from the generalist superset network without dog data. Right: The accuracy on the test set over training.

Generalist Without Dogs Initialization. Figure 47 shows the test set accuracy vs. training iterations with initialization from the more distant generalist without dogs network. The highest overall accuracy achieved was 38.869% after 122800 training iterations. The top-1 error rate of this model was 92%. Compared to random initialization, the overall accuracy increased by .985% and the top-1 error rate was *increased* by 2%.

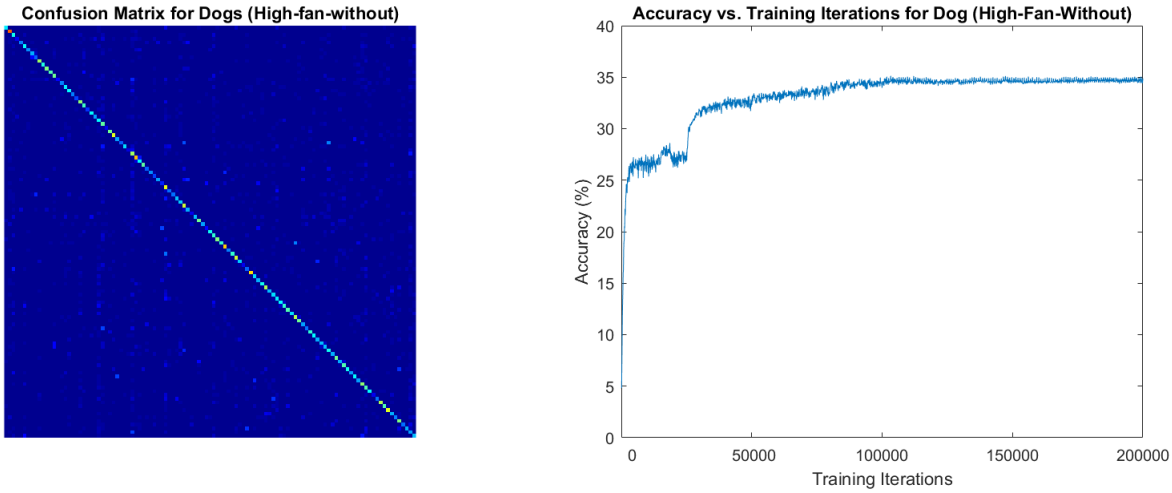

Figure 48. Results from the dog high-fan-without network. Left: The confusion matrix generated by the best model for dog with initialization from the high-fan superset network without dog data. Right: The accuracy on the test set over training.

High-Fan Without Dogs Initialization. Figure 44 shows the test set accuracy vs. training iterations with initialization from the more distant high-fan without dogs network. The highest overall accuracy achieved was 35.127% after 103400 training iterations. The top-1 error rate of this model was 93%. Compared to random initialization, the overall accuracy *decreased* by 2.757% and the top-1 error rate was *increased* by 3%.

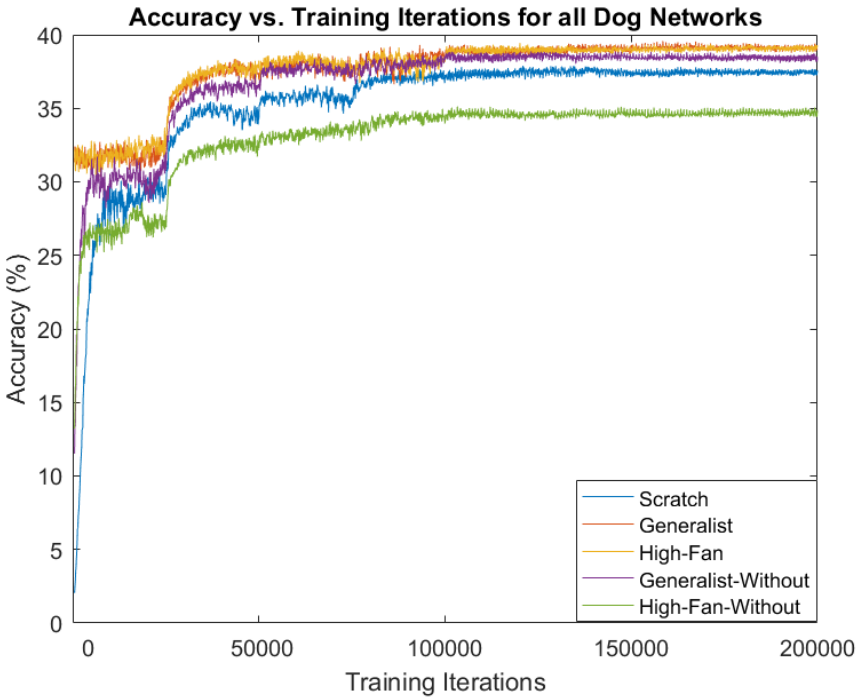

Figure 49. The performance of all dog networks on one figure.

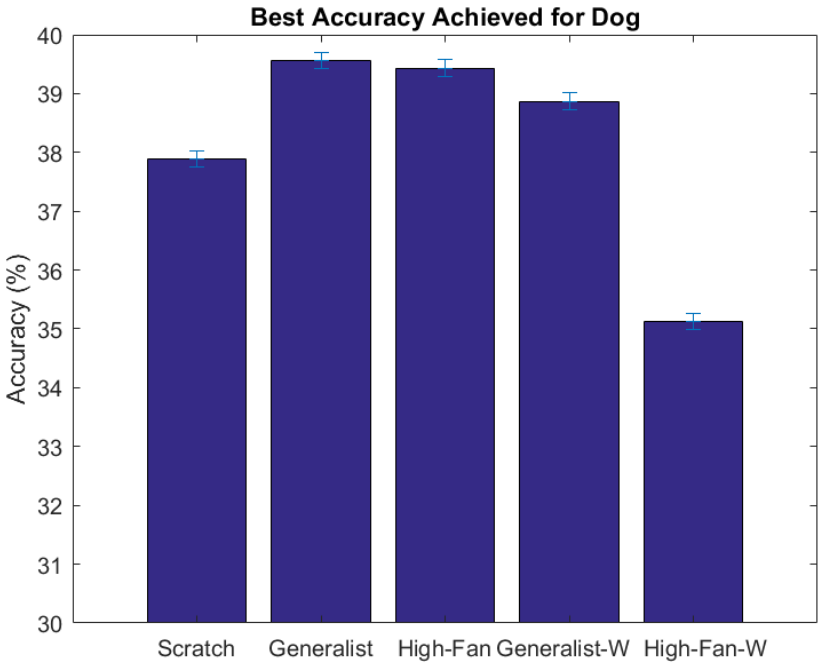

Figure 50. The best accuracy of all dog networks on one figure.

Figures 49 and 50 show the overall results for the Dog dataset. In all cases but one, transfer learning improved the performance over the random initialization. The best source was the generalist superset network, which had the greatest increase in overall accuracy to 39.568%. It was also the best source for reducing the top-1 error rate. It had a top-1 error rate of 86% compared to the scratch initialization network's top-1 error rate of 90%. The generalist network also took the longest to converge at 158400 training iterations. All other transferred networks took less training to converge than the scratch network, which took 134200 training iterations. The class responsible for the top-1 error rate was different for each network, except for the two generalist networks.

This dataset requires a difference of .15% accuracy in order to be considered statistically significant with a 95% confidence. The high-fan and generalist networks do not meet this criteria for statistical significance in their difference with each other.

4.3.4 Cats.

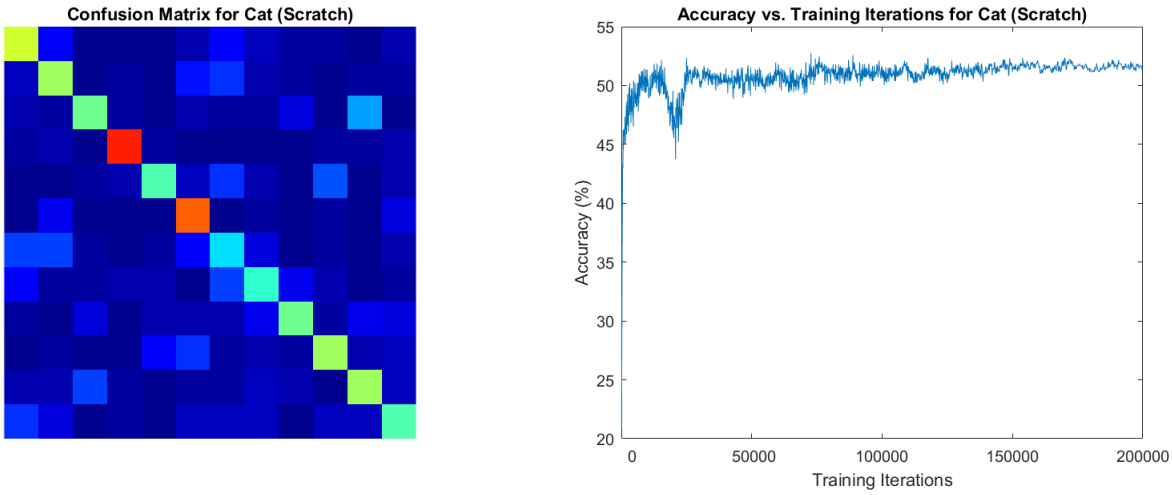

Figure 51. Results from the cat scratch network. Left: The confusion matrix generated by the best model for cat with random initialization. Right: The accuracy on the test set over training.

Scratch Initialization. Figure 51 shows the test set accuracy vs. training iterations with random initialization. The highest overall accuracy achieved was 52.764% after 72800 training iterations. The top-1 error rate of this model was 66%.

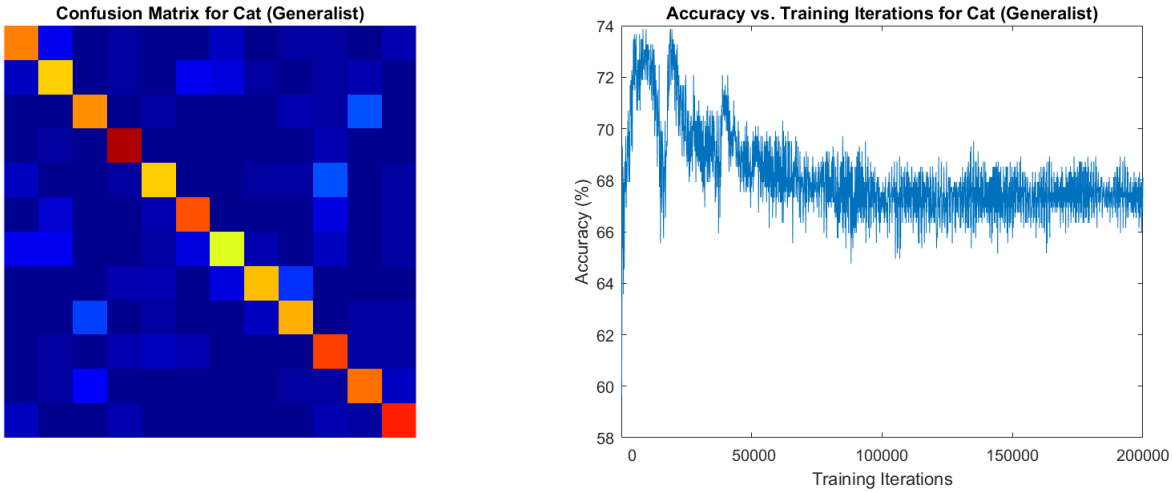

Figure 52. Results from the cat generalist network. Left: The confusion matrix generated by the best model for cat with initialization from the generalist superset network. Right: The accuracy on the test set over training.

Generalist Initialization. Figure 52 shows the test set accuracy vs. training iterations with initialization from the generalist network. The highest overall accuracy achieved was 73.869% after 19400 training iterations. The top-1 error rate of this model was 42%. Compared to random initialization, the overall accuracy increased by 21.105% and the top-1 error rate was reduced by 24%.

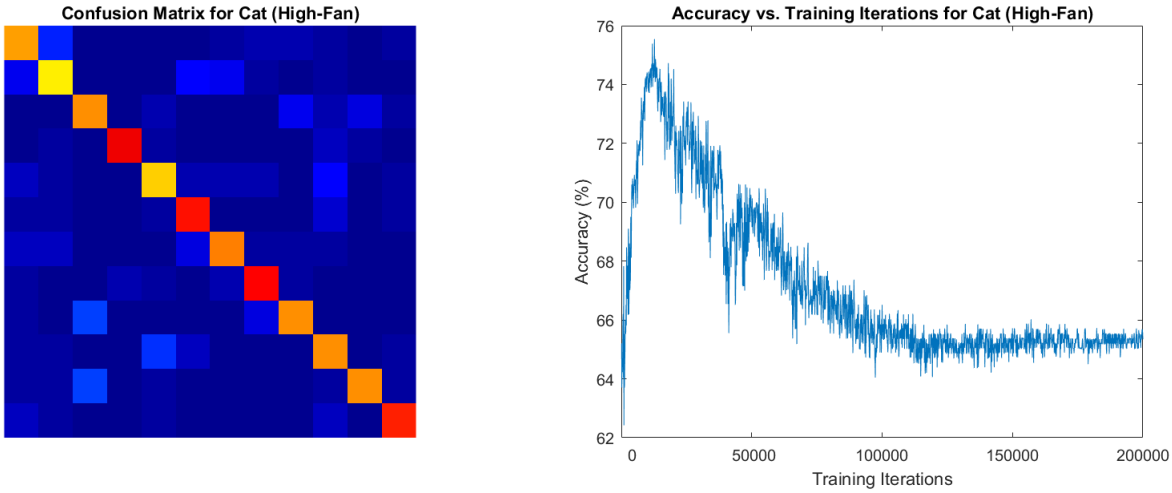

Figure 53. Results from the cat high-fan network. Left: The confusion matrix generated by the best model for cat with initialization from the high-fan superset network. Right: The accuracy on the test set over training.

High-Fan Initialization. Figure 53 shows the test set accuracy vs. training iterations with initialization from the high-fan network. The highest overall accuracy achieved was 75.544% after 12600 training iterations. The top-1 error rate of this model was 36%. Compared to random initialization, the overall accuracy increased by 22.78% and the top-1 error rate was reduced by 30%.

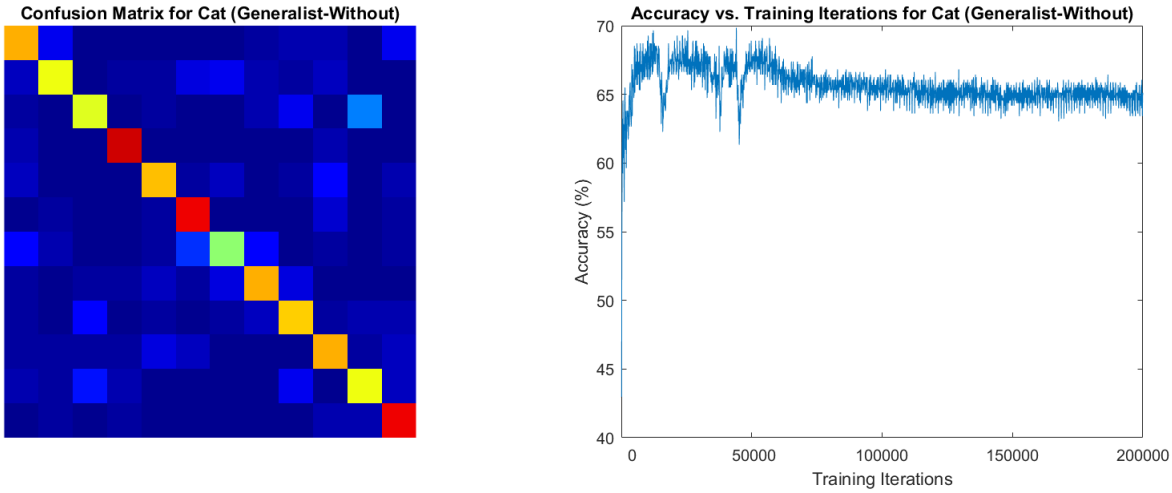

Figure 54. Results from the cat generalist-without network. Left: The confusion matrix generated by the best model for cat with initialization from the generalist superset network without cat data. Right: The accuracy on the test set over training.

Generalist Without Cats Initialization. Figure 54 shows the test set accuracy vs. training iterations with initialization from the more distant generalist without cats network. The highest overall accuracy achieved was 69.849% after 43800 training iterations. The top-1 error rate of this model was 50%. Compared to random initialization, the overall accuracy increased by 17.085% and the top-1 error rate was reduced by 16%.

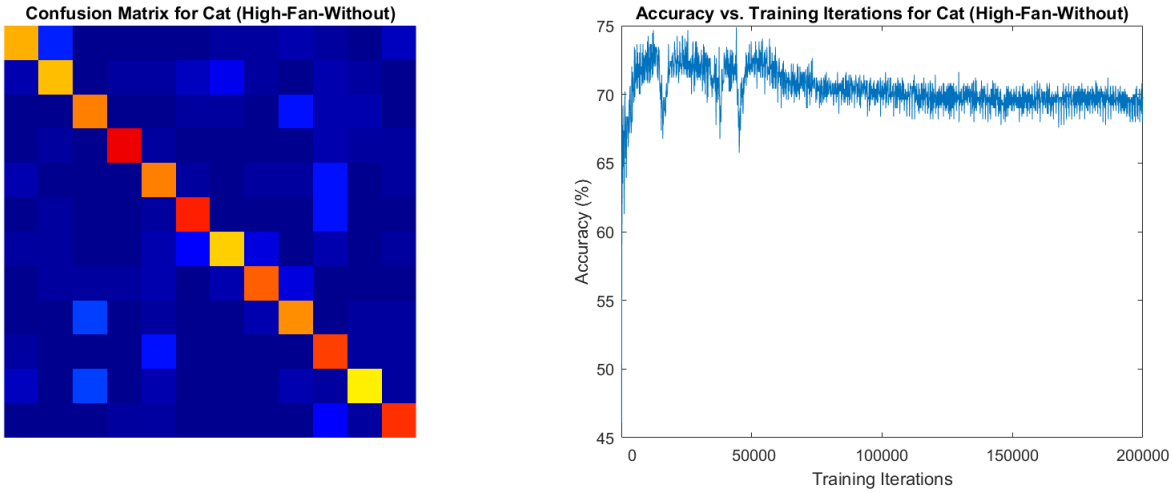

Figure 55. Results from the cat high-fan-without network. Left: The confusion matrix generated by the best model for cat with initialization from the high-fan superset network without cat data. Right: The accuracy on the test set over training.

High-Fan Without Cats Initialization. Figure 55 shows the test set accuracy vs. training iterations with initialization from the more distant high-fan without cats network. The highest overall accuracy achieved was 74874% after 28400 training iterations. The top-1 error rate of this model was 36%. Compared to random initialization, the overall accuracy increased by 22.11% and the top-1 error rate was reduced by 30%.

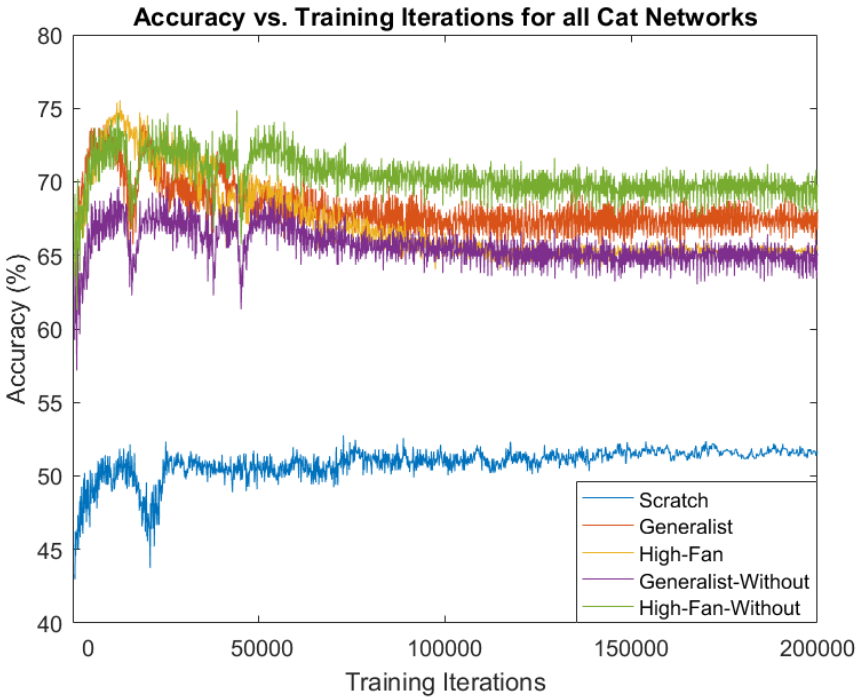

Figure 56. The performance of all cat networks on one figure.

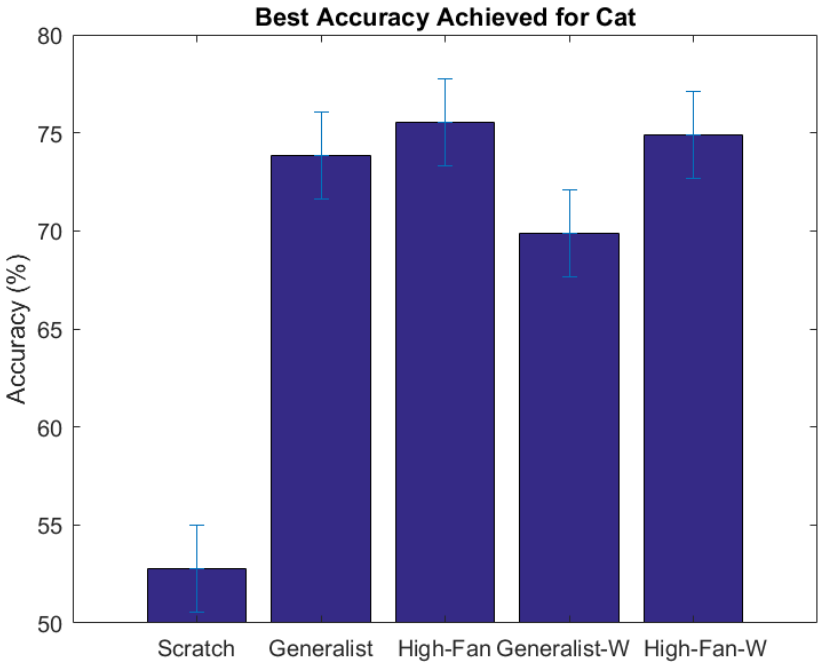

Figure 57. The best accuracy of all cat networks on one figure.

Figures 56 and 57 show the overall results for the cat dataset. In all cases transfer learning improved the performance over the random initialization. The best source was the high-fan superset network, which had the greatest increase in overall accuracy of 22.78%. It was also the best source for reducing the top-1 error rate. It had a top-1 error rate of 36% compared to the scratch initialization network's top-1 error rate of 66%. It also took least training iterations at 12600 iterations. It is worth noting that all networks took substantially less training than the scratch network (less than half in three out of four case). The class responsible for the top-1 error rate was the same for the two generalist networks and the scratch network. The two high-fan networks each had different classes for the top-1 error rate.

This dataset requires a difference of 2.22% accuracy in order to be considered statistically significant with 95% confidence. The generalist, high-fan, and high-fan-without networks are not statistically significant in their difference with each other. The general-without network and generalist network are also not statistically significant in their difference. The generalist-without and high-fan-without networks are also not different to a statistically significant degree.

4.3.5 Flowers.

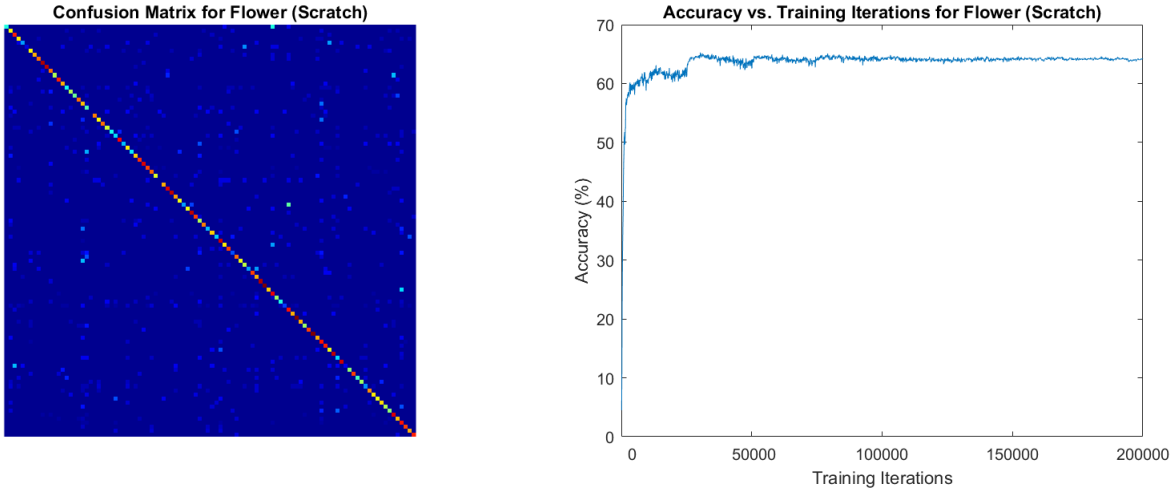

Figure 58. Results from the flower scratch network. Left: The confusion matrix generated by the best model for flower with random initialization. Right: The accuracy on the test set over training.

Scratch Initialization. Figure 58 shows the test set accuracy vs. training iterations with random initialization. The highest overall accuracy achieved was 65.256% after 38600 training iterations. The top-1 error rate of this model was 100%.

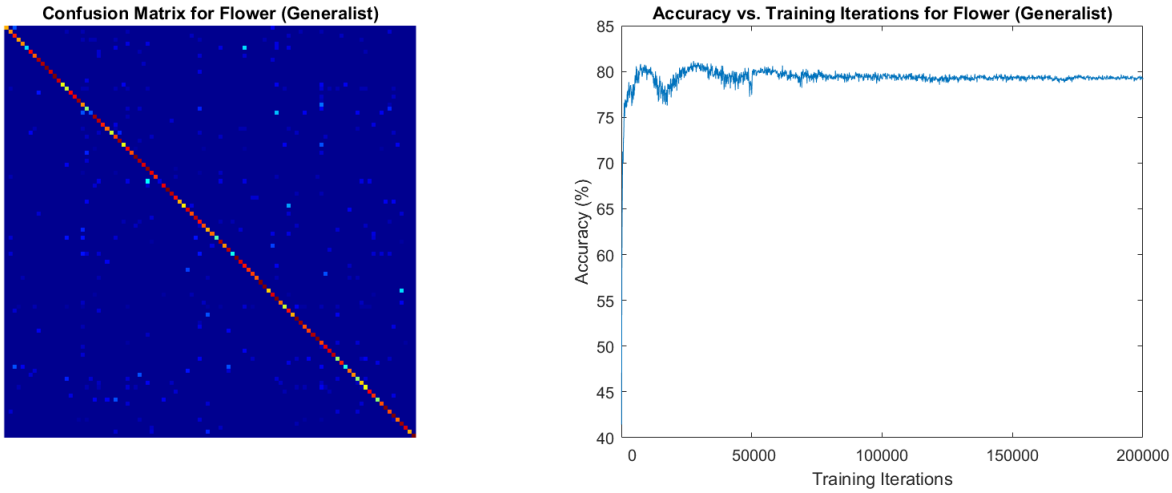

Figure 59. Results from the flower generalist network. Left: The confusion matrix generated by the best model for flower with initialization from the generalist superset network. Right: The accuracy on the test set over training.

Generalist Initialization. Figure 59 shows the test set accuracy vs. training iterations with initialization from the generalist network. The highest overall accuracy achieved was 81.135% after 27700 training iterations. The top-1 error rate of this model was 90.91%. Compared to random initialization, the overall accuracy increased by 15.879% and the top-1 error rate was reduced by 9.09%.

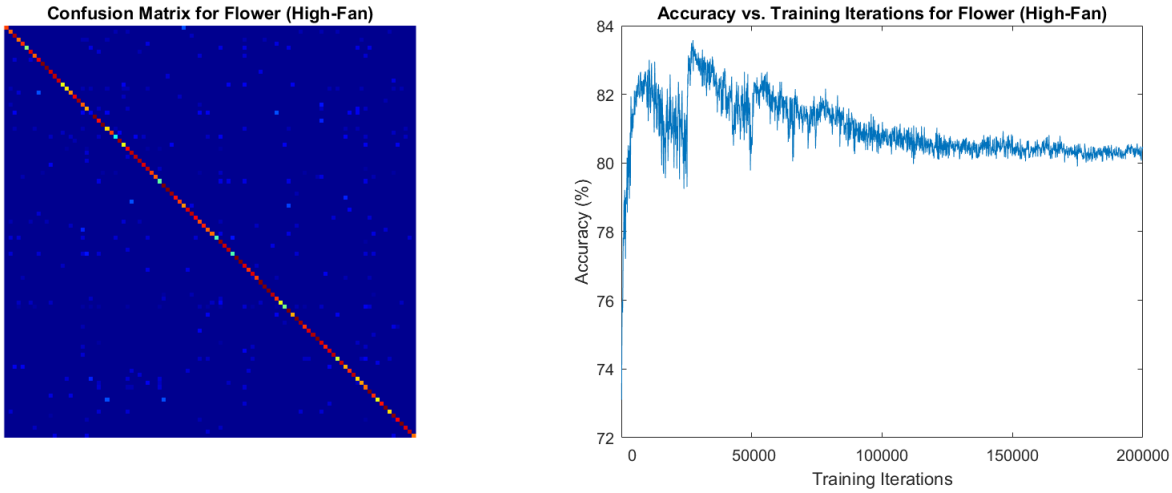

Figure 60. Results from the flower high-fan network. Left: The confusion matrix generated by the best model for flower with initialization from the high-fan superset network. Right: The accuracy on the test set over training.

High-Fan Initialization. Figure 60 shows the test set accuracy vs. training iterations with initialization from the high-fan superset network. The highest overall accuracy achieved was 83.579% after 36000 training iterations. The top-1 error rate of this model was 90%. Compared to random initialization, the overall accuracy increased by 18.323% and the top-1 error rate was reduced by 10%.

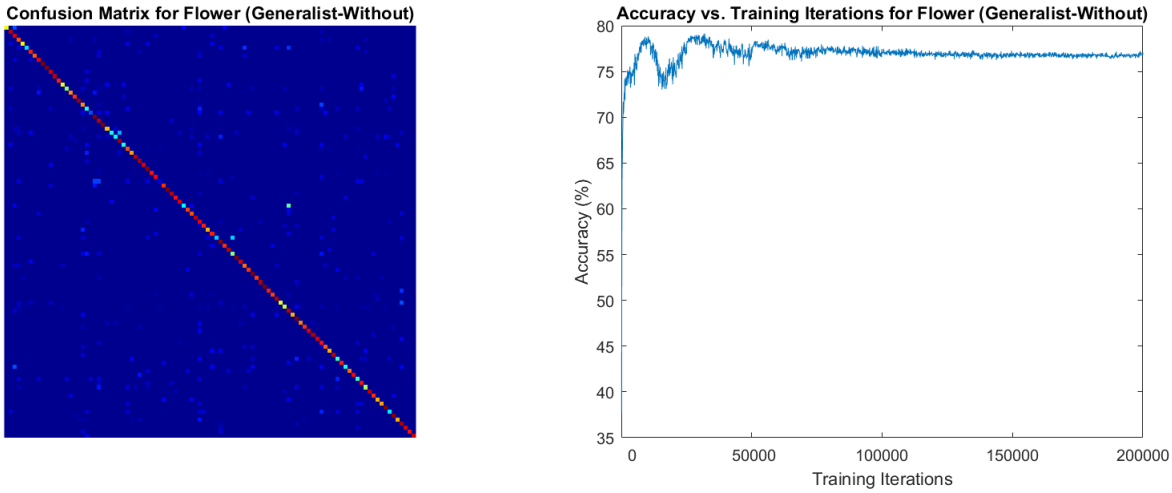

Figure 61. Results from the flower generalist-without network. Left: The confusion matrix generated by the best model for flower with initialization from the generalist superset network without flower data. Right: The accuracy on the test set over training.

Generalist Without Flowers Initialization. Figure 61 shows the test set accuracy vs. training iterations with initialization form the generalist without flowers network. The highest overall accuracy achieved was 79.144% after 25800 training iterations. The top-1 error rate of this model was 90.91%. Compared to random initialization, the overall accuracy increased by 13.888% and the top-1 error rate was reduced by 9.09%.

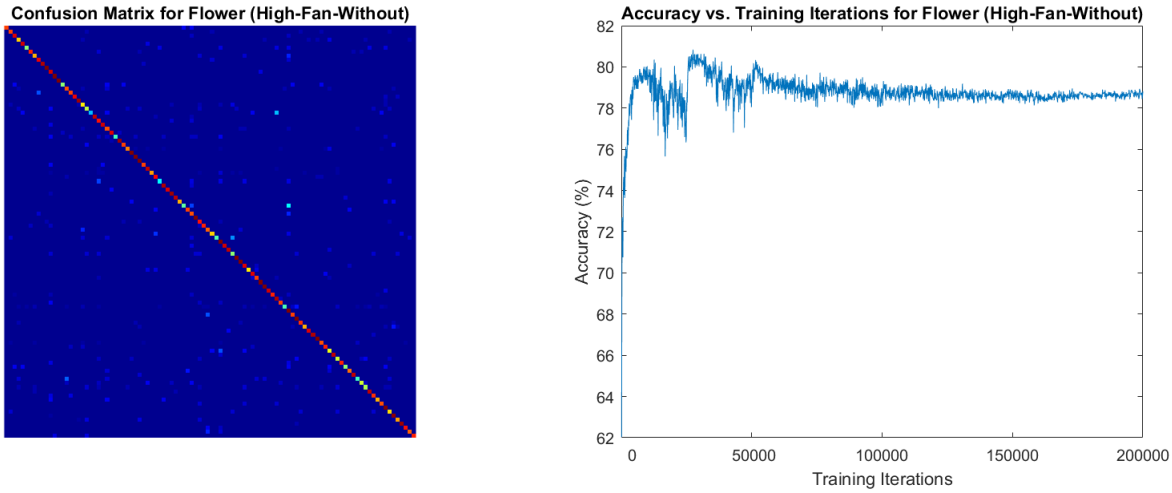

Figure 62. Results from the flower high-fan-without network. Left: The confusion matrix generated by the best model for flower with initialization from the high-fan superset network without flower data. Right: The accuracy on the test set over training.

High-Fan Without Flowers Initialization. Figure 62 shows the test set accuracy vs. training iterations with initialization from the high-fan network without flowers. The highest overall accuracy achieved was 80.836% after 34000 training iterations. The top-1 error rate of this model was 70%. Compared to random initialization, the overall accuracy increased by 15.58% and the top-1 error rate was reduced by 30%.

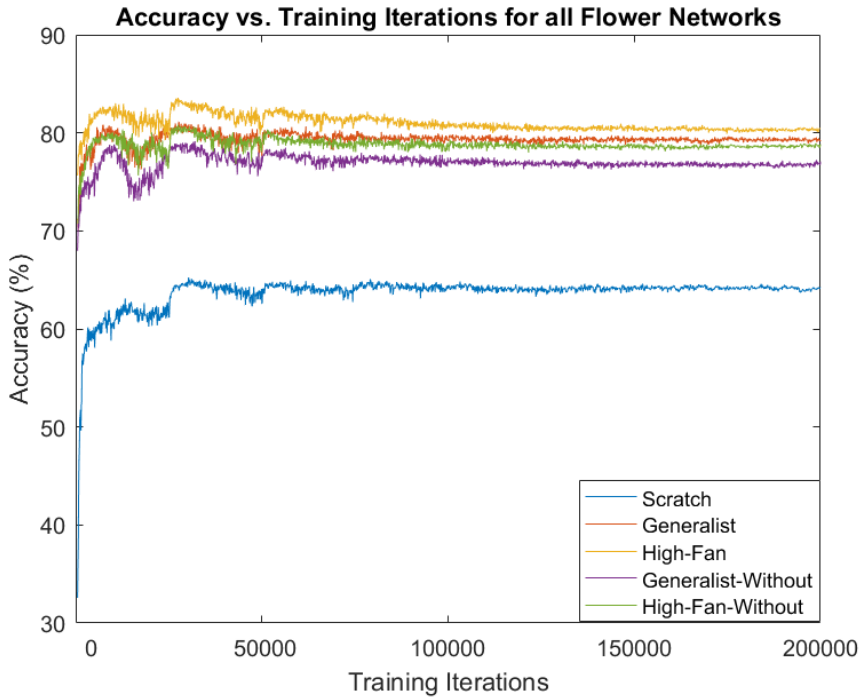

Figure 63. The performance of all flower networks on one figure.

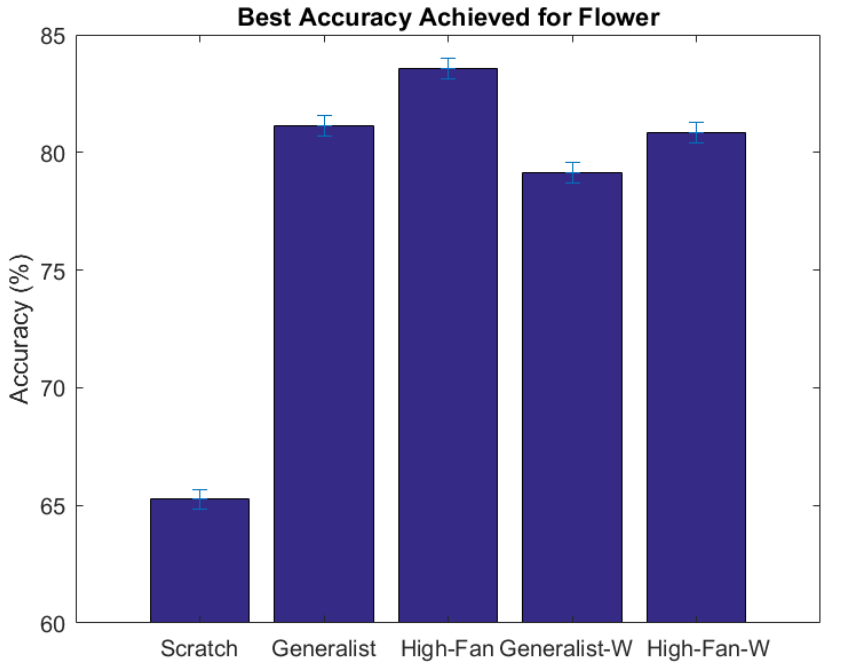

Figure 64. The best accuracy of all flower networks on one figure.

Figures 63 and 64 show the overall results for the Flower dataset. In all cases but one, transfer learning improved the performance over the random initialization. The best source was the high-fan superset network, which had the greatest increase in overall accuracy to 83.579%. The best source for reducing the top-1 error rate was the more distant high-fan without flowers network. It had a top-1 error rate of 70% compared to the scratch initialization network's top-1 error rate of 100%. The scratch network also took the longest to converge at 38600 training iterations. All other transferred networks took less training to converge than the scratch network. The class responsible for the top-1 error rate was the same for the scratch and high-fan networks. A different class was the also the source of the top-1 error for the generalist networks.

This dataset requires a difference of .77% accuracy in order to be considered statistically significant with 95% confidence. The high-fan-without network is not different enough to achieve statistical significance from either the generalist-without or the generalist network.

4.3.6 Birds.

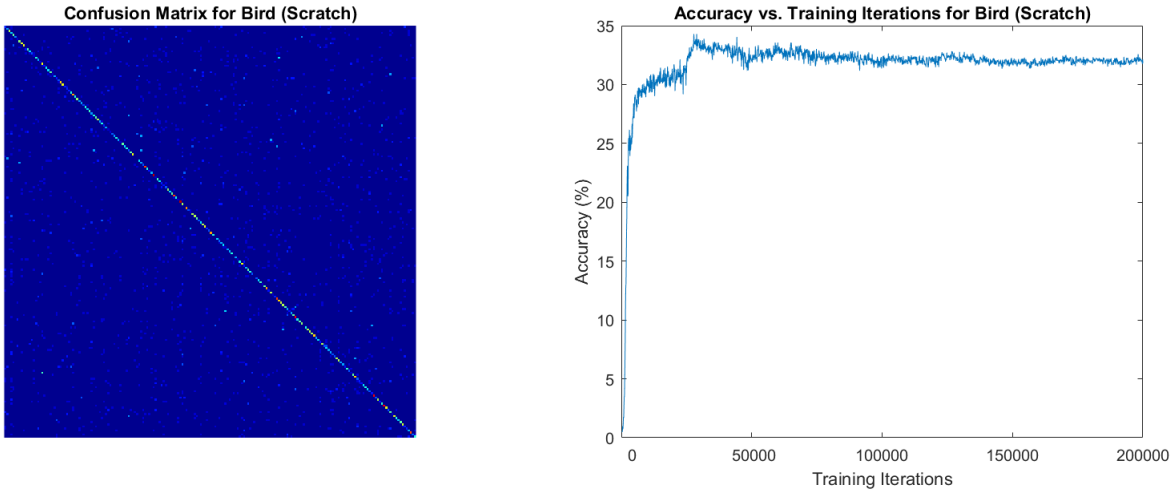

Figure 65. Results from the bird scratch network. Left: The confusion matrix generated by the best model for bird with random initialization. Right: The accuracy on the test set over training.

Scratch Initialization. Figure 65 shows the test set accuracy vs. training iterations with random initialization. The highest overall accuracy achieved was 34.294% after 28000 training iterations. The top-1 error rate of this model was 100%.

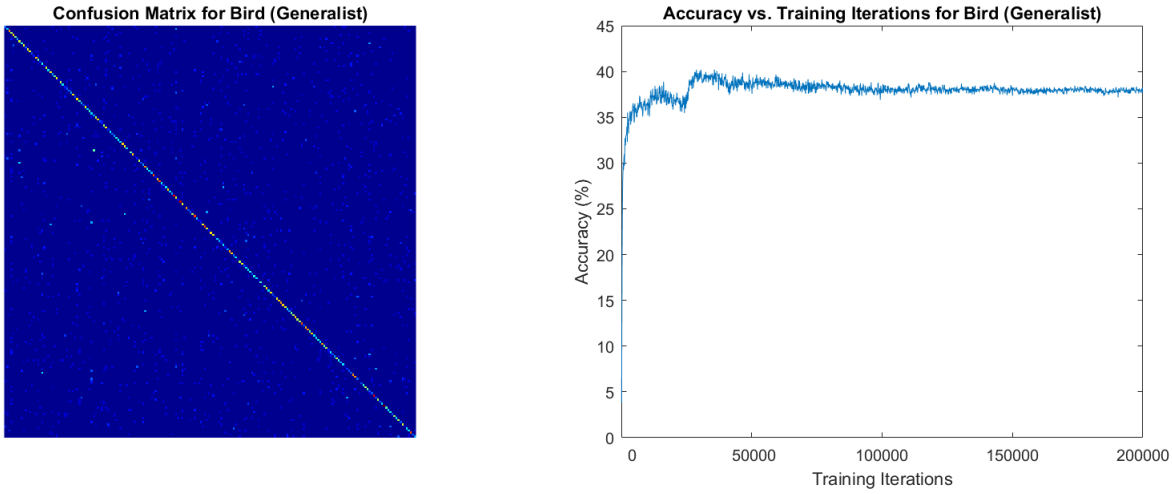

Figure 66. Results from the bird generalist network. Left: The confusion matrix generated by the best model for bird with initialization from the generalist superset network. Right: The accuracy on the test set over training.

Generalist Initialization. Figure 66 shows the test set accuracy vs. training iterations with initialization from the generalist network. The highest overall accuracy achieved was 40.192% after 30000 training iterations. The top-1 error rate of this model was 100%. Compared to random initialization, the overall accuracy increased by 5.898% and the top-1 error rate was unchanged.

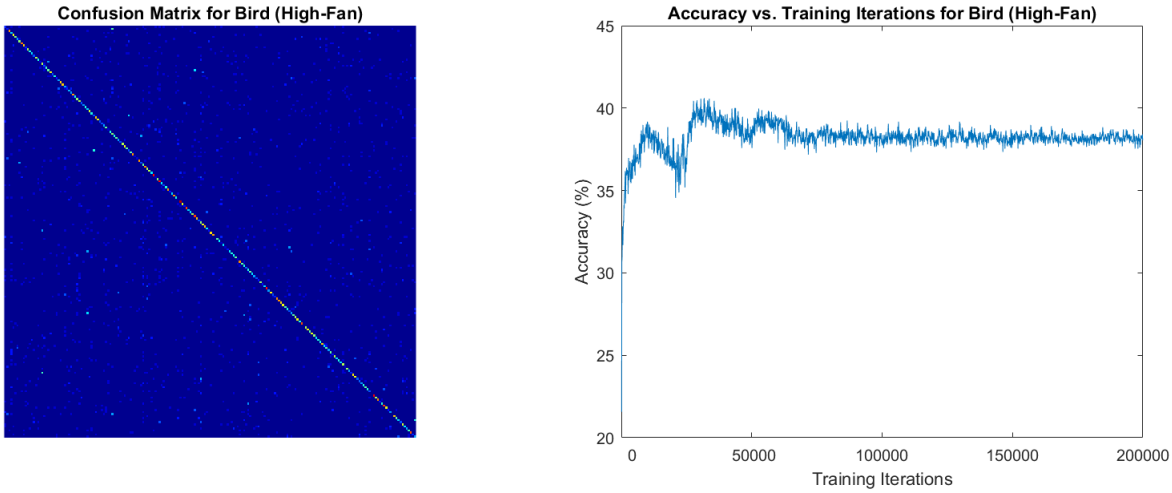

Figure 67. Results from the bird high-fan network. Left: The confusion matrix generated by the best model for bird with initialization from the high-fan superset network. Right: The accuracy on the test set over training.

High-Fan Initialization. Figure 67 shows the test set accuracy vs. training iterations with initialization from the high-fan network. The highest overall accuracy achieved was 40.604% after 30500 training iterations. The top-1 error rate of this model was 100%. Compared to random initialization, the overall accuracy increased by 6.31% and the top-1 error rate was unchanged.

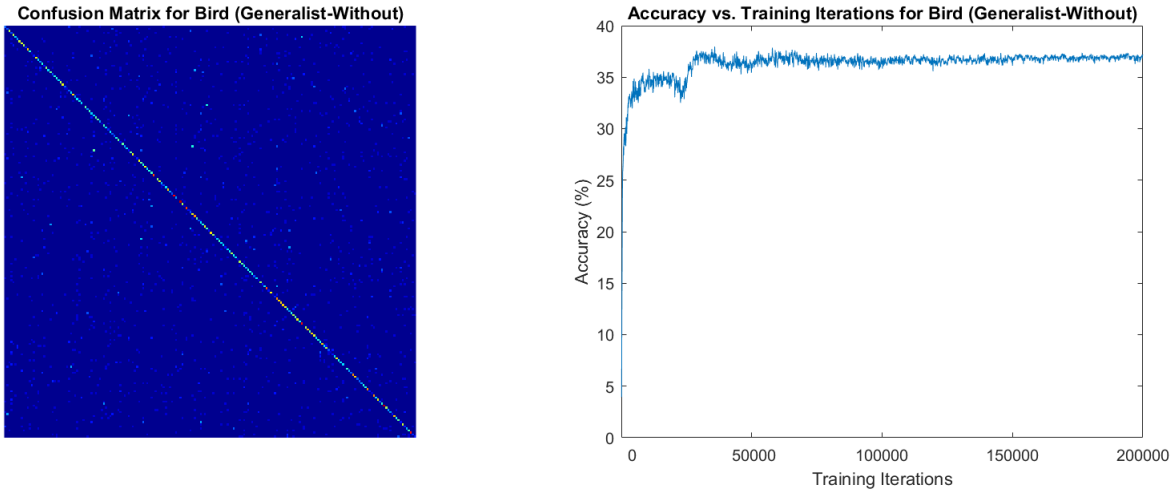

Figure 68. Results from the bird generalist-without network. Left: The confusion matrix generated by the best model for bird with initialization from the generalist superset network without bird data. Right: The accuracy on the test set over training.

Generalist Without Birds Initialization. Figure 68 shows the test set accuracy vs. training iterations with initialization from a generalist network trained on the entire dataset except for birds. The highest overall accuracy achieved was 37.963% after 35800 training iterations. The top-1 error rate of this model was 100%. Compared to random initialization, the overall accuracy increased by 3.669% and the top-1 error rate was unchanged.

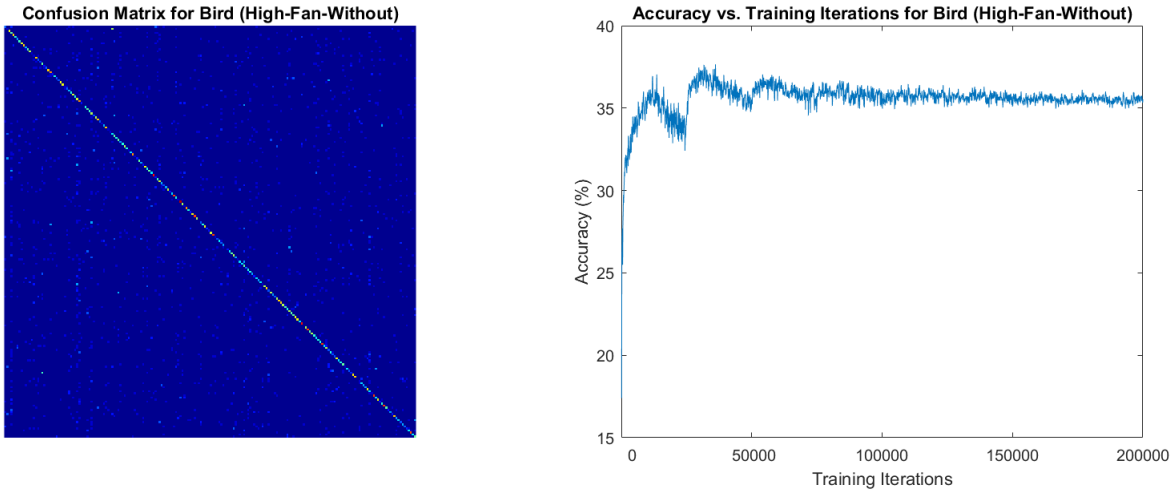

Figure 69. Results from the bird high-fan-without network. Left: The confusion matrix generated by the best model for bird with initialization from the high-fan superset network without bird data. Right: The accuracy on the test set over training.

High-Fan Without Birds Initialization. Figure 69 shows the test set accuracy vs. training iterations with initialization from a high-fan network trained on the entire dataset except for birds. The highest overall accuracy achieved was 37.654% after 36200 training iterations. The top-1 error rate of this model was 100%. Compared to random initialization, the overall accuracy increased by 3.36% and the top-1 error rate was unchanged.

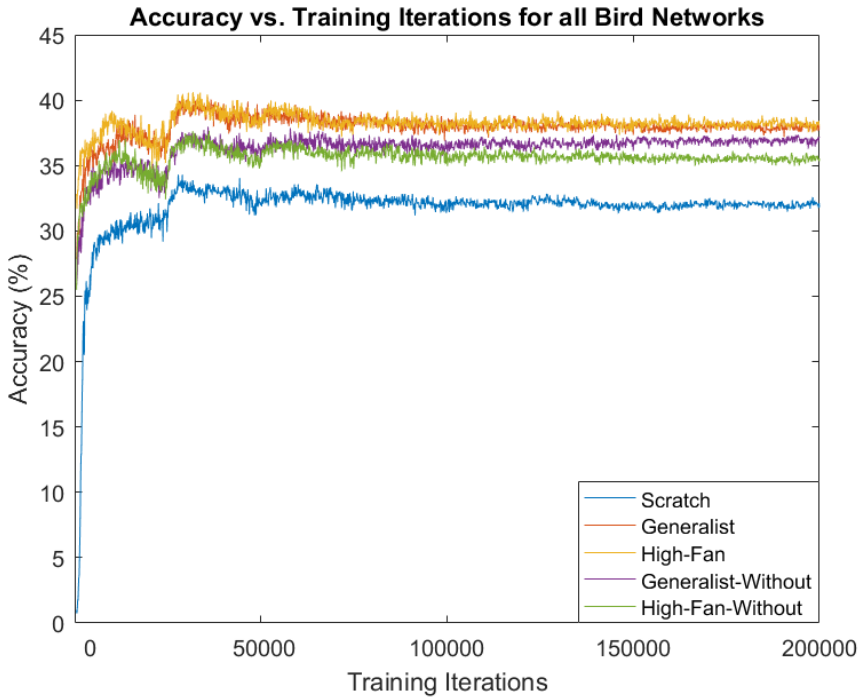

Figure 70. The performance of all bird networks on one figure.

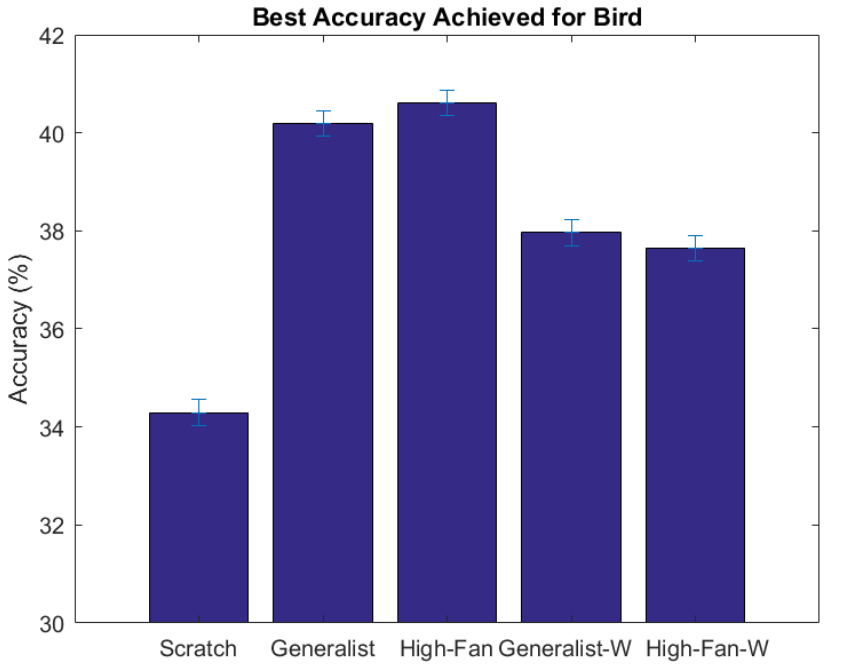

Figure 71. The best accuracy of all bird networks on one figure.

Figures 70 and 71 show the overall results for the bird dataset. In all cases transfer learning improved the performance over the random initialization. The best source was the high-fan superset network, which had the greatest increase in overall accuracy to 6.31%. All networks had a top-1 error of 100%. They also had multiple classes with a 100% error rate, although the fine-tuned networks had fewer of these classes than the scratch network. This is partly because some classes had as few as 10 images in the test set. All fine-tuned networks also took more training iterations than the scratch network, although not a substantial amount more. It is worth noting that all networks took substantially less training than the scratch network (less than half the train three out of four case).

For this dataset, statistical significance with a confidence of 95% requires a difference greater than .31%. The generalist-without and high-fan-without models are within this margin for each other. This is also true of the generalist and high-fan networks.

4.3.7 Planes.

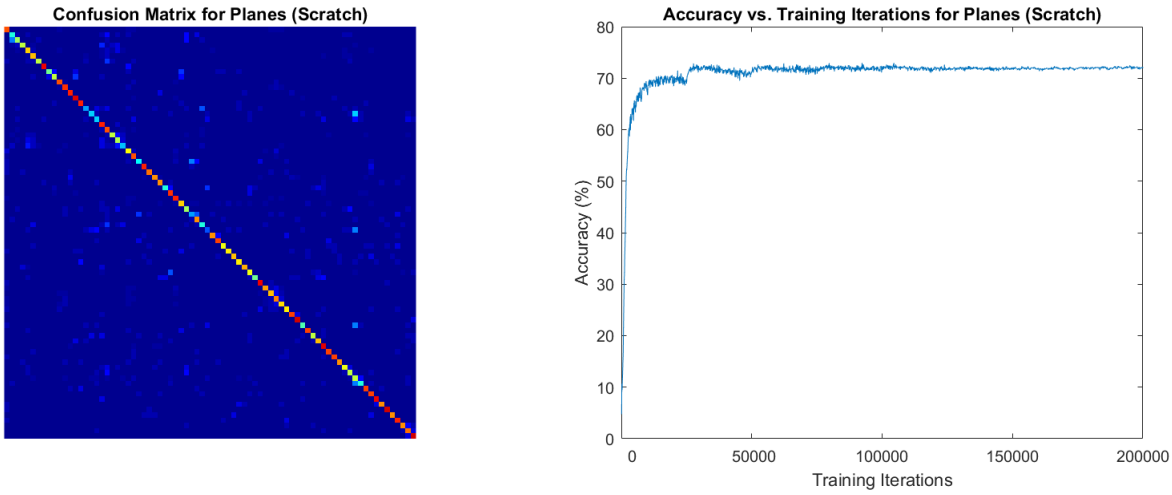

Figure 72. Results from the plane scratch network. Left: The confusion matrix generated by the best model for plane with random initialization. Right: The accuracy on the test set over training.

Scratch Initialization. Figure 72 shows the test set accuracy vs. training iterations with random initialization. The highest overall accuracy achieved was 70.405% after 92300 training iterations. The top-1 error rate of this model was 80%.

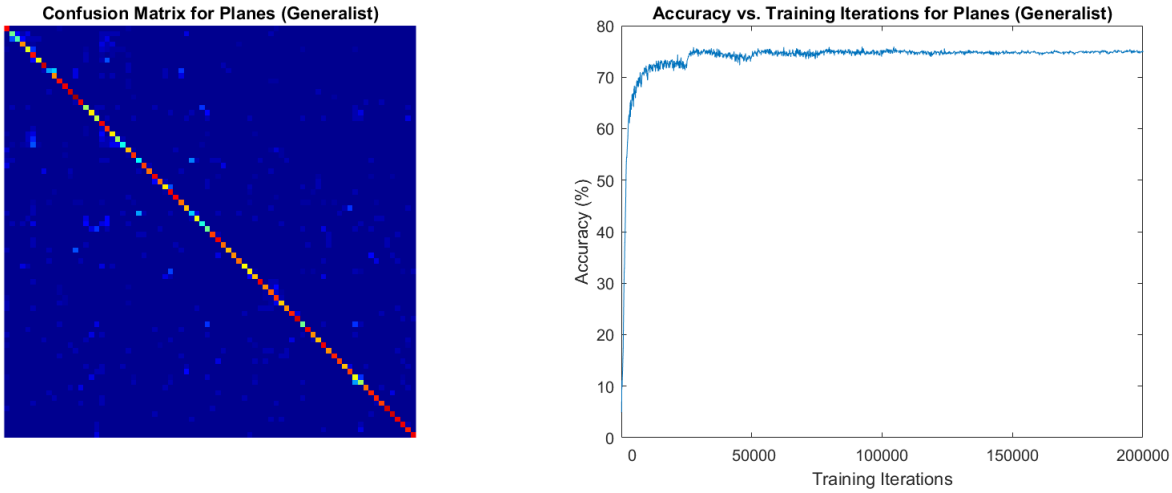

Figure 73. Results from the plane generalist network. Left: The confusion matrix generated by the best model for plane with initialization from the generalist superset network. Right: The accuracy on the test set over training.

Generalist Initialization. Figure 73 shows the test set accuracy vs. training iterations with initialization from the generalist network. The highest overall accuracy achieved was 75.933% after 30600 training iterations. The top-1 error rate of this model was 72%. Compared to random initialization, the overall accuracy increased by 5.528% and the top-1 error rate was reduced by 8%.

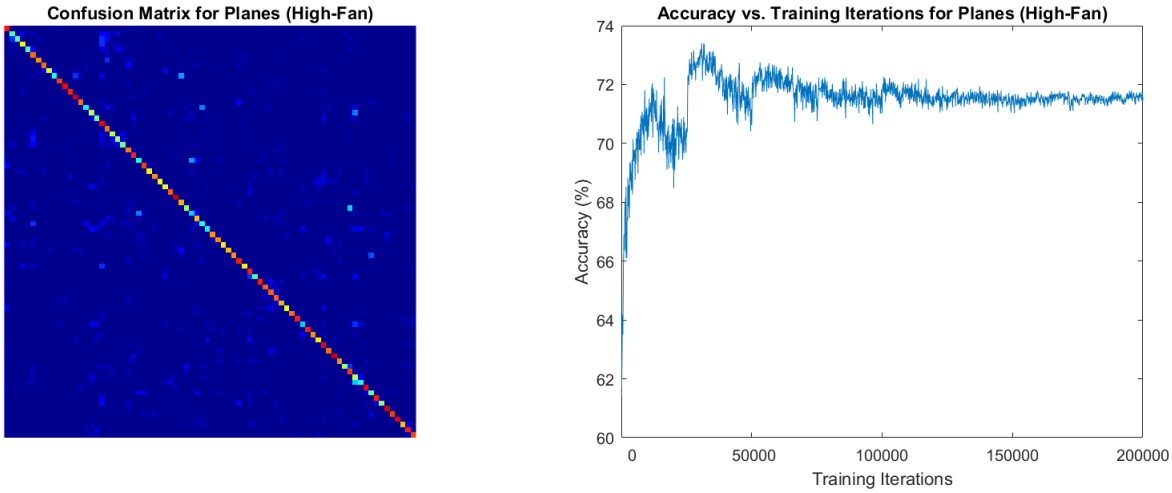

Figure 74. Results from the plane high-fan network. Left: The confusion matrix generated by the best model for plane with initialization from the high-fan superset network. Right: The accuracy on the test set over training.

High-Fan Initialization. Figure 74 shows the test set accuracy vs. training iterations with initialization from the high-fan network. The highest overall accuracy achieved was 73.416% after 30800 training iterations. The top-1 error rate of this model was 68%. Compared to random initialization, the overall accuracy increased by 3.011% and the top-1 error rate was reduced by 12%.

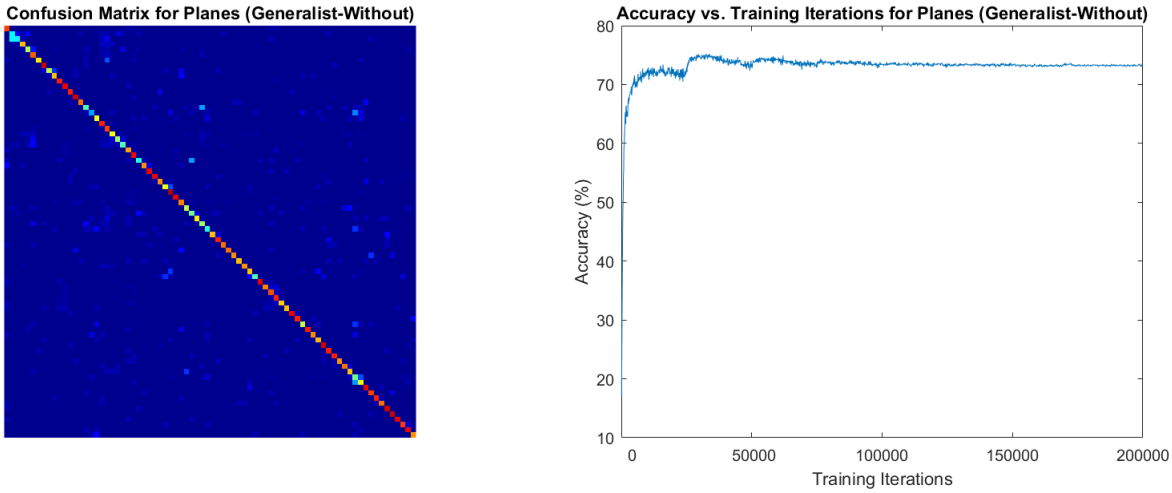

Figure 75. Results from the plane generalist-without network. Left: The confusion matrix generated by the best model for plane with initialization from the generalist superset network without plane data. Right: The accuracy on the test set over training.

Generalist Without Planes Initialization. Figure 75 shows the test set accuracy vs. training iterations with initialization from a generalist network without planes. The highest overall accuracy achieved was 75.214% after 32500 training iterations. The top-1 error rate of this model was 72%. Compared to random initialization, the overall accuracy increased by 4.809% and the top-1 error rate was reduced by 8%.

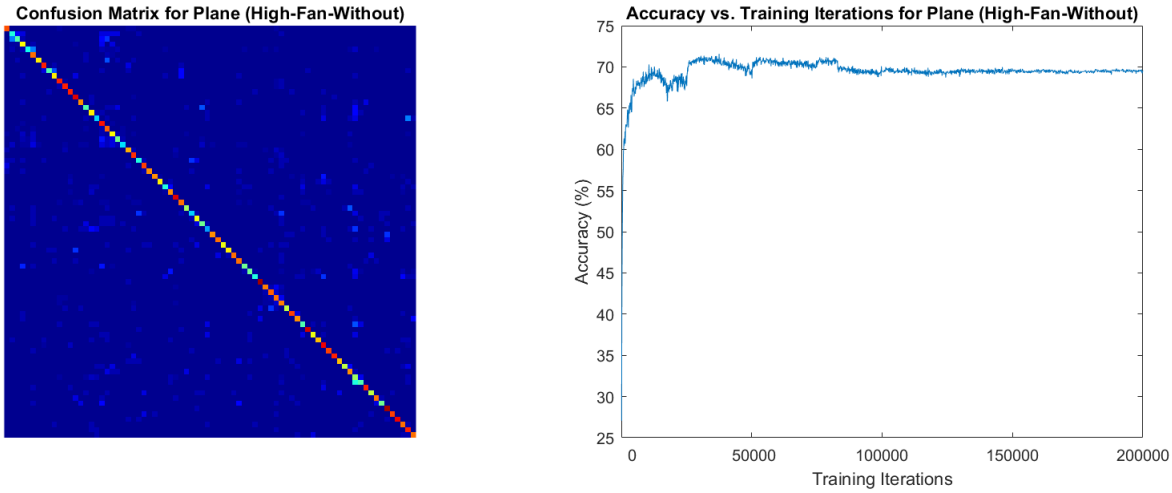

Figure 76. Results from the plane high-fan-without network. Left: The confusion matrix generated by the best model for plane with initialization from the high-fan superset network without plane data. Right: The accuracy on the test set over training.

High-Fan Without Planes Initialization. Figure 76 shows the test set accuracy vs. training iterations with initialization from the high-fan without planes network. The highest overall accuracy achieved was 71.573% after 80100 training iterations. The top-1 error rate of this model was 72%. Compared to random initialization, the overall accuracy was increased by 1.168% and the top-1 error rate was reduced by 8%.

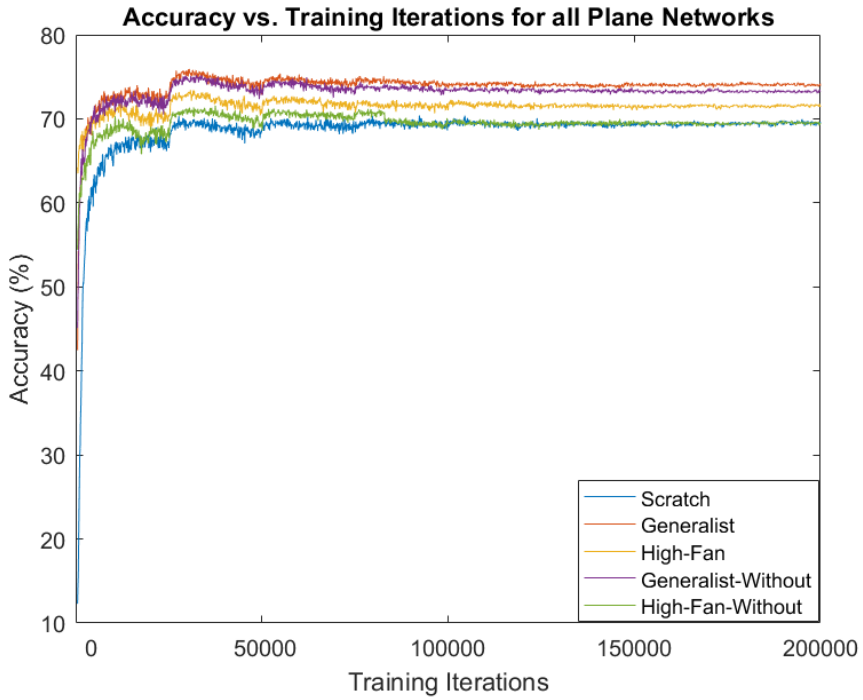

Figure 77. The performance of all plane networks on one figure.

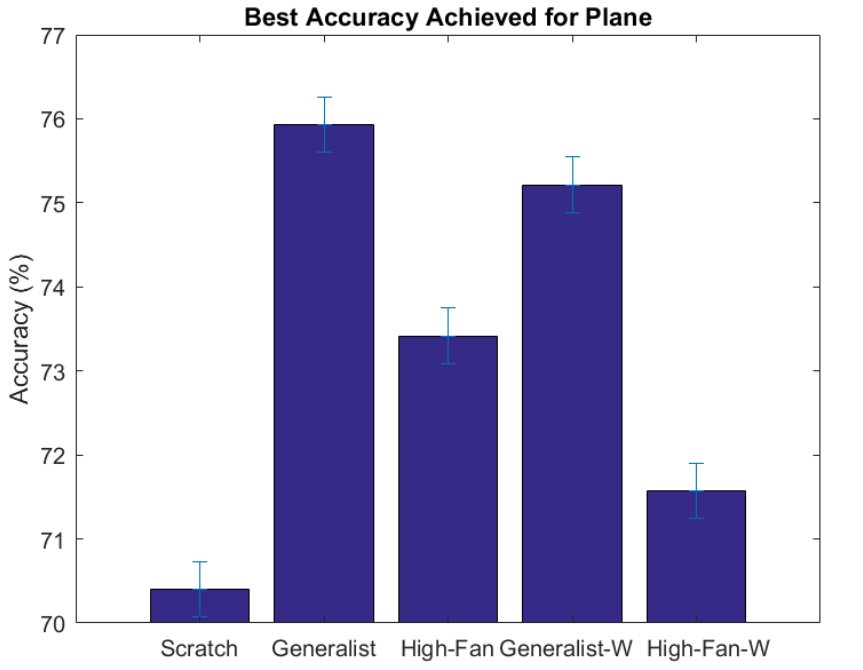

Figure 78. The best accuracy of all plane networks on one figure.

Figures 77 and 78 show the overall results for the plane dataset. In all cases transfer learning improved the performance over the random initialization. The best source was the generalist superset network, which had the greatest increase in overall accuracy of 5.528%. It also took least training iterations at 30600 iterations. The best source for reducing the top-1 error rate was the high-fan superset network. It had a top-1 error rate of 68% compared to the scratch initialization network's top-1 error rate of 80%. It is worth noting that all networks took substantially less training than the scratch network with the exception of the high-fan without planes superset source, which only improved the top-1 error rate. The class responsible for the top-1 error rate was the different each network.

A difference of .61% accuracy is considered statistically significant with 95% confidence. The scratch and high-fan-without networks do not meet this threshold. The generalist and generalist-without also do not meet this threshold.

4.3.8 Overall Results.

In all three cases, the top-performing source is the high-fan network. In each case, the generalist network had one fewer win than the high-fan network. The more distant generalist-without and high-fan-without networks always had equal results and always had fewer wins than the high-fan network.

Table 10 shows the results and figure 79 provides a visual representation of the same data. When viewed this way, the high-fan network achieves the best overall result with a mean error reduction rate of 28.5%. The generalist network actually

Table 9. A tabulation of the "winners" from the third experiment.

	S	G	HF	GW	HFW
Wins	0	2	3	1	1
Statistically Significant Wins	0	0	1	0	0
Within Margin of Winning	1	5	6	3	3

achieves the lowest mean error reduction rate of 22.9%. However, this is largely attributable to its underperformance on the signs dataset, which did not have any statistically significant results. When removed, the mean error reduction rate increases to 23.6% while the generalist-without and high-fan-without networks fall to 20.0% and 19.8%, respectively. The high-fan network still retains the highest rate at 24.2%, however.

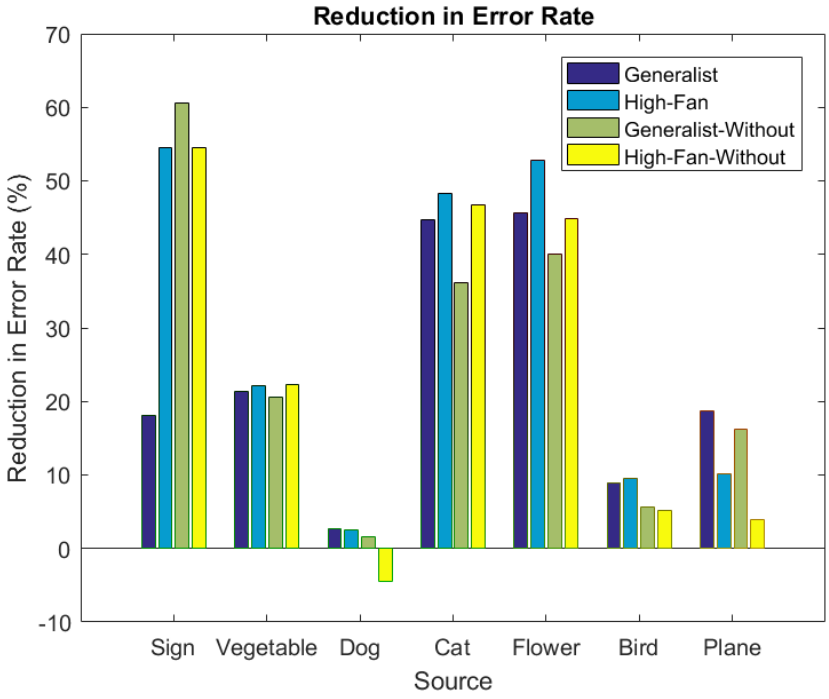

Figure 79. The reduction in error rate for all datasets using the scratch network as the baseline performance.

Table 10. The reduction in error rate for each dataset

	G	HF	GW	HFW
Sign	18.1657	54.5414	60.6114	54.5414
Vegetable	21.2968	22.0706	20.5247	22.3421
Dog	2.7095	2.4938	1.5842	-4.4402
Cat	44.6799	48.2259	36.1694	46.8075
Flower	45.7029	52.7343	39.9724	44.8423
Bird	8.9763	9.6034	5.5840	5.1137
Plane	18.6788	10.1740	16.2494	3.9466
Mean	22.871	28.5491	25.8136	24.7362

4.4 Comparing Ensemble Networks

This experiment compares the results of a an ensemble of randomly initialized networks to an ensemble of networks trained with transfer learning. Three additional scratch networks to use along with the scratch network from the planes dataset above. They individually achieved an overall accuracy of 68.247, 69.438, 69.775. The ensemble of fine-tuned networks is made up of the four networks from the planes dataset above.

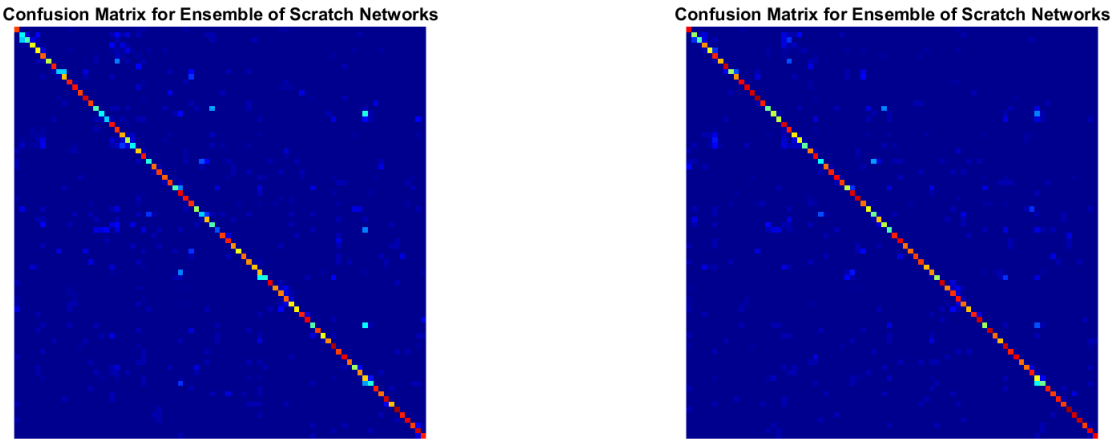

Figure 80. The confusion matrices for the ensemble networks. Left: The confusion matrix generated by the ensemble of scratch networks. Right: The confusion matrix generated by the ensemble of fine-tuned networks.

Figure 80 shows the confusion matrix for the two ensembles. The ensemble of scratch networks achieved an overall accuracy of 73.146 and a top-1 error rate of 80%. The ensemble of fine-tuned networks achieved an overall accuracy of 78.719% and a top-1 error rate of 68%. Both ensembles outperformed any of the individual networks in the ensemble. Given a .61% margin for a 95% confidence interval, the scratch ensemble still underperforms the best fine-tuned network (generalist at 75.933% accuracy) to a statistically significant degree. Figure 84 shows these results.

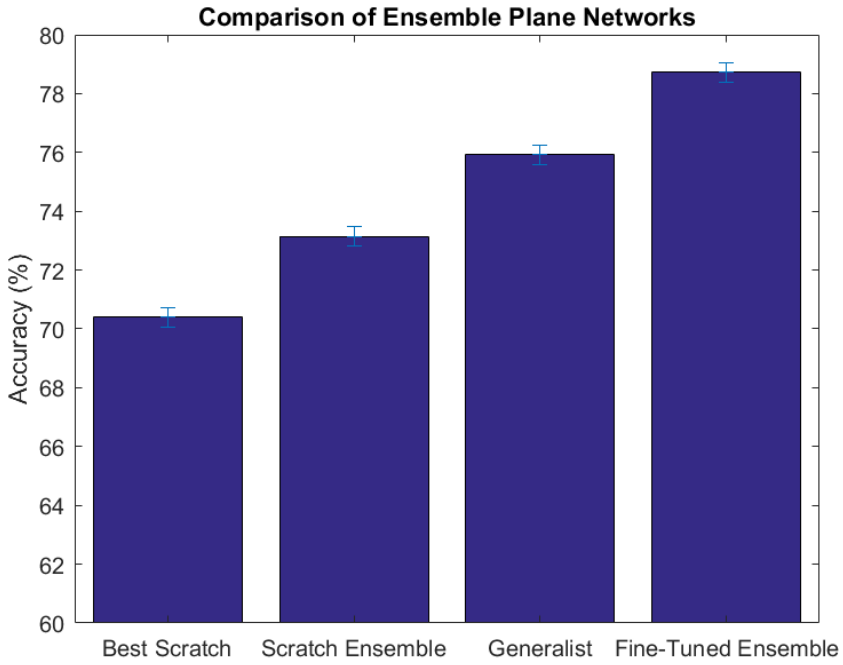

Figure 81. Comparison of the ensemble networks and the top performing scratch and fine-tuned networks

This concludes the results generated by all of the experiments.

V. Conclusions

5.1 Discussion of Results

This thesis presented several experiments regarding parameter fine-tuning. The previous chapter reported the raw results from these experiments. This chapter will discuss the results and explain what conclusions may or may not be drawn from them. Some of the most relevant charts are reproduced here for ease of reference.

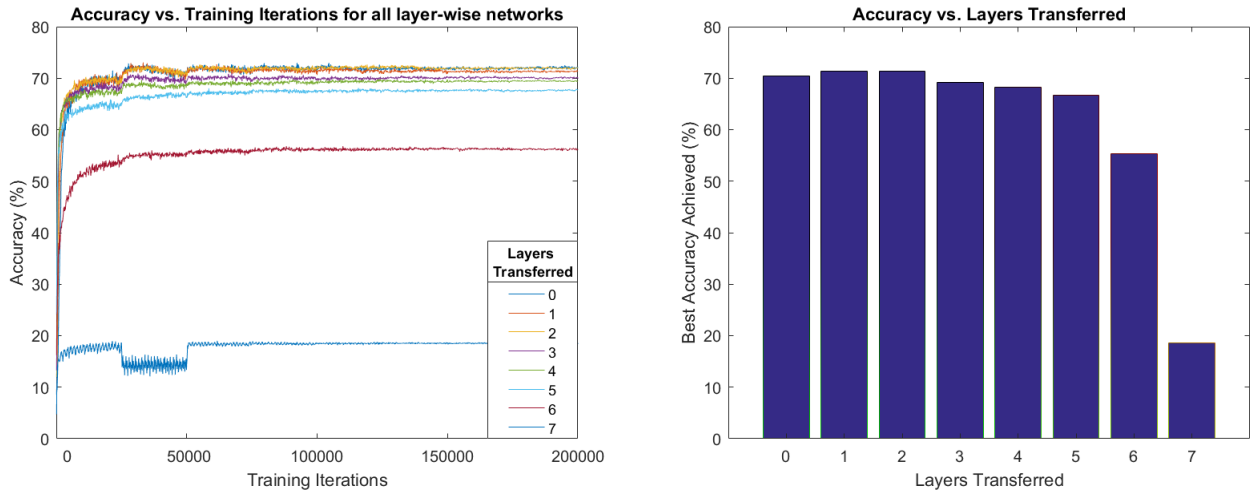

Figure 82. The results of all the layer-wise networks. Left: The performance of all the networks on one figure. Right: The best accuracy of each network.

Determining Specificity of Features. This experiment recreated some of the work done by Yosinski *et al* on a different dataset [52]. The results (shown concisely in figure 82) suggest that the first couple of layers from a ConvNet are universal. As depth increases, layers become less transferable because they are specific to the dataset on which they were trained. This demonstrates the need to continue learning in the previous transferred layers. The results from the first few layers improved over random initialization, but not to a statistically significant degree. However, they

demonstrated the same trend found in Yosinki's work, suggesting that this trend is universal.

Optimizing Learn Rate. The previous experiment shows that learning is necessary in all layers. This experiment seeks to find the optimal learn rate in those layers. The results show that as the learn rate is increased in the transferred layers to the point where it matches the learn rate in the randomly initialized layers, the overall accuracy of the ConvNet improves. This suggests, contrary to Girshick *et al* that "clobbering learning" should be avoided [22]. Rather than a technique that transfers knowledge that should be preserved, parameter fine-tuning should be thought of as a superior initialization compared to Gaussian noise.

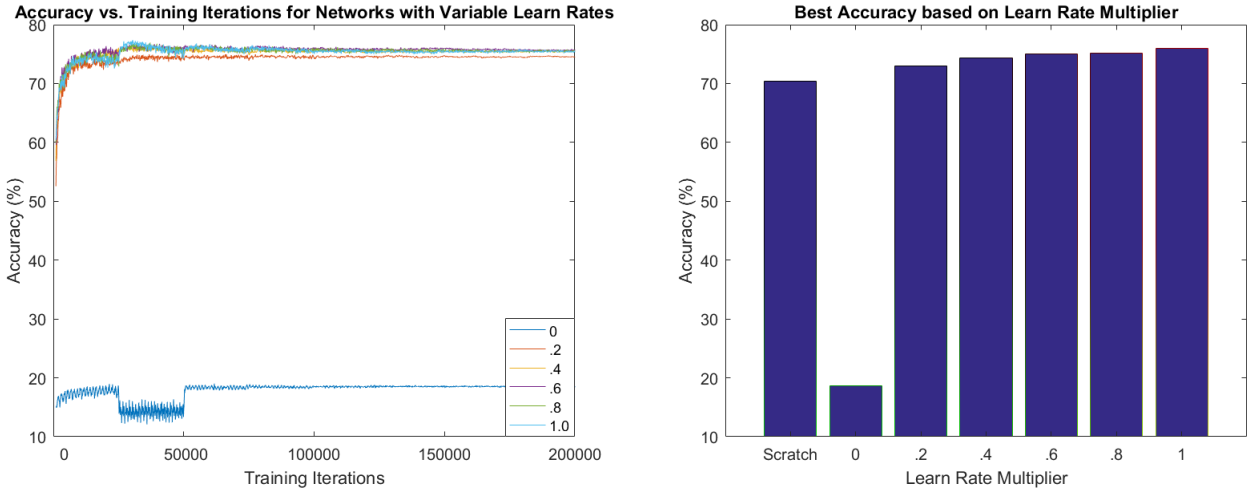

Figure 83. The combined results of all the networks with different learning rate multipliers. Left: The performance of all the networks on one figure based on their learn rate multiplier. Right: The best accuracy of each network.

Measuring Effect of Source Task on Target Task. Having established the optimal learning rate, the next goal is to determine the best source task. The results from this experiment are mixed. The results clearly demonstrate that parameter fine-tuning almost always outperforms random initialization. In 27/28 cases,

this is true. When statistical significance is taken into account, this number falls to 22/28; however, 4 of those cases come from the signs dataset where 100% accuracy is still not statistically significant from the baseline set by the scratch network. Also noteworthy is the fact that initial performance is much better on fine-tuned networks compared to scratch networks. After a few hundred training iterations, fine-tuned networks generally achieve an accuracy not far from the best accuracy achieved by the model. The scratch network, however, is still not much better than a random classifier. The amount of training to achieve the optimal model is not necessarily reduced, but the amount of training to achieve a reasonable model is greatly reduced. This suggests fine-tuning has value in applications which may call for rapid prototyping.

In general, it seems that the high-fan network transfers the best. It was the top performer compared to the other networks in terms of both wins and mean error reduction rate. However, the generalist network comes close behind. The more distant networks trained without that particular subset of data almost always either mirror the performance of that closer network or underperform it. This is especially apparent when looking at figure 79.

The amount of training data in the target task seems to have little effect on the performance of fine-tuning. It was thought that the more data in the target dataset, the less useful fine-tuning would be. This appears to be the case for the dog dataset, which has the most training images available and shows little gain over random initialization. However, the bird dataset showed little gain over random initialization and had the least amount of training data per class available.

Table 11. A tabulation of the "winners" from the third experiment.

	S	G	HF	GW	HFW
Wins	0	2	3	1	1
Statistically Significant Wins	0	0	1	0	0
Within Margin of Winning	1	5	6	3	3

Comparing Ensemble Networks. Ensembles of ConvNets have long been known to outperform individual ConvNets [46][48]. Because of this, it was suspected that an ensemble of fine-tuned ConvNets would outperform a similar ensemble of randomly initialized ConvNets. This turned out to be the case. Perhaps more noteworthy is that the ensemble of randomly initialized ConvNets either underperformed or matched the performance of the individual fine-tuned networks. This suggests that fine-tuning is as effective or more effective than training ensembles alone.

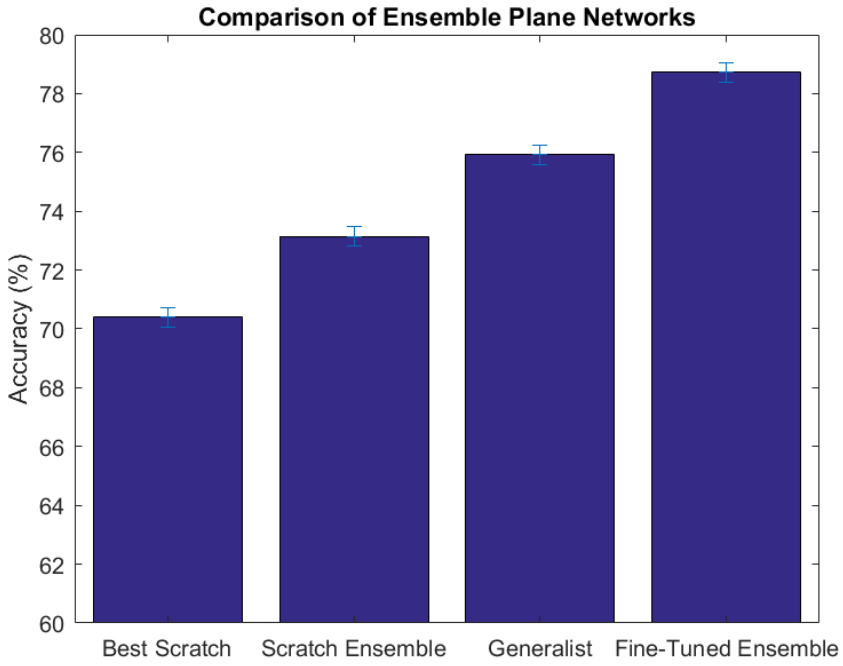

Figure 84. Comparison of the ensemble networks and the top performing scratch and fine-tuned networks

5.2 Future Work

This thesis presented a framework that relies on a dataset that can be divided in multiple ways. After training a network on various configurations of the dataset, it is then applied to a new dataset. The results of the different networks on the new dataset are directly comparable. Using this basic experimental framework, there are

a number of ways to extend this work. This section presents some ideas.

Exhaustive Testing of Dataset. Only experiment three was exhaustively tested against the entire of the dataset. The other three experiments were only tested on the planes subset of the data due to time constraints. It would be important to ensure that these results are consistent across all datasets and aren't anomalous. The result of the ensemble experiment is especially important to confirm since it is the most applicable result.

Moving Beyond Fine-tuning. As discussed in chapter two, algorithms such as R-CNN use the output of a ConvNet as a generic feature generator to solve other computer vision problems. Although Azizpour has done some work in testing the effect of different source tasks on these algorithms, the work done is hardly conclusive [4]. This thesis provides a framework for effectively finding the answer to this question.

Moving Beyond Fine-Grained Imagery. For this thesis, every target task was learning a specific fine-grained dataset. Many applications may not have a fine-grained target task. In order to confirm these results are universal, an additional dataset would be needed.

Optimal Ensemble Composition. Ensembles are effective because a single ConvNet might overfit the dataset in some specific way. An ensemble overcomes this by using multiple ConvNets that are unlikely to all overfit in the same way. Do different initializations provide this, or does the random nature of stochastic gradient descent do this?

Testing Different Architectures. It is assumed that these results would be universal regardless of the underlying network architecture. Empirical results would be needed to confirm this assumption. In particular, it would be interesting to see the results on GoogLeNet, the current leading architecture.

5.3 Final Remarks

Convolutional Neural Networks have achieved state of the art performance in many computer vision problems in recent years. Image classification is an important problem facing the academic community and is a piece of many problems facing the Air Force such as automated aerial refueling. This thesis shows that transfer learning via parameter fine-tuning is an effective way to improve the performance of ConvNets.

VI. Appendix A: Full Dataset Listing

Table 12. A full list of the classes, images, and sources from the dataset.

Major Class	Minor Class	Images	Train	Test	Source
vegetable	cayenne	730	548	182	ImageNet
vegetable	cucumber	800	600	200	ImageNet
vegetable	tomato	800	600	200	ImageNet
vegetable	radish	800	600	200	ImageNet
vegetable	carrot	800	600	200	ImageNet
vegetable	fava	800	600	200	ImageNet
vegetable	shallot	686	515	171	ImageNet
vegetable	broccoli	800	600	200	ImageNet
vegetable	pumpkin	800	600	200	ImageNet
vegetable	black	515	387	128	ImageNet
vegetable	asparagus	800	600	200	ImageNet
vegetable	brussel_sprouts	800	600	200	ImageNet
vegetable	artichoke	771	579	192	ImageNet
vegetable	cauliflower	800	600	200	ImageNet
vegetable	spinach	748	561	187	ImageNet
vegetable	bell	800	600	200	ImageNet
vegetable	pinto	223	168	55	ImageNet
vegetable	snow_pea	800	600	200	ImageNet
vegetable	kidney	726	545	181	ImageNet
vegetable	mushroom	800	600	200	ImageNet
vegetable	okra	800	600	200	ImageNet
vegetable	leek	536	402	134	ImageNet
vegetable	plantain	717	538	179	ImageNet
vegetable	eggplant	800	600	200	ImageNet
cat	Siamese	200	150	50	Oxford Pets
cat	Egyptian_Mau	194	146	48	Oxford Pets

cat	Birman	200	150	50	Oxford Pets
cat	Russian_Blue	200	150	50	Oxford Pets
cat	Ragdoll	200	150	50	Oxford Pets
cat	Bombay	200	150	50	Oxford Pets
cat	Sphynx	200	150	50	Oxford Pets
cat	British_Shorthair	200	150	50	Oxford Pets
cat	Maine_Coon	200	150	50	Oxford Pets
cat	Persian	200	150	50	Oxford Pets
cat	Abyssinian	198	149	49	Oxford Pets
cat	Bengal	200	150	50	Oxford Pets
flower	bearded_iris	54	41	13	Oxford Flowers
flower	hard-leaved_pocket_orchid	60	45	15	Oxford Flowers
flower	purple_coneflower	85	64	21	Oxford Flowers
flower	bee_balm	66	50	16	Oxford Flowers
flower	foxglove	162	122	40	Oxford Flowers
flower	poinsettia	93	70	23	Oxford Flowers
flower	carnation	52	39	13	Oxford Flowers
flower	japanese_anemone	55	42	13	Oxford Flowers
flower	moon_orchid	40	30	10	Oxford Flowers
flower	alpine_sea_holly	43	33	10	Oxford Flowers
flower	colts_foot	87	66	21	Oxford Flowers
flower	desert-rose	63	48	15	Oxford Flowers
flower	bolero_deep_blue	40	30	10	Oxford Flowers
flower	giant_white_arum_lily	56	42	14	Oxford Flowers
flower	californian_poppy	102	77	25	Oxford Flowers
flower	sword_lily	130	98	32	Oxford Flowers
flower	corn_poppy	41	31	10	Oxford Flowers
flower	primula	93	70	23	Oxford Flowers
flower	water_lily	194	146	48	Oxford Flowers
flower	spring_crocus	42	32	10	Oxford Flowers

flower	petunia	258	194	64	Oxford Flowers
flower	azalea	96	72	24	Oxford Flowers
flower	rose	171	129	42	Oxford Flowers
flower	frangipani	166	125	41	Oxford Flowers
flower	stemless_gentian	66	50	16	Oxford Flowers
flower	peruvian_lily	82	62	20	Oxford Flowers
flower	windflower	54	41	13	Oxford Flowers
flower	blackberry_lily	48	36	12	Oxford Flowers
flower	lenten_rose	67	51	16	Oxford Flowers
flower	daffodil	59	45	14	Oxford Flowers
flower	orange_dahlia	67	51	16	Oxford Flowers
flower	siam_tulip	41	31	10	Oxford Flowers
flower	gazania	78	59	19	Oxford Flowers
flower	oxeye_daisy	49	37	12	Oxford Flowers
flower	spear_thistle	48	36	12	Oxford Flowers
flower	bishop_of_llandaff	109	82	27	Oxford Flowers
flower	gaura	67	51	16	Oxford Flowers
flower	yellow_iris	49	37	12	Oxford Flowers
flower	toad_lily	41	31	10	Oxford Flowers
flower	garden_phlox	45	34	11	Oxford Flowers
flower	grape_hyacinth	41	31	10	Oxford Flowers
flower	red_ginger	42	32	10	Oxford Flowers
flower	canna_lily	82	62	20	Oxford Flowers
flower	fritillary	91	69	22	Oxford Flowers
flower	osteospermum	61	46	15	Oxford Flowers
flower	geranium	114	86	28	Oxford Flowers
flower	hippeastrum_	76	57	19	Oxford Flowers
flower	anthurium	105	79	26	Oxford Flowers
flower	pink_primrose	40	30	10	Oxford Flowers
flower	mexican_petunia	82	62	20	Oxford Flowers

flower	tree_poppy	62	47	15	Oxford Flowers
flower	prince_of_wales_feathers	40	30	10	Oxford Flowers
flower	bougainvillea	128	96	32	Oxford Flowers
flower	sunflower	61	46	15	Oxford Flowers
flower	wild_pansy	85	64	21	Oxford Flowers
flower	king_protea	49	37	12	Oxford Flowers
flower	mallow	66	50	16	Oxford Flowers
flower	barbeton_daisy	127	96	31	Oxford Flowers
flower	pink-yellow_dahlia	109	82	27	Oxford Flowers
flower	artichoke	78	59	19	Oxford Flowers
flower	watercress	184	138	46	Oxford Flowers
flower	cape_flower	108	81	27	Oxford Flowers
flower	magnolia	63	48	15	Oxford Flowers
flower	english_marigold	65	49	16	Oxford Flowers
flower	sweet_william	85	64	21	Oxford Flowers
flower	pincushion_flower	59	45	14	Oxford Flowers
flower	black-eyed_susan	54	41	13	Oxford Flowers
flower	globe_thistle	45	34	11	Oxford Flowers
flower	clematis	112	84	28	Oxford Flowers
flower	globe-flower	41	31	10	Oxford Flowers
flower	great_masterwort	56	42	14	Oxford Flowers
flower	lotus	137	103	34	Oxford Flowers
flower	fire_lily	40	30	10	Oxford Flowers
flower	camellia	91	69	22	Oxford Flowers
flower	mexican_aster	40	30	10	Oxford Flowers
flower	wallflower	196	147	49	Oxford Flowers
flower	tree_mallow	58	44	14	Oxford Flowers
flower	trumpet_creeper	58	44	14	Oxford Flowers
flower	love_in_the_mist	46	35	11	Oxford Flowers
flower	snapdragon	87	66	21	Oxford Flowers

flower	morning_glory	107	81	26	Oxford Flowers
flower	passion_flower	251	189	62	Oxford Flowers
flower	silverbush	52	39	13	Oxford Flowers
flower	ball_moss	46	35	11	Oxford Flowers
flower	thorn_apple	120	90	30	Oxford Flowers
flower	bromelia	63	48	15	Oxford Flowers
flower	tiger_lily	45	34	11	Oxford Flowers
flower	blanket_flower	49	37	12	Oxford Flowers
flower	columbine	86	65	21	Oxford Flowers
flower	hibiscus	131	99	32	Oxford Flowers
flower	monkshood	46	35	11	Oxford Flowers
flower	balloon_flower	49	37	12	Oxford Flowers
flower	ruby-lipped_cattleya	75	57	18	Oxford Flowers
flower	pelargonium	71	54	17	Oxford Flowers
flower	sweet_pea	56	42	14	Oxford Flowers
flower	cautleya_spicata	50	38	12	Oxford Flowers
flower	cyclamen_	154	116	38	Oxford Flowers
flower	buttercup	71	54	17	Oxford Flowers
flower	common_dandelion	92	69	23	Oxford Flowers
flower	bird_of_paradise	85	64	21	Oxford Flowers
flower	canterbury_bells	40	30	10	Oxford Flowers
flower	marigold	67	51	16	Oxford Flowers
bird	Lazuli_Bunting	58	44	14	Caltech-UCSD Birds
bird	Green_tailed_Towhee	60	45	15	Caltech-UCSD Birds
bird	Heermann_Gull	60	45	15	Caltech-UCSD Birds
bird	Ovenbird	60	45	15	Caltech-UCSD Birds
bird	Yellow_headed_Blackbird	56	42	14	Caltech-UCSD Birds
bird	Crested_Auklet	44	33	11	Caltech-UCSD Birds
bird	Horned_Lark	60	45	15	Caltech-UCSD Birds
bird	Groove_billed_Ani	60	45	15	Caltech-UCSD Birds

bird	Red_breasted_Merganser	60	45	15	Caltech-UCSD Birds
bird	Barn_Swallow	60	45	15	Caltech-UCSD Birds
bird	Common_Tern	60	45	15	Caltech-UCSD Birds
bird	Ring_billed_Gull	60	45	15	Caltech-UCSD Birds
bird	Parakeet_Auklet	53	40	13	Caltech-UCSD Birds
bird	Chuck_will_Widow	56	42	14	Caltech-UCSD Birds
bird	Northern_Flicker	60	45	15	Caltech-UCSD Birds
bird	Indigo_Bunting	60	45	15	Caltech-UCSD Birds
bird	Canada_Warbler	60	45	15	Caltech-UCSD Birds
bird	White_necked_Raven	60	45	15	Caltech-UCSD Birds
bird	Henslow_Sparrow	60	45	15	Caltech-UCSD Birds
bird	White_breasted_Kingfisher	60	45	15	Caltech-UCSD Birds
bird	Yellow_bellied_Flycatcher	59	45	14	Caltech-UCSD Birds
bird	Glaucous_winged_Gull	59	45	14	Caltech-UCSD Birds
bird	Acadian_Flycatcher	59	45	14	Caltech-UCSD Birds
bird	Anna_Hummingbird	60	45	15	Caltech-UCSD Birds
bird	Pelagic_Cormorant	60	45	15	Caltech-UCSD Birds
bird	Swainson_Warbler	56	42	14	Caltech-UCSD Birds
bird	Pied_Kingfisher	60	45	15	Caltech-UCSD Birds
bird	White_throated_Sparrow	60	45	15	Caltech-UCSD Birds
bird	Scott_Oriole	60	45	15	Caltech-UCSD Birds
bird	White_Pelican	50	38	12	Caltech-UCSD Birds
bird	Western_Wood_Pewee	60	45	15	Caltech-UCSD Birds
bird	Brown_Pelican	60	45	15	Caltech-UCSD Birds
bird	Le_Conte_Sparrow	59	45	14	Caltech-UCSD Birds
bird	Gray_crowned_Rosy_Finch	59	45	14	Caltech-UCSD Birds
bird	Bobolink	60	45	15	Caltech-UCSD Birds
bird	Bay_breasted_Warbler	60	45	15	Caltech-UCSD Birds
bird	Pacific_Loon	60	45	15	Caltech-UCSD Birds
bird	Tennessee_Warbler	59	45	14	Caltech-UCSD Birds

bird	Geococcyx	60	45	15	Caltech-UCSD Birds
bird	Song_Sparrow	60	45	15	Caltech-UCSD Birds
bird	Savannah_Sparrow	60	45	15	Caltech-UCSD Birds
bird	Lincoln_Sparrow	59	45	14	Caltech-UCSD Birds
bird	Yellow_billed_Cuckoo	59	45	14	Caltech-UCSD Birds
bird	Black_billed_Cuckoo	60	45	15	Caltech-UCSD Birds
bird	Hooded_Merganser	60	45	15	Caltech-UCSD Birds
bird	Forsters_Tern	60	45	15	Caltech-UCSD Birds
bird	White_breasted_Nuthatch	60	45	15	Caltech-UCSD Birds
bird	Prothonotary_Warbler	60	45	15	Caltech-UCSD Birds
bird	Chestnut_sided_Warbler	60	45	15	Caltech-UCSD Birds
bird	American_Crow	60	45	15	Caltech-UCSD Birds
bird	American_Redstart	60	45	15	Caltech-UCSD Birds
bird	Rose_breasted_Grosbeak	60	45	15	Caltech-UCSD Birds
bird	Cerulean_Warbler	60	45	15	Caltech-UCSD Birds
bird	Warbling_Vireo	60	45	15	Caltech-UCSD Birds
bird	Olive_sided_Flycatcher	60	45	15	Caltech-UCSD Birds
bird	Rusty_Blackbird	60	45	15	Caltech-UCSD Birds
bird	Long_tailed_Jaeger	60	45	15	Caltech-UCSD Birds
bird	Louisiana_Waterthrush	60	45	15	Caltech-UCSD Birds
bird	Cape_Glossy_Starling	60	45	15	Caltech-UCSD Birds
bird	Field_Sparrow	59	45	14	Caltech-UCSD Birds
bird	Chipping_Sparrow	60	45	15	Caltech-UCSD Birds
bird	Carolina_Wren	60	45	15	Caltech-UCSD Birds
bird	Bewick_Wren	60	45	15	Caltech-UCSD Birds
bird	Grasshopper_Sparrow	60	45	15	Caltech-UCSD Birds
bird	Black_capped_Vireo	51	39	12	Caltech-UCSD Birds
bird	Ruby_throated_Hummingbird	60	45	15	Caltech-UCSD Birds
bird	Pine_Warbler	60	45	15	Caltech-UCSD Birds
bird	Bohemian_Waxwing	60	45	15	Caltech-UCSD Birds

bird	House_Sparrow	60	45	15	Caltech-UCSD Birds
bird	Hooded_Warbler	60	45	15	Caltech-UCSD Birds
bird	Loggerhead_Shrike	60	45	15	Caltech-UCSD Birds
bird	Philadelphia_Vireo	59	45	14	Caltech-UCSD Birds
bird	Golden_winged_Warbler	59	45	14	Caltech-UCSD Birds
bird	Least_Flycatcher	59	45	14	Caltech-UCSD Birds
bird	Pine_Grosbeak	60	45	15	Caltech-UCSD Birds
bird	Worm_eating_Warbler	59	45	14	Caltech-UCSD Birds
bird	Mangrove_Cuckoo	53	40	13	Caltech-UCSD Birds
bird	Nighthawk	60	45	15	Caltech-UCSD Birds
bird	Dark_eyed_Junco	60	45	15	Caltech-UCSD Birds
bird	Baltimore_Oriole	60	45	15	Caltech-UCSD Birds
bird	Blue_winged_Warbler	60	45	15	Caltech-UCSD Birds
bird	Ivory_Gull	60	45	15	Caltech-UCSD Birds
bird	Sage_Thrasher	60	45	15	Caltech-UCSD Birds
bird	Blue_Grosbeak	60	45	15	Caltech-UCSD Birds
bird	Red_eyed_Vireo	60	45	15	Caltech-UCSD Birds
bird	House_Wren	59	45	14	Caltech-UCSD Birds
bird	Baird_Sparrow	50	38	12	Caltech-UCSD Birds
bird	Caspian_Tern	60	45	15	Caltech-UCSD Birds
bird	Painted_Bunting	58	44	14	Caltech-UCSD Birds
bird	Horned_Puffin	60	45	15	Caltech-UCSD Birds
bird	Prairie_Warbler	60	45	15	Caltech-UCSD Birds
bird	Mallard	60	45	15	Caltech-UCSD Birds
bird	Green_Violetear	60	45	15	Caltech-UCSD Birds
bird	Brown_Creeper	59	45	14	Caltech-UCSD Birds
bird	Brandt_Cormorant	59	45	14	Caltech-UCSD Birds
bird	Fox_Sparrow	60	45	15	Caltech-UCSD Birds
bird	Gray_Kingbird	59	45	14	Caltech-UCSD Birds
bird	Least_Auklet	41	31	10	Caltech-UCSD Birds

bird	Spotted_Catbird	45	34	11	Caltech-UCSD Birds
bird	Yellow_breasted_Chat	59	45	14	Caltech-UCSD Birds
bird	Common_Raven	59	45	14	Caltech-UCSD Birds
bird	Western_Grebe	60	45	15	Caltech-UCSD Birds
bird	Least_Tern	60	45	15	Caltech-UCSD Birds
bird	Shiny_Cowbird	60	45	15	Caltech-UCSD Birds
bird	Red_cockaded_Woodpecker	58	44	14	Caltech-UCSD Birds
bird	Black_and_white_Warbler	60	45	15	Caltech-UCSD Birds
bird	California_Gull	60	45	15	Caltech-UCSD Birds
bird	Slaty_backed_Gull	50	38	12	Caltech-UCSD Birds
bird	Black_throated_Blue_Warbler	59	45	14	Caltech-UCSD Birds
bird	Scarlet_Tanager	60	45	15	Caltech-UCSD Birds
bird	Blue_headed_Vireo	60	45	15	Caltech-UCSD Birds
bird	Cliff_Swallow	60	45	15	Caltech-UCSD Birds
bird	Red_faced_Cormorant	52	39	13	Caltech-UCSD Birds
bird	Whip_poor_Will	49	37	12	Caltech-UCSD Birds
bird	Elegant_Tern	60	45	15	Caltech-UCSD Birds
bird	Bronzed_Cowbird	60	45	15	Caltech-UCSD Birds
bird	Red_winged_Blackbird	60	45	15	Caltech-UCSD Birds
bird	Bank_Swallow	59	45	14	Caltech-UCSD Birds
bird	Evening_Grosbeak	60	45	15	Caltech-UCSD Birds
bird	Laysan_Albatross	60	45	15	Caltech-UCSD Birds
bird	Orchard_Oriole	59	45	14	Caltech-UCSD Birds
bird	Rock_Wren	60	45	15	Caltech-UCSD Birds
bird	Artic_Tern	58	44	14	Caltech-UCSD Birds
bird	Cardinal	57	43	14	Caltech-UCSD Birds
bird	Great_Grey_Shrike	60	45	15	Caltech-UCSD Birds
bird	Red_bellied_Woodpecker	60	45	15	Caltech-UCSD Birds
bird	Eastern_Towhee	60	45	15	Caltech-UCSD Birds
bird	Cedar_Waxwing	60	45	15	Caltech-UCSD Birds

bird	Sayornis	60	45	15	Caltech-UCSD Birds
bird	Yellow_throated_Vireo	59	45	14	Caltech-UCSD Birds
bird	Mourning_Warbler	60	45	15	Caltech-UCSD Birds
bird	Green_Jay	57	43	14	Caltech-UCSD Birds
bird	Blue_Jay	60	45	15	Caltech-UCSD Birds
bird	Fish_Crow	60	45	15	Caltech-UCSD Birds
bird	Nelson_Sharp_tailed_Sparrow	59	45	14	Caltech-UCSD Birds
bird	Myrtle_Warbler	60	45	15	Caltech-UCSD Birds
bird	Eared_Grebe	60	45	15	Caltech-UCSD Birds
bird	Brewer_Sparrow	59	45	14	Caltech-UCSD Birds
bird	Pigeon_Guillemot	58	44	14	Caltech-UCSD Birds
bird	Common_Yellowthroat	60	45	15	Caltech-UCSD Birds
bird	American_Three_toed_Woodpecker	50	38	12	Caltech-UCSD Birds
bird	Boat_tailed_Grackle	60	45	15	Caltech-UCSD Birds
bird	Black_Tern	60	45	15	Caltech-UCSD Birds
bird	American_Goldfinch	60	45	15	Caltech-UCSD Birds
bird	Tree_Swallow	60	45	15	Caltech-UCSD Birds
bird	Summer_Tanager	60	45	15	Caltech-UCSD Birds
bird	Hooded_Oriole	60	45	15	Caltech-UCSD Birds
bird	Belted_Kingfisher	60	45	15	Caltech-UCSD Birds
bird	Gray_Catbird	59	45	14	Caltech-UCSD Birds
bird	Sooty_Albatross	58	44	14	Caltech-UCSD Birds
bird	Western_Meadowlark	60	45	15	Caltech-UCSD Birds
bird	Frigatebird	60	45	15	Caltech-UCSD Birds
bird	European_Goldfinch	60	45	15	Caltech-UCSD Birds
bird	Vermilion_Flycatcher	60	45	15	Caltech-UCSD Birds
bird	Tropical_Kingbird	60	45	15	Caltech-UCSD Birds
bird	Clark_Nutcracker	60	45	15	Caltech-UCSD Birds
bird	Vesper_Sparrow	60	45	15	Caltech-UCSD Birds
bird	Brewer_Blackbird	59	45	14	Caltech-UCSD Birds

bird	Gadwall	60	45	15	Caltech-UCSD Birds
bird	Ringed_Kingfisher	60	45	15	Caltech-UCSD Birds
bird	Nashville_Warbler	60	45	15	Caltech-UCSD Birds
bird	Horned_Grebe	60	45	15	Caltech-UCSD Birds
bird	Rhinoceros_Auklet	48	36	12	Caltech-UCSD Birds
bird	Purple_Finch	60	45	15	Caltech-UCSD Birds
bird	Florida_Jay	60	45	15	Caltech-UCSD Birds
bird	Scissor_tailed_Flycatcher	60	45	15	Caltech-UCSD Birds
bird	Tree_Sparrow	60	45	15	Caltech-UCSD Birds
bird	Black_footed_Albatross	60	45	15	Caltech-UCSD Birds
bird	Mockingbird	60	45	15	Caltech-UCSD Birds
bird	Downy_Woodpecker	60	45	15	Caltech-UCSD Birds
bird	Northern_Fulmar	60	45	15	Caltech-UCSD Birds
bird	Black_throated_Sparrow	60	45	15	Caltech-UCSD Birds
bird	Western_Gull	60	45	15	Caltech-UCSD Birds
bird	Northern_Waterthrush	60	45	15	Caltech-UCSD Birds
bird	White_eyed_Vireo	60	45	15	Caltech-UCSD Birds
bird	Marsh_Wren	60	45	15	Caltech-UCSD Birds
bird	Pileated_Woodpecker	60	45	15	Caltech-UCSD Birds
bird	Magnolia_Warbler	59	45	14	Caltech-UCSD Birds
bird	Pied_billed_Grebe	60	45	15	Caltech-UCSD Birds
bird	White_crowned_Sparrow	60	45	15	Caltech-UCSD Birds
bird	Rufous_Hummingbird	60	45	15	Caltech-UCSD Birds
bird	Seaside_Sparrow	60	45	15	Caltech-UCSD Birds
bird	Pomarine_Jaeger	60	45	15	Caltech-UCSD Birds
bird	American_Pipit	60	45	15	Caltech-UCSD Birds
bird	Yellow_Warbler	60	45	15	Caltech-UCSD Birds
bird	Winter_Wren	60	45	15	Caltech-UCSD Birds
bird	Green_Kingfisher	60	45	15	Caltech-UCSD Birds
bird	Red_legged_Kittiwake	53	40	13	Caltech-UCSD Birds

bird	Kentucky_Warbler	59	45	14	Caltech-UCSD Birds
bird	Brown_Thrasher	59	45	14	Caltech-UCSD Birds
bird	Red_headed_Woodpecker	60	45	15	Caltech-UCSD Birds
bird	Orange_crowned_Warbler	60	45	15	Caltech-UCSD Birds
bird	Harris_Sparrow	60	45	15	Caltech-UCSD Birds
bird	Cactus_Wren	60	45	15	Caltech-UCSD Birds
bird	Wilson_Warbler	60	45	15	Caltech-UCSD Birds
bird	Clay_colored_Sparrow	59	45	14	Caltech-UCSD Birds
bird	Herring_Gull	60	45	15	Caltech-UCSD Birds
bird	Cape_May_Warbler	60	45	15	Caltech-UCSD Birds
bird	Great_Crested_Flycatcher	60	45	15	Caltech-UCSD Birds
bird	Palm_Warbler	60	45	15	Caltech-UCSD Birds
sign	sign_00031	149	112	37	KUL Belgian Sign Benchmark
sign	sign_00054	166	125	41	KUL Belgian Sign Benchmark
sign	sign_00047	178	134	44	KUL Belgian Sign Benchmark
sign	sign_00053	223	168	55	KUL Belgian Sign Benchmark
sign	sign_00038	495	372	123	KUL Belgian Sign Benchmark
sign	sign_00041	159	120	39	KUL Belgian Sign Benchmark
sign	sign_00040	290	218	72	KUL Belgian Sign Benchmark
sign	sign_00007	247	186	61	KUL Belgian Sign Benchmark
sign	sign_00061	376	282	94	KUL Belgian Sign Benchmark
sign	sign_00018	200	150	50	KUL Belgian Sign Benchmark
sign	sign_00001	137	103	34	KUL Belgian Sign Benchmark
sign	sign_00028	176	132	44	KUL Belgian Sign Benchmark
sign	sign_00021	88	66	22	KUL Belgian Sign Benchmark
sign	sign_00056	128	96	32	KUL Belgian Sign Benchmark
sign	sign_00017	262	197	65	KUL Belgian Sign Benchmark
sign	sign_00057	119	90	29	KUL Belgian Sign Benchmark
sign	sign_00032	738	554	184	KUL Belgian Sign Benchmark
sign	sign_00035	214	161	53	KUL Belgian Sign Benchmark

sign	sign_00039	295	222	73	KUL Belgian Sign Benchmark
sign	sign_00045	158	119	39	KUL Belgian Sign Benchmark
sign	sign_00019	394	296	98	KUL Belgian Sign Benchmark
sign	sign_00037	129	97	32	KUL Belgian Sign Benchmark
sign	sign_00013	129	97	32	KUL Belgian Sign Benchmark
sign	sign_00022	436	327	109	KUL Belgian Sign Benchmark
dog	Airedale	745	559	186	ImageNet
dog	toy_poodle	800	600	200	ImageNet
dog	border_terrier	800	600	200	ImageNet
dog	tibetan_terrier	460	345	115	ImageNet
dog	soft-coated_terrier	800	600	200	ImageNet
dog	curly-coated_retriever	238	179	59	ImageNet
dog	australian_terrier	650	488	162	ImageNet
dog	griffon	122	92	30	ImageNet
dog	bullterrier	800	600	200	ImageNet
dog	standard_poodle	800	600	200	ImageNet
dog	old_english_sheepdog	723	543	180	ImageNet
dog	toy_terrier	409	307	102	ImageNet
dog	miniature_poodle	422	317	105	ImageNet
dog	shepherd	800	600	200	ImageNet
dog	shetland_sheepdog	800	600	200	ImageNet
dog	wolfhound	788	591	197	ImageNet
dog	greyhound	800	600	200	ImageNet
dog	Rottweiler	783	588	195	ImageNet
dog	african_hunting_dog	791	594	197	ImageNet
dog	bull_mastiff	554	416	138	ImageNet
dog	kuvasz	579	435	144	ImageNet
dog	miniature_schnauzer	800	600	200	ImageNet
dog	clumber	513	385	128	ImageNet
dog	basenji	800	600	200	ImageNet

dog	border_collie	800	600	200	ImageNet
dog	cairn	800	600	200	ImageNet
dog	whippet	800	600	200	ImageNet
dog	sussex_spaniel	395	297	98	ImageNet
dog	EntleBucher	106	80	26	ImageNet
dog	Pekinese	800	600	200	ImageNet
dog	boxer	800	600	200	ImageNet
dog	sealyham_terrier	182	137	45	ImageNet
dog	west_highland_white_terrier	800	600	200	ImageNet
dog	maltese_dog	800	600	200	ImageNet
dog	schipperke	800	600	200	ImageNet
dog	otterhound	158	119	39	ImageNet
dog	golden_retriever	800	600	200	ImageNet
dog	chesapeake_bay_retriever	800	600	200	ImageNet
dog	bedlington_terrier	436	327	109	ImageNet
dog	Bernese_mountain_dog	800	600	200	ImageNet
dog	beagle	800	600	200	ImageNet
dog	springer	800	600	200	ImageNet
dog	deerhound	800	600	200	ImageNet
dog	labrador_retriever	800	600	200	ImageNet
dog	wire-haired_fox_terrier	211	159	52	ImageNet
dog	briard	767	576	191	ImageNet
dog	tibetian_mastiff	800	600	200	ImageNet
dog	flat-coated_retriever	800	600	200	ImageNet
dog	Doberman	800	600	200	ImageNet
dog	Flandres	800	600	200	ImageNet
dog	bulldog	800	600	200	ImageNet
dog	Saluki	702	527	175	ImageNet
dog	Samoyed	800	600	200	ImageNet
dog	Appenzeller	634	476	158	ImageNet

dog	greater_swiss_mountain_dog	634	476	158	ImageNet
dog	coonhound	156	117	39	ImageNet
dog	elkhound	477	358	119	ImageNet
dog	malinois	800	600	200	ImageNet
dog	ridgeback	800	600	200	ImageNet
dog	lakeland_terrier	499	375	124	ImageNet
dog	affenpinscher	278	209	69	ImageNet
dog	boston_bull	800	600	200	ImageNet
dog	hairless	163	123	40	ImageNet
dog	Dinmont	451	339	112	ImageNet
dog	dingo	800	600	200	ImageNet
dog	norfolk_terrier	470	353	117	ImageNet
dog	borzoi	800	600	200	ImageNet
dog	kelpie	800	600	200	ImageNet
dog	ibizan_hound	433	325	108	ImageNet
dog	kerry_blue_terrier	373	280	93	ImageNet
dog	Shih-Tzu	800	600	200	ImageNet
dog	silky_terrier	800	600	200	ImageNet
dog	brittany_spaniel	800	600	200	ImageNet
dog	blehnheim_spaniel	386	290	96	ImageNet
dog	norwich_terrier	516	387	129	ImageNet
dog	Chihuahua	800	600	200	ImageNet
dog	Pyrenees	800	600	200	ImageNet
dog	groenendael	756	567	189	ImageNet
dog	chow	800	600	200	ImageNet
dog	pug	800	600	200	ImageNet
dog	keeshond	800	600	200	ImageNet
dog	irish_terrier	800	600	200	ImageNet
dog	standard_schnauzer	418	314	104	ImageNet
dog	giant_schnauzer	736	552	184	ImageNet

dog	husky	786	590	196	ImageNet
dog	pointer	368	276	92	ImageNet
dog	malamute	800	600	200	ImageNet
dog	yorkshire_terrier	800	600	200	ImageNet
dog	bluetick	252	189	63	ImageNet
dog	gordon_setter	800	600	200	ImageNet
dog	irish_water_spaniel	268	201	67	ImageNet
dog	komondor	382	287	95	ImageNet
dog	english_setter	800	600	200	ImageNet
dog	collie	800	600	200	ImageNet
dog	bloodhound	648	486	162	ImageNet
dog	scotch_terrier	800	600	200	ImageNet
dog	pinscher	800	600	200	ImageNet
dog	Newfoundland	800	600	200	ImageNet
dog	basset	800	600	200	ImageNet
dog	Bernard	800	600	200	ImageNet
dog	eskimo_dog	800	600	200	ImageNet
dog	redbone	454	341	113	ImageNet
dog	Pomeranian	800	600	200	ImageNet
dog	Dane	800	600	200	ImageNet
dog	Lhasa	800	600	200	ImageNet
dog	dhole	407	306	101	ImageNet
dog	American_Staffordshire_terrier	720	540	180	ImageNet
dog	papillon	703	528	175	ImageNet
dog	Leonberg	491	369	122	ImageNet
dog	foxhound	211	159	52	ImageNet
dog	japanese_spaniel	143	108	35	ImageNet

Bibliography

1. Convolutional neural networks (lenet). http://deeplearning.net/tutorial/lenet.html.

2. Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Man, Rajat Monga, Sherry Moore, Derek Murray, Jon Shlens, Benoit Steiner, Ilya Sutskever, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Oriol Vinyals, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *arXiv:1603.04467v2*, page 19, 2015.

3. Han Altae-Tran, Bharath Ramsundar, Aneesh S Pappu, and Vijay Pande. Low Data Drug Discovery with One-shot Learning. *arXiv:1611.03199*, pages 1–20, 2016.

4. Hossein Azizpour, Ali Sharif Razavian, Josephine Sullivan, Atsuto Maki, and Stefan Carlsson KTH. From Generic to Specific Deep Representations for Visual Recognition. *Computer Vision and Pattern Recognition*, 2015.

5. Soheil Bahrampour, Naveen Ramakrishnan, Lukas Schott, and Mohak Shah. Comparative Study of Deep Learning Software Frameworks. *arXiv:1511.06435v3*, 2016.

6. Herbert Bay, Tinne Tuytelaars, Luc Van Gool, A. Leonardis, Horst Bischof, and Axel Pinz. SURF: Speeded Up Robust Features. *Computer Vision*, 3951:404–417, 2006.

7. Yoshua Bengio. Practical Recommendations for Gradient-Based Training of Deep Architectures. In *arXiv:1206.5533*, pages 437–478. 2012.

8. Yoshua Bengio, Patrice Simard, and P Frasconi. Learning Long Term Dependencies with Gradient Descent is Difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

9. James Bergstra, Olivier Breuleux, Frederic Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math compiler in Python. *Proceedings of the Python for Scientific Computing Conference (SciPy)*, (Scipy):1–7, 2010.

10. Leon Bottou. Stochastic Gradient Descent Tricks. In *Neural Networks: Tricks of the Trade*, chapter 18, pages 421–436. Springer Berlin Heidelberg, 2012.

11. Kumar Chellapilla, S Puri, and Patrice Simard. High Performance Convolutional Neural Networks for Document Processing. *Tenth International Workshop on Frontiers in Handwriting Recognition*, 2006.

12. Dan Claudiu Ciresan, Ueli Meier, and Jürgen Schmidhuber. Multi-column Deep Neural Networks for Image Classification. *CVPR*, pages 3642–3649, 2012.

13. Ronan Collobert. Torch7: A matlab-like environment for machine learning. *BigLearn, NIPS Workshop*, pages 1–6, 2011.

14. Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. *Proceedings of the 25th international conference on Machine learning*, pages 160–167, 2008.

15. David E. Ruineihart, Geoffrey E Hinton and Ronald J. Williams. Learning Internal Representations by Error Propagation. Technical Report 1, University of California San Diego La Jolla Instititue for Cognitive Science, 1985.

16. Chen Debao. Degree of approximation by superpositions of a sigmoidal function. *Approximation Theory and its Applications*, 9(3):17–28, 1993.

17. Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition. *International Conference on Machine Learning*, 32:647–655, 2014.

18. Dumitru Erhan, Aaron Courville, and Yoshua Bengio. Understanding Representations Learned in Deep Architectures. *Network*, (September 2016):1–25, 2010.

19. Clement Farabet, Berin Martini, Polina Akselrod, Selçuk Talay, Yann LeCun, and Eugenio Culurciello. Hardware Accelerated Convolutional Neural Networks for Synthetic Vision Systems. *Proc. International Symposium on Circuits and Systems (ISCAS'10)*, 2010.

20. Kunihiko Fukushima and Sei Miyake. Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position. *Pattern Recognition*, 15(6):455–469, 1982.

21. Colin Fyfe. The Multilayer Perceptron: backprop. In *Artificial Neural Networks and Information Theory*, chapter 4, pages 43–62. University of Paisley, 1.2 edition, 2000.

22. Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014.

23. Yanming Guo, Yu Liu, Ard Oerlemans, Songyang Lao, Song Wu, and Michael S. Lew. Deep learning for visual understanding: A review. *Neurocomputing*, 187:27–48, 2016.

24. Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *ArXiv e-prints*, pages 1–18, 2012.

25. D. H. Hubel and T. N. Wiesel. Receptive Fields, Binocular Interaction and Functional Architecture in the Cat's Visual Cortex. *Journal of Physiology*, 160:106–154, 1962.

26. D. H. Hubel and T. N. Wiesel. Receptive Fields and Functional Architecture of monkey striate cortex. *Journal of Physiology*, 195:215–243, 1968.

27. Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional Architecture for fast feature embedding. *Proceedings of the ACM International Conference on Multimedia - MM '14*, pages 675–678, 2014.

28. Aditya Khosla, Nityananda Jayadevaprakash, Bangpeng Yao, and Li Fei-Fei. Novel dataset for fine-grained image categorization: Stanford dogs. *First Workshop on Fine-Grained Visual Categorization in IEEE Conference on Computer Vision and Pattern Recognition*, 2011.

29. Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. *... Science Department, University of Toronto, Tech. ...*, pages 1–60, 2009.

30. Alex Krizhevsky. One weird trick for parallelizing convolutional neural networks. *arXiv preprint*, pages 1–7, 2014.

31. Alex Krizhevsky, IIya Sulskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information and Processing Systems (NIPS)*, pages 1–9, 2012.

32. Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361:255–258, 1995.

33. Yann LeCun, Yoshua Bengio, and Geoffrey E Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

34. Yann LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W Hubbard, and L. D. Jackel. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4):541–551, 1989.

35. Yann LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W Hubbard, and L. D. Jackel. Handwritten Digit Recognition with a Back-Propagation Network. *Advances in Neural Information Processing Systems*, 2, 1990.

36. Yann LeCun, Leon Bottou, and Yoshua Bengio. Reading Checks with Multilayer Graph Transformer Networks. *International Conference on Acoustics, Speech, and Signal Processing*, pages 151–154, 1997.

37. Yann LeCun, Fu Jie Huang Fu Jie Huang, and Leon Bottou. Learning Methods for Generic Object Recognition with Invariance to Pose and Lighting. *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, 2:II–97 – 104, 2004.

38. David G Lowe. SIFT. *Computer Vision*, 2:1150–1157, 1999.

39. Subhransu Maji, Esa Rahtu, Juho Kannala, Matthew Blaschko, and Andrea Vedaldi. Fine-Grained Visual Classification of Aircraft. *TechReport*, 2013.

40. Robert Mash, Nicholas Becherer, Brian Woolley, and John Pecarina. Toward Aircraft Recognition With Convolutional Neural Networks. In *National Aerospace & Electronics Conference*, Dayton, 2016.

41. Vinod Nair and Geoffrey E Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. *Proceedings of the 27th International Conference on Machine Learning*, (3):807–814, 2010.

42. Maria-Elena Nilsback and Andrew Zisserman. A Visual Vocabulary for Flower Classification. In *Comptuer Vision and Pattern Recognition*, pages 1447–1454, 2006.

43. O M Parkhi, A Vedaldi, A Zisserman, and C V Jawahar. Cats and dogs. *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3498–3505, 2012.

44. Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Josephine Carlsson. CNN Features off-the-shelf: an Astounding Baseline for Recognition. *Computer Vision and Pattern Recognition*, 2014.

45. Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. *arXiv preprint arXiv*, page 1312.6229, 2013.

46. Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *ImageNet Challenge*, pages 1–10, 2014.

47. Daniel Strigl, Klaus Kofler, and Stefan Podlipnig. Performance and Scalability of GPU-Based Convolutional Neural Networks. *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, pages 317–324, 2010.

48. C Szegedy, W Liu, Yangqing Jia, and Pierre Sermanet. Going deeper with convolutions. *arXiv preprint arXiv: 1409.4842*, 2014.

49. Radu Timofte, Karel Zimmermann, and Luc Van Gool. Multi-view traffic sign detection, recognition, and 3D localisation. *Machine Vision and Applications*, 25(3):633–647, 2014.

50. J. R R Uijlings, K. E A Van De Sande, T. Gevers, and A. W M Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 104(2):154–171, 2013.

51. C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The Caltech-UCSD Birds-200-2011 Dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011.

52. Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *Advances in Neural Information Processing Systems 27 (Proceedings of NIPS)*, 27:1–9, 2014.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
23–03-2017	Master's Thesis	Jun 2015 — Mar 2017

4. TITLE AND SUBTITLE

Transfer Learning in Convolutional Neural Networks for Fine-Grained Image Classification

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Becherer, Nicholas C. 2d Lt, USAF

5d. PROJECT NUMBER

16G189

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENG-MS-17-M-005

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Ba T. Nguyen
DR-03, Senior Flight Control Engineer
Aerospace Systems Directorate
2130 Eighth Street, WPAFB, OH 45433-7542
COMM 937-938-4617
Email: ba.nguyen@us.af.mil

10. SPONSOR/MONITOR'S ACRONYM(S)

AFRL/RQQC

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
Approved for Public Release; distribution unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

In recent years, convolutional neural networks have achieved state of the art performance in a number of computer vision problems such as image classification. Prior research has shown that a transfer learning technique known as parameter fine-tuning wherein a network is pre-trained on different datasets can boost the performance of these networks. However, the topic of identifying the best source dataset and learning strategy for a given target domain is largely unexplored. Thus, this research presents and evaluates various transfer learning methods for fine-grained image classification as well as the effect on ensemble networks. The main contributions are a framework to evaluate the effectiveness of transfer learning, an optimal strategy for parameter fine-tuning, and a thorough demonstration of its effectiveness. The experimental framework and findings will help to train models in reduced time and with improved accuracy for target recognition and automated aerial refueling.

15. SUBJECT TERMS

Convolutional Neural Networks, Transfer Learning, Parameter Fine-tuning, Image Classification, Computer Vision

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Lt Col John Pecarina, AFIT/ENG
U	U	U	U	157	19b. TELEPHONE NUMBER *(include area code)* (937) 255-3636, x3368; john.pecarina@afit.edu