4-22-2019 11:30 AM

Gabor Filter Initialization And Parameterization Strategies In Convolutional Neural Networks

Long Pham
The University of Western Ontario

Supervisor
McIsaac, Kenneth
The University of Western Ontario

Graduate Program in Electrical and Computer Engineering
A thesis submitted in partial fulfillment of the requirements for the degree in Master of Engineering Science

Acknowledgments

I would especially like to mention my advisor, Dr. Ken McIsaac, for supporting me throughout the course of 3 years. Thank you for always being the friendly guy that could always provide insight and providing a path during the course of this thesis. Thank you for supplying the tools I needed to succeed in the research. I have grown to see past just doing research but enjoying it as much as possible to become a better researcher, engineer, and friend.

I would like to mention my colleagues Justin Szoke-Sieswerda, Alexis Pascual, and Matt Cross for mentoring and getting me through the hard times who have provided very informative direction for my thesis.

I would like to thank my family who have supported me through my journey of writing this thesis.

I would like to mention the Western Outdoors Club and all the friends I have met during this experience that have helped relieve stress during the hard times and learning how to enjoy life outside of academia.

Lastly, I would like to mention my loving girlfriend Hilary Scott for heavily supporting me through university. You have always been there when I needed you the most.

Abstract

Convolutional neural networks (CNN) have been widely known in literature to be extremely effective for classifying images. Some of the filters learned during training of the first layer of a CNN resemble the Gabor filter. Gabor filters are extremely good at extracting features within an image. We have taken this as an incentive by replacing the first layer of a CNN with the Gabor filter to increase speed and accuracy for classifying images. We created two simple 5-layer AlexNet-like CNNs comparing grid-search to random-search for initializing the Gabor filter bank. We trained on MNIST, CIFAR-10, and CIFAR-100 as well as a rock dataset created at Western University to study the classification of rock images using a CNN. When training on this rock dataset, we use an architecture from literature and use our Gabor filter substitution method to show the usage of the Gabor filter. Using the Gabor convolutional neural network (GCNN) showed improvements in the training speed across all datasets tested. We also found that the GCNN underperforms when dropout is added, even when overfitting becomes an issue. The size of the Gabor filter bank becomes a hyperparameter that can be tuned per dataset. Applying our Gabor filter replacement method to a 3-layer CNN reduced final accuracy at epoch 200 by 1.16% but showed large improvements in the speed of convergence during training with 93.44% accuracy on a validation set after 10 epochs compared to the original network's 82.19%.

Keywords: convolutional neural network, Gabor filter, initialization strategies, random-search, grid-search, paramerization, dropout, GCNN

Contents

List of Figures

List of Tables

List of Appendices

Chapter 1

Introduction

1.1 Motivation

Since the inceptions of CNNs brought from the 1950s by Hubel and Wiesel [9], the brain's anatomy analogy brought the term "Neocognitron", which is a model of how neurons within the brain communicate to each other. A CNN takes the same idea of the Neocognitron where we have an input (image) that is fed into a network of neurons and spits out the identification of an image. Fifty years later, the architecture of CNNs have bloomed from this idea and matured for the use of image processing to classify, localize, and detect images. Alexander Krizhevsky's neural network created in 2012 [12] brought forward and gave new life to the meaning of convolutional neural networks. Krizhevsky's AlexNet built a CNN used for the ImageNet Large Scale Visual Recognition Competition (ILSVRC) competition in 2012 and took first place with outstanding results with a top-5 error rate of 15.3%. The top-5 error rate is the measurement of accurately classifying an image within a dataset using 5 predictions of what the image is and comparing these results with the ground-truth (the correct answer). This was a milestone because they had scored more than 10% better than second place. Not only did this bring a faster, and more efficient way to do computations on images, it brought competition to see who can build the best image classifier using these CNNs.

Six years later, we find ourselves using networks like ResNet, VGG-16, VGG-19, GoogLeNet, DenseNet, and many more. These newer, faster networks provide higher scores in comparison to DenseNet's top-1 and top-5 error rates scoring 20.27% and a 5.17% respectively (DenseNet was the best network in 2017 that won the ILSVRC competition). The latest networks are marginally better than each other in terms of speed and accuracy and is dependent on the dataset. No single network stands out as the "top", or "go-to" network and each offer their own advantages that will be described in Section 2. Finding faster and more efficient ways to increase training speed while maintaining or increasing accuracy is the name of the game and everyone in the field of deep learning is trying to create a faster, more efficient network.

Convolutional neural networks (CNN) performance are usually measured with 2 numbers: the training time and the accuracy (or alternatively, the top-5/top-1 error rate. i.e., We can train MNIST for 200 epochs each running 10 seconds resulting a top-5 error rate of 99.5%. People can usually balance between the two metrics, i.e., a lower training time can result in lower error rates and vice versa. The ultimate goal is to find a low training time, either by hardware (more expensive) or software (less expensive) and have a high accuracy rate for a given dataset. Size does matter in the case of datasets because of the limitations of computation.

The first use of CNNs in AlexNet, they had used Nvidia's GTX580 card to do their computations on ImageNet, which is a very large dataset and this took them approximately 6 days to compute [12]. Nowadays, we have faster, more efficient cards that are more capable of processing such a dataset. Having the right hardware is only half the battle in training models for such datasets. Improving the software is the other half.

Although CNNs are currently the fastest method for image processing, they are still slow to train on very large datasets. By improving the CNNs architecture, analyzing images can become faster, and more efficient. Not only can we unlock new potential breakthroughs but we can also reduce training times to create CNN models. This can be seen as a reduction in cost of computing such large datasets, to using less energy, and ultimately saving time. Keeping these benefits in mind, there's always a need to make "things" faster, better, and more efficient. This

was the driving factor of creating GoogLeNet [24].

The Gabor filter is a type of bandpass filter that specializes in finding features like edges within an image. The filter is able modulate over an image to extract features on different angles. The Gabor filter is a good candidate filter to be used before or during training [26]. Using a small variation on Krizhevsky's network, and using his DeConvNet, he was able to show visualizations of each layer in AlexNet. Within this network, the features extracted by the network resembled the Gabor kernel itself (Figure 1.1). The question arises that if the features gathered in a CNN resemble and/or are very similar to images of a Gabor kernel, then is it possible to skip learning and training earlier stages of the network by using this filter? There have already been many papers discussing the use of Gabor filters within their networks [2] [15] [19] [16] that shows very promising results in terms of speed and accuracy for various databases.

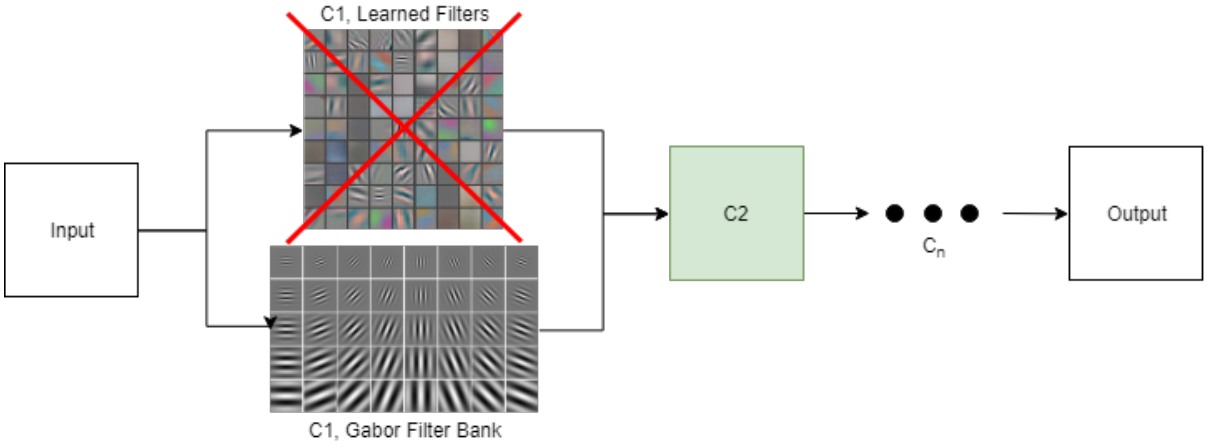


Figure 1.1: As seen from [26], we see the first layer that was learned during a simple AlexNet-like CNN. We can see that in some of the filters that it learns, it appears to look like the Gabor filter. Using this, we skip the step of learning the Gabor filter, and just insert the Gabor filter in the first layer as a shortcut to training.

The purpose of this thesis is to introduce initialization strategies to try and effectively decrease the training times for individual neural networks. More specifically, there has been recent interest in Gabor Filters. Using the implementation of Gabor filters instead of using learned filters in earlier layers shows that training time can be reduced dramatically, while

maintaining similar, if not better, results at the end of training [19]. The question of how parameterize the Gabor filter within the early layers is still open. Finding a good setup for a CNN can be a challenge for either known or unknown datasets. The goal of this thesis is to: 1) explore methods of implementing the Gabor filter, 2) find ways to enhance the speed and accuracy for a given dataset, 3) and find when and how the Gabor filter can be used to effectively reduce training times while keeping similar, if not better accuracy than the current the state-of-the-art (SOTA).

1.2 Background

1.2.1 What is an Image?

Within the computing domain, we can describe an image as a matrix of numbers. These numbers are represented as different light intensities which altogether make up the image. A pixel is the term representing the numbers within the matrix assorted in rows and columns. The pixel values are data dependent and can range from a minimum to a maximum value. For simplicity sake, let's say that the pixel light intensity values are represented as an 8-bit number, or a range of 0 to 255. The higher the value, the greater the intensity (the pixel appears brighter with a higher intensity). Colored images are overlapping matrices with 3 separate channels: red, green, and blue. In image processing, handling color images require more processing than grayscale because there are 3 channels to compute on instead of 1. Because images are represented in terms of matrices full of intensity values, we can use computation to process and analyze an image or an array of images.

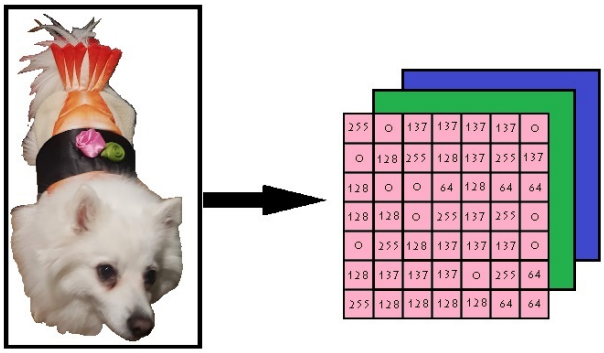


Figure 1.2: An RGB image of a dog that shows the following RGB matrix which has 3 separate matrices, one for the red channel, one for the green channel, and one for the blue channel. If the image was a grayscale image, there would only be one channel.

1.2.2 Details about CNN

In image processing, computers deal with images in terms of pixels. These pixels are numbers that the computer knows how to read and understand (can be seen in the matrices in Figure 1.2 on the right). These numbers are known as intensity levels and they are usually arranged in a matrix like fashion with rows and columns. Images can vary from small images of 128×128 pixels to larger images of, say, 2048×2048 pixels. Images themselves can also have up to 3 channels (for red, blue, and green channels) or alternatively, if the image is in grayscale, there is only 1 channel. When using a neural network on an image, it accounts for just the raw data (pixels) for the network to analyze and learn about the image. The larger the image, the more time is required to process the image. Alternatively, if the neural network is fast and efficient with the usage of good algorithms, the computation required to analyze the image can be greatly reduced. Having good hardware can also reduce computation time but taking this route is expensive.

Convolutional neural networks (CNN) are many filters or kernels that are usually small in size (3×3, 5×5, 7×7, etc.) that are computed on top of the input image to produce feature maps. These filters slide through the image from left to right, top to bottom with a term called *stride*. The stride is just how many pixels the kernel moves within the image. Once the filter moves through the image, it creates an array of feature maps. These feature maps are matrices

of numbers that correspond to the image's features such as edges, or shapes. These feature maps keep going through the network with the help of other layers like the pooling layer, or the activation layer until it reaches the end where normally the last couple of layers are the fully connected layers which help with classification. See Figure 2.12.

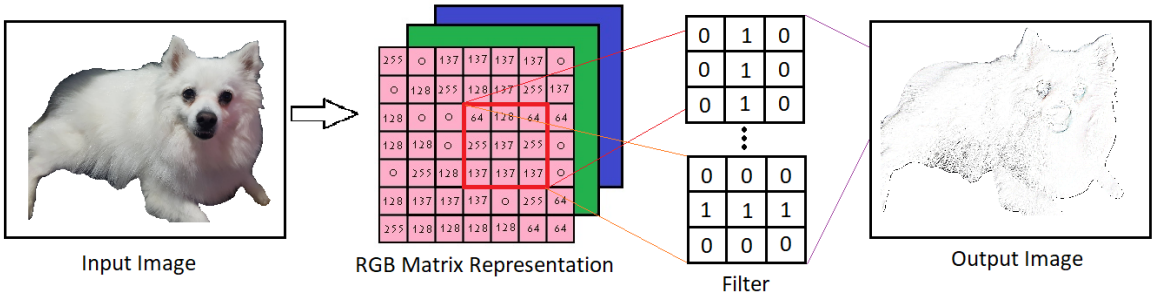


Figure 1.3: An input image of a dog getting fed into a convolution process, which is just a matrix multiplication operation done on the original input image to produce the output image. Here, the filter bank finds lines within the image and outputs the lines found from the original image.

From Figure 1.3, we can see a simple convolutional process which happens within the CNN. Different convolutions can happen throughout a CNN and during the training time, these filters are "learned" by the neural network through many rounds iterations or epochs. Through backpropagation, these filters will be updated through training in order to give a greater response of finding such features. Backpropagation is the operation of computing the error throughout the network and its goal is to reduce the error, and in turn, increase the accuracy of the network. During training, if the loss function begins to increase, that signifies that the model is starting to overfit on the dataset. Overfitting is the term used to describe a model training on a dataset where it can correctly identify the classes within the training set, but when new data appears, the model struggles to identify the class.

When the convolutions compute on the input image, the feature maps produced detect low level features at first and then show higher level features the deeper the network is. For example, in the first layer, the feature maps learned are usually detecting features like edges. As the layers go down, these edges found become corners, and eventually can become basic

shapes. As the pattern progresses, the feature maps will start to draw out classes that resemble, for example, a picture of a dog's face. It is important to note here that in the early stages of the CNN, edge detection is commonly found first. This can be seen in [26]. This thesis is to find the link between Gabor filters and using them in the first layer of a CNN because Gabor filters are very good at detecting edge-like features (Figure 1.1).

Zeiler's DeConvNet is essentially a backwards-flowing CNN [27] [26]. They produce images of the feature maps that have been learned within each of the layers from a modified AlexNet (Figure 2.12). In the first layer of this figure, you can see that the CNN learns about edges. It is very common for a CNN to learn about edges as a low level feature. As the network progresses, more complex features are learned such as corners and arches, to shapes, and patterns, which can also be seen from the figure previously mentioned.

The important piece here is that Gabor filters already look like the features that the CNN learns within the early layers, if not just the first (Figure 1.6). These filters will be described in the next section. With this observation, researchers have been interested in the usage of Gabor filters with CNNs. There have been useful studies found that show and prove the efficiency of using this filter. A typical scenario of using Gabor filter is that instead of the CNN "learning" their own filters (which may take some or many epochs) it replaces the learning phase by substituting the Gabor filter bank in the first layer of a CNN. This results better efficiency of the network and reduces the training time [19] [16] since we are not training and consequently learning from large networks or alternatively taking weights from a pre-trained network like ImageNet. Training time for ImageNet takes a lot of resources and many hours of training time and taking shortcuts, like inserting the Gabor filter bank in layer 1, can help cut down training time.

1.2.3 Gabor Filter

The Gabor filter was named after Dennis Gabor in 1946 [4]. A Gabor filter is a type of bandpass filter which computes on a range within a frequency to accept or reject computations done on

the filter. The Gabor filter is used for finding different textures, edges, and feature extractions and is found from using a Gaussian kernel function that is modulated by a sinusoidal wave.

complex

$$g(x, y; \lambda, \theta, \psi, \sigma, \gamma) = (-\frac{x'^2 + \gamma^{2'} y'^2}{2\sigma^2}) \exp(i(2\pi \frac{x'}{\lambda} + \psi)) \tag{1.1}$$

real

$$g(x, y; \lambda, \theta, \psi, \sigma, \gamma) = (-\frac{x'^2 + \gamma^{2'} y'^2}{2\sigma^2}) \cos(2\pi \frac{x'}{\lambda} + \psi) \tag{1.2}$$

imaginary

$$g(x, y; \lambda, \theta, \psi, \sigma, \gamma) = (-\frac{x'^2 + \gamma^{2'} y'^2}{2\sigma^2}) \sin(2\pi \frac{x'}{\lambda} + \psi) \tag{1.3}$$

where

$$x' = x \cos\theta + y \sin\theta \tag{1.4}$$

and

$$y' = -x \sin\theta + y \cos\theta \tag{1.5}$$

The Gabor filter's complex equation given above (Equation 1.1) shows the combination of the real and imaginary parts of the waveform (Equations 1.2 and 1.3). In Equations 1.4 and 1.5, these are what control the Gabor filter's center frequency which show the highest response of the filter. The power spectrum of the Gabor filter is made of 2 impulses of a sine wave and a Gaussian. When we multiply these in the spatial domain, it is also known to be a convolution in the frequency domain. Due to the uncertainty principle which says that we cannot know the frequency of a particle if we look at the particle within a snapshot of time (or vice versa),

we cannot accurately measure the its counterpart. Using this notion, the complex Gabor filter equation (Equation 1.1 compromises the localization in the time and frequency domain [3].

Within the filter, it uses many parameters that can be tuned and changed to find and extract information. These parameters each offer their own attribute to the filter and apply different results to the image. Using the OpenCV library, we can parameterize the filter with the following variables described below. With so many parameters it would be hard to find a solution for small or large datasets.

Gabor Kernel Size, *ksize*

The *ksize* is the size of the kernel of the Gabor Filter that is specified by the user within the OpenCV library. From Figure 1.4, we can see that with smaller kernels, the images appear to find certain textures, while having larger kernels, seems to find an large objects. After the kernel size, $ksize = 31$, there appears to be no change to the filtered image, which shows some scaling invariance. Please note that the image input is of size 220×220 pixels. Using larger images may have different results and at some point larger kernel sizes show no differences in the filtered image. This is because as the filter size gets larger, it almost matches the input image itself, showing only the Gaussian blurring effect.

(a) Original Image (b) 3×3 (c) 7×7 (d) 15×15 (e) 25×25

(f) 31×31 (g) 51×51 (h) 101×101 (i) 201×201 (j) 301×301

Figure 1.4: Given an input image of size 220×220 pixels, we change the *ksize* starting from row 1 where $ksize = [3, 7, 15, 25]$, and row 2 where $ksize = [31, 51, 101, 201, 301]$ from left to right. The parameters $\sigma, \theta, \lambda, \gamma$, and ψ are fixed at $[3.0, 0.0, 8.0, 0.3, 0.0]$ respectively.

Standard Deviation (in Gaussian Distribution), σ

Sigma, σ, within equation 1.2 is the standard deviation within the Gaussian function which controls the spread within the function. From Figure 1.5, it appears that the image gets blurrier.

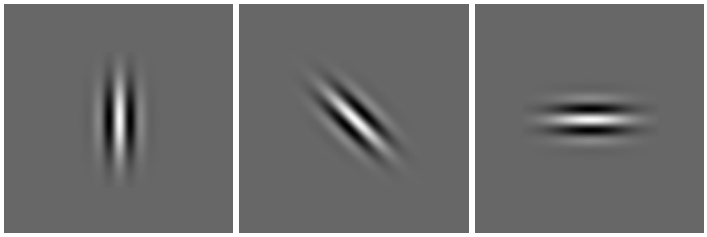
(a) Original Image (b) $\sigma = 1.0$ (c) $\sigma = 2.0$ (d) $\sigma = 3.0$ (e) $\sigma = 4.0$

Figure 1.5: Given an input image of size 220×220 pixels, we change $\sigma = [1.0, 2.0, 3.0, 4.0]$ from left to right. The parameters $ksize, \theta, \lambda, \gamma$, and ψ are fixed at $[15 \times 15, 0.0, 10.0, 0.5, 0.0]$ respectively.

Orientation, θ

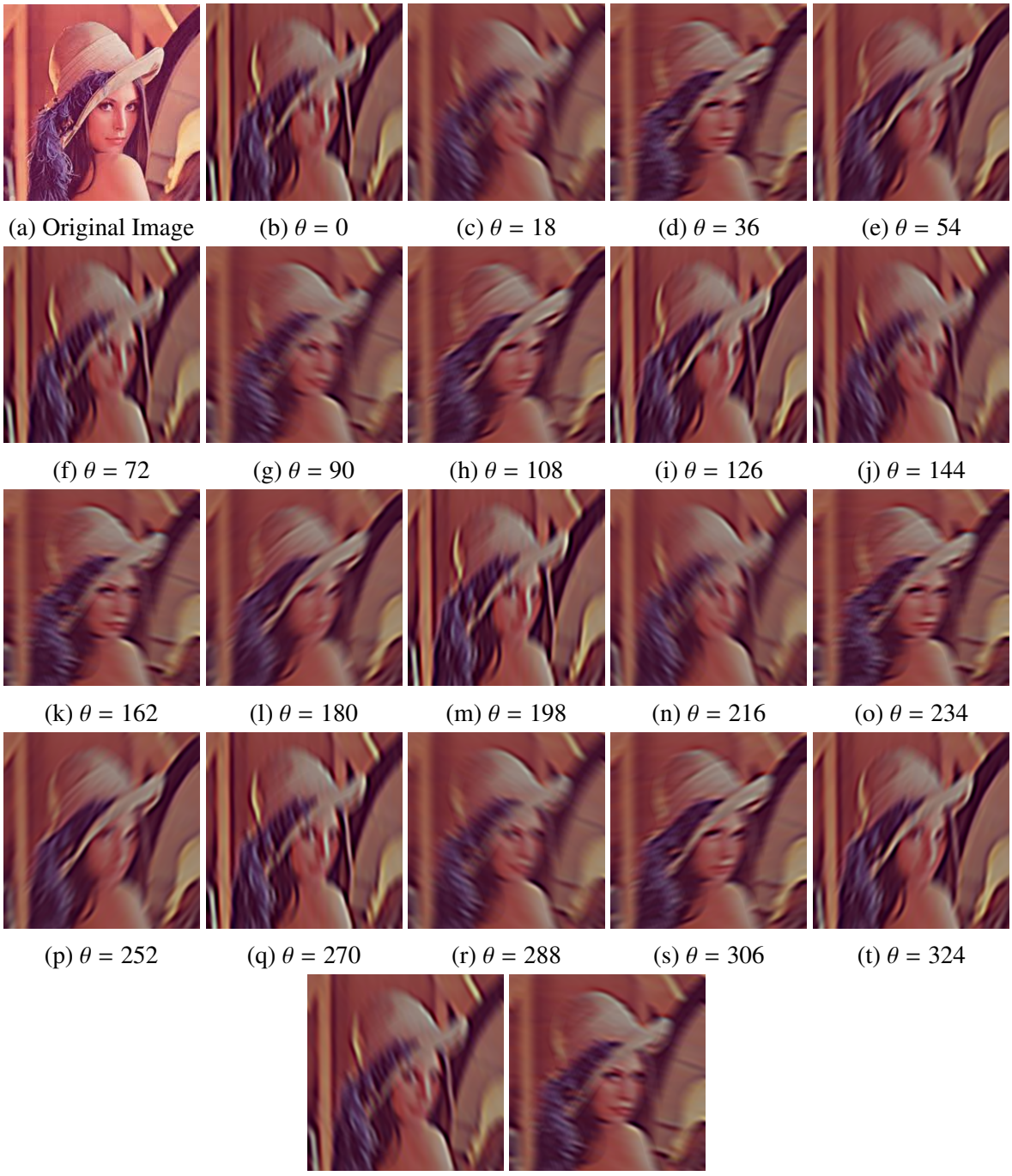
Theta, θ, is a state within the sinusoidal wave that shows different orientations of the filter. Traversing through this variable through the waveform will provide different features and ex-

tractions from the image. In the Figure 1.7, we can see a "circling" effect. The filter is finding the features and extractions from different angles based off of the theta. This is the most important and influential variable. Some features may not be found, or will not be strongly visible in certain angles, however, other angles may expose such features. By combining the different extractions of the image, the features can be exposed more. We will see the effect of this later. As we can see from Figure 1.6 we can see that the kernel appears to be rotating on itself which will allow the filter to grab and detect edges on certain frequencies of the waveform.



(a) $\theta = 0$ (b) $\theta = 45.0$ (c) $\theta = 90.0$

Figure 1.6: Given an input image of 100×100, the Gabor kernel provided is found using the following parameters σ, λ, γ, and ψ that are set to $[\sigma, 10.0, 0.5, 0.0]$ respectively, where $\sigma = 0.56\lambda$ [18].



(a) Original Image (b) $\theta = 0$ (c) $\theta = 18$ (d) $\theta = 36$ (e) $\theta = 54$

(f) $\theta = 72$ (g) $\theta = 90$ (h) $\theta = 108$ (i) $\theta = 126$ (j) $\theta = 144$

(k) $\theta = 162$ (l) $\theta = 180$ (m) $\theta = 198$ (n) $\theta = 216$ (o) $\theta = 234$

(p) $\theta = 252$ (q) $\theta = 270$ (r) $\theta = 288$ (s) $\theta = 306$ (t) $\theta = 324$

(u) $\theta = 342$ (v) $\theta = 360$

Figure 1.7: Given an input image of size 220×220, we change the theta starting from row 1 where $\theta = [0, 18, 36, 54]$, row 2 where $\theta = [72, 90, 108, 126, 144]$, row 3 where $\theta = [162, 180, 198, 216, 234]$, row 4 where $\theta = [252, 270, 288, 306, 324]$, and row 5 where $\theta = [342, 360]$ from left to right. The parameters $ksize, \sigma, \lambda, \gamma$, and ψ are fixed at $[15 \times 15, 3.0, 10.0, 0.5, 0.0]$ respectively. Notice that the image is "rotating" around the center point of the image.

Wavelength of Sinusoidal factor, λ

Lambda, λ controls the size of the wavelength within the sinusoid. This controls the width of the bars, from the Gabor filter. Having a larger λ will have a wider bar, while a lower value will result in a more narrow bar. As a general guideline, the wavelength value should be smaller than one fifth of the input image's size. From the following Figure 1.9, we can see that as λ gets to a larger value, the image becomes more clear. It appears less segmented due to the filter being larger. We can see in Figure 1.8 that as λ increases, the kernel becomes larger [18].

(a) $\lambda = 5.0$ (b) $\lambda = 10.0$ (c) $\lambda = 15.0$

Figure 1.8: Given an input image of 100×100, the Gabor kernel provided is found using the following parameters σ, θ, γ, and ψ that are set to $[\sigma, 0.0, 0.5, 0.0]$ respectively, where $\sigma = 0.56\lambda$ [18].

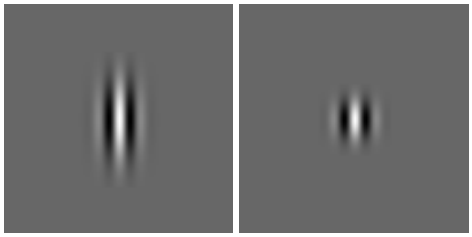
(a) Original Image (b) $\lambda = 3.0$ (c) $\lambda = 4.5$ (d) $\lambda = 6.0$ (e) $\lambda = 9.0$

Figure 1.9: Given an input image of size 220×220, we change λ where $\lambda = [3.0, 4.5, 6.0, 9.0]$ from left to right. The parameters $ksize, \sigma, \theta, \gamma$, and ψ are fixed at $[15 \times 15, 3.0, 0.0, 0.5, 0.0]$ respectively.

Ellipticity or Spatial Aspect Ratio, γ

Gamma controls the height of the filter. As the value gets lower, the filter approaches the size of a pixel, while a larger value will span the filter to the size of the image. Figure 1.11 gets less blurrier the higher the value. This resembles the kernel behaviour in Figure 1.10 where

the kernel size is smaller in height as the value increases. This would mean that as the kernel slides through the image, more information can be gathered since the window is sliding more often.



(a) $\gamma = 0.5$ (b) $\gamma = 1.0$

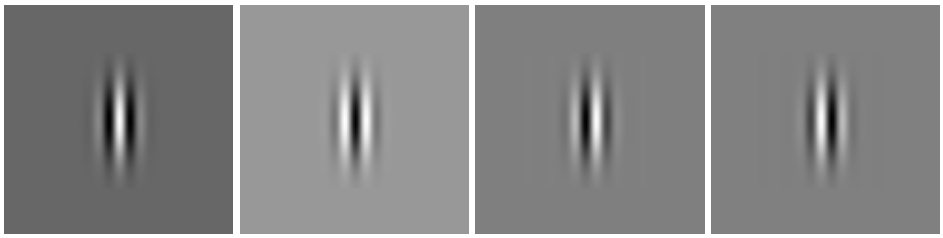
Figure 1.10: Given an input image of 100×100, the Gabor kernel provided is found using the following parameters σ, θ, λ, and ψ that are set to $[\sigma, 0.0, 10.0, 0.0]$ respectively, where $\sigma = 0.56\lambda$ [18].

(a) Original Image (b) $\gamma = 0.2$ (c) $\gamma = 0.4$ (d) $\gamma = 0.6$ (e) $\gamma = 0.8$

Figure 1.11: Given an input image of size 220×220, we change γ where $\gamma = [0.2, 0.4, 0.6, 0.8]$ from left to right. The parameters $ksize, \sigma, \theta, \lambda$, and ψ are fixed at $[15 \times 15, 3.0, 0.0, 10.0, 0.0]$ respectively.

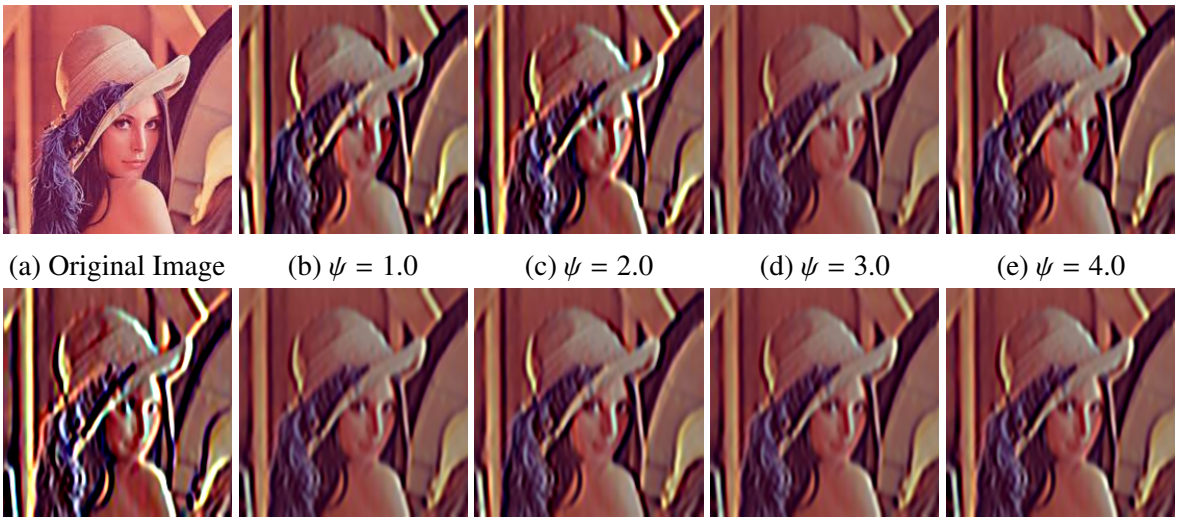
Phase Offset, ψ

Psi, ψ, is usually not used, however, it still has some uses. As you can see from Figure 1.12, the kernel "bands" are shifting either left or right. Because Gabor filters are used as a bank of filters, finding the number of filters in the bank generally nullifies off-setting the frequency of the sinusoidal.



(a) $\psi = 0.0$ (b) $\psi = 180.0$ (c) $\psi = -90.0$ (d) $\psi = 90.0$

Figure 1.12: Given an input image of 100×100, the Gabor kernel provided is found using the following parameters σ, θ, λ, and γ that are set to $[\sigma, 0.0, 10.0, 0.5]$ respectively, where $\sigma = 0.56\lambda$ [18].



(a) Original Image (b) $\psi = 1.0$ (c) $\psi = 2.0$ (d) $\psi = 3.0$ (e) $\psi = 4.0$

(f) $\psi = 5.0$ (g) $\psi = 6.0$ (h) $\psi = 7.0$ (i) $\psi = 9.0$ (j) $\psi = 12.0$

Figure 1.13: Given an input image of size 220×220, we change ψ where $\psi = [1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 9.0, 12.0]$ from left to right. The parameters $ksize, \sigma, \theta, \lambda$, and γ are fixed at $[15 \times 15, 3.0, 0.0, 10.0, 1.0]$ respectively.

1.2.4 Relationship between Gabor Filter and CNN Initialization

Because the Gabor filter is able to find edges and features at different frequencies, the Gabor filter can be used as a filtering method for datasets or to the extent of using them within the CNNs since they share the same behaviour of extracting such features. The large advantage of using Gabor filters is that the CNN does not need to learn the Gabor filter but instead is just given the filter as a starting place. This implies that the network can save training time since the network has a strong starting place. Note however, that this benefit in training time

may not apply in all datasets, since the search process required in finding the right Gabor filter parameters can sometimes swamp the savings in training time.

Since AlexNet, researchers have been trying to find many ways to improve the speed and/or accuracy of these networks. There are many different ways that a CNN can be improved upon. Techniques and strategies such as pre-filtering images before feeding the data into the CNN [10], altering layers or changing layer structures such as dropout layers or pooling layers [7], finding ways to decrease or increase the number of parameters or finding parameters that allow a CNN to converge efficiently, using and/or manipulating different types of filters (static or dynamic).

With the rise in usage of Gabor filters, we will discuss the idea of using and manipulating Gabor filters within the first layer and expanding to other layers within a network.

New strategies such as dropout [7] [12] have been used in increase accuracy for CNNs. Dropout is a tool used when training CNNs to drop dependencies between neurons in the network. When these dependendices are dropped, the CNN is less likely to overfit on its training data. Overfitting is when the model accuractly predicts its training data but when exposed to unseen data, the CNN struggles or fails to identify the image. New structures of CNNs were formed based off of AlexNet and eventually formed new networks of their own, such as ResNet [6]. Researchers have been trying to find new ways for improving neural networks using different and new initialization methods. The first paper linked to Gabor filters within neural networks was created by Caldern et. al. [2]. Their proposed network was very similar to a typical AlexNet [12] whereas the biggest change was replacing the initial (first) layer with a Gabor layer. By including their 'boosting' method, they had achieved a 0.68% misclassification accuracy; impressive for a network created in 2003.

For the next 14 years, there was unnoticed research done using Gabor filters within CNNs. Then came Mr. Sarwar in 2017 [19] who found and rediscovered that using Gabor Filters within the CNN architecture is beneficial by obtaining similar results in terms of accuracy with a large benefit of being faster computationally and more efficient according to their efficiency

algorithm [19]. Ozbulak in 2018 [16] used Gabor filters with the commonly tested MNIST, CIFAR-10, and CIFAR-100 datasets (Section 3.1). His results using a basic AlexNet-like CNN architecture scored 99.25%, 75.73%, and 28.55% test accuracies respectively. In comparison, they had compared their results to a common initialization method for their first layer using the Xavier distribution and the resulting accuracy they obtained had scored higher than the Xavier distribution method [16].

Ozbulak [16] created his Gabor filter bank using a grid search method spreading the Gabor filters across a range of the parameters $(\sigma, \lambda, \theta, \gamma, \psi)$ across 96 filters, his simple grid search did not fill every possible variation, but was enough to provide a boost in training and testing accuracies. This gave insight on whether we can improve once more that the Gabor filter can be tuned further by tightening the ranges of the parameter intervals.

As a solution to improve on the Gabor filter initialization, this thesis is about finding the ideal parameterization of the Gabor filter, finding ideal ranges of these parameters, and explaining how each parameter impacts the results. Additionally, instead of using the grid search method for the Gabor filter, by using what Bergstra found [1], we will describe a simple method for randomizing the distribution of the Gabor parameters to see whether randomization of the Gabor filter bank is an improvement.

1.3 Goals

In this thesis we will show the following:

1. Confirm that the Gabor filter as a substitution for the first layer is powerful enough to extract features in a dataset.

2. Show the difference between using a simple or complex dataset using the Gabor filter bank in the CNN as the network of choice. We design experiments to include changing how many filters we include for the Gabor filter bank for the first layer.

3. Show how each parameter within the Gabor filter impacts the training within MNIST and CIFAR-10. The parameters, θ, ψ σ, λ, and γ can be tuned but we show they must be linked to a range. We show that the Gabor parameters have independent effects during training.

4. Show that the Gabor filter bank used in the CNN's first layer is just as powerful as training with a non-Gabor filter bank CNN.

5. Show that the Gabor filter bank inserted in the first layer of a CNN is able to train faster in the early phases of training than other notable methods like an Xavier initialization for layers in the network.

6. Show the difference between using a random-search or a grid-search for the Gabor filter bank and how it can be used within a CNN. Show how using the initialized Gabor filter bank can train on multiple datasets.

7. Show how dropout impacts the CNN using a Gabor filter bank in the first layer for a CNN.

Chapter 2

Related Work

2.1 Brief History of CNN Model Breakthroughs

This section will describe some of the known CNN structures that have improved the error rate of analyzing ImageNet beginning from 2012 to 2018. The networks will be briefly summarized and described to point out the main contribution to a better neural network structure that we use today for image processing.

The first notable CNN implemented to handle very large datasets (ImageNet) was called AlexNet which was created by Alexander Krizhevsky. He and his team gave new meaning to CNNs with his outstanding win for the ILSVRC-2012 challenge with a top-5 error of 15.3%. A top-5 error rate is the rate at which the labelled image is classified correctly within 5 predictions. Having a lower error rate is better. Krizhevsky and his team were able to do this through new breakthroughs such as the use of dropout layers [12], using rectified linear units (ReLU) instead of typical tanh functions for neuron activations, and data augmentation techniques in order to reduce overfitting in their model.

For their dropout layers, they found that it was most effective when using the dropout layers towards the end of their network. In the first and second fully connected layers of their network, it was used with a 50% probability to zero out hidden neurons. For a neuron to

become activated, it was found that by using the rectified linear units (ReLU) instead of the equivalent function *tanh*, training times could be reduced by up to factor of 4 as they have shown [12]. A ReLU looks like the *tanh* function that has mapped all negative values to 0, while positive values are mapped to the same positive values. With their data augmentation techniques, they created duplicate images from translations, simple horizontal reflections, and color calibrations on images. This allowed their model to achieve a better top-5 error rate by about 1%. These newer techniques used lead them to receive first place in their competition beating the second place's top-5 error rate of 26.2%.

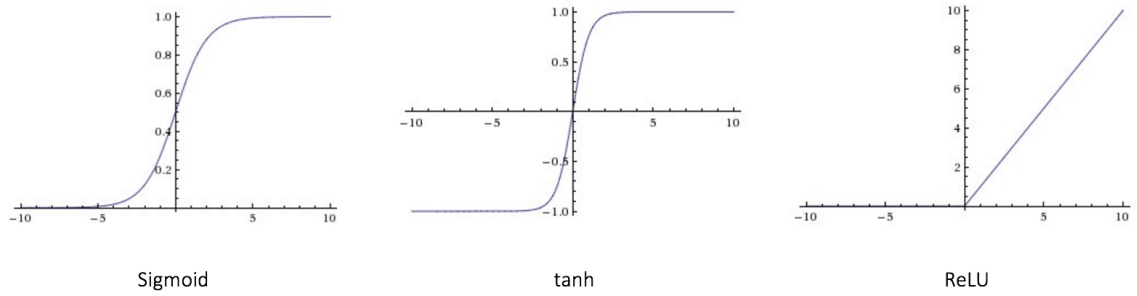


Figure 2.1: Activation functions of a Sigmoid, *tanh*, and ReLU that are used after each convolutional layer within a CNN.

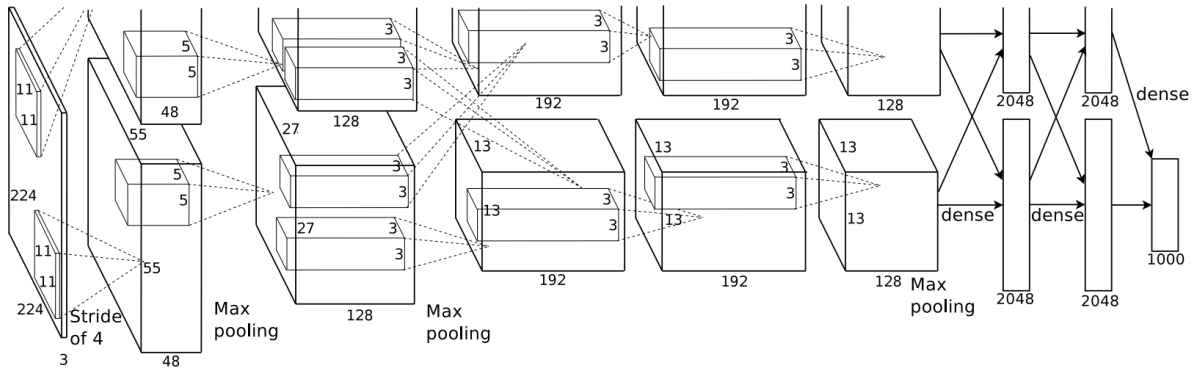


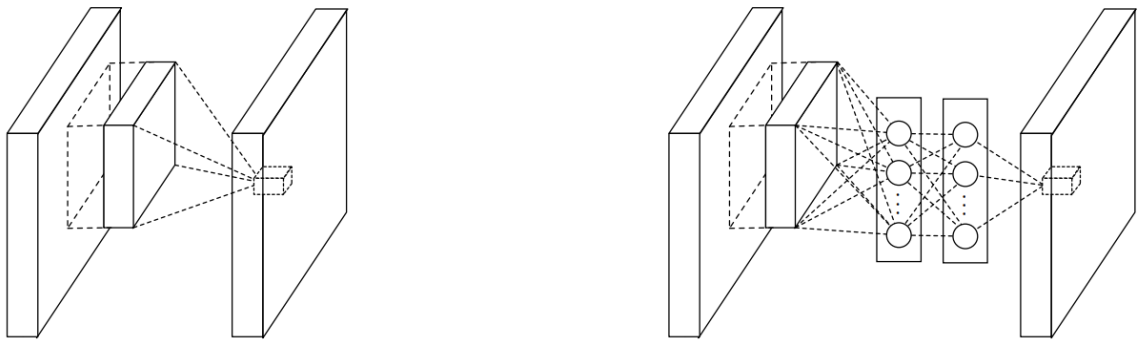
Figure 2.2: AlexNet architecture [12].

CNNs did not really become feasible until this breakthrough. This paper was likely the foundation or beginning point where CNNs were finally fast and/or efficient enough for people to use. The error rates do not seem to sound impressive by today's standards but AlexNet was

noted with praise because of the error rate gap between previous models versus the convolutional neural networks. AlexNet can be thought of as the beginning of CNNs for this reason. Networks created and modified in later years are using AlexNet as the foundation of where to start. As we will see in other works, either slight modifications or newer breakthroughs will be used on top of what AlexNet architecture.

It is important to note that with new techniques, it can drastically reduce error rates of such large datasets. Since then, researchers have been improving and creating new architectures that were based off of the AlexNet while using these new techniques or altering them.

Lin, Chen, and Yao [14] proposed the idea of creating small networks within the CNN layers in 2013. Specifically, they claimed that providing the layers within a typical CNN with another network, which was an MLP, resulted in faster and better scoring benchmarks on the datasets of CIFAR-10. CIFAR-100, and MNIST. Their claim was that using typical CNNs' simple linear models do not provide enough feature extractions, which they referred to as abstractions. It was better to provide a non linear model on top of the feature extraction part of the layers to provide better results. They created what they called "Network in Network" or NIN which adds multiple non-linear activation functions that is mapped from a local patch's input. The MLP used shared all its information for all of the local receptive fields of the input.



(a) Linear convolution layer (b) Mlpconv layer

Figure 2.3: NiN multilayer perceptron (MLP) [14].

With new techniques to work around or with CNNs, training times can result in better

scoring accuracy and speed for completion of training. The authors who created NIN reinforced that using dropout will reduce overfitting and increase accuracy of training data. It is worth noting that feature extraction is very important for the performance of the network and how we find these features can make a very large impact on the end result. The authors claimed that feature extractions are generally non-linear and by using non-linear filters over linear filters, higher abstractions could be found. Higher abstractions are better for the CNN overall since it extracts more information and different variations of the input.

Gabor filter are used to find different variations of the same input map. The Gabor filter as described previously can extract the input space at different angles which can provide features that linear filters may not all see. Although linear filters can extract information, one must use many linear filters to find the variations of an input.

Lin, Chen, and Yao [14] stated that using non-linear functions on the inputs result in higher abstractions because the abstractions found are variations of the input. Using a Gabor filter on top of this (or replacing these non-linear functions) is likely to produce these higher abstractions because of the nature of the Gabor function. This is a promising find that Gabor filters can naturally produce such variations and is likely to work well with CNNs' inputs since they can be quite non-linear.

Sermanet [20] furthered AlexNet by implementing what they called a sliding window over multiple scales throughout the network. Submitting their network, that they called OverFeat, to the 2013 ImageNet challenge won them first place for localization and detection, and fourth place for classification. Their resulting top-5 error rate was 14.2% in the competition and post competition after some minor tweaks, resulted in a 13.6% top-5 error rate.

Sermanet focused on their novel idea of sliding a window at multiple scales. Although, not a new invention for image processing, it was never used in CNNs since OverFeat. The sliding window is essentially a linear filter applied to the entire image. This filter is then used at multiple scales to provide the network with more images. Another novelty they used was the idea of finding the areas of where objects are located and provide a bounding box over

the region. Combining both of the ideas mentioned provided the network a large amount of evidence for their training and testing. The network was very similar to the original AlexNet using the same number of layers, and the dropout novelty to reduce overfitting.

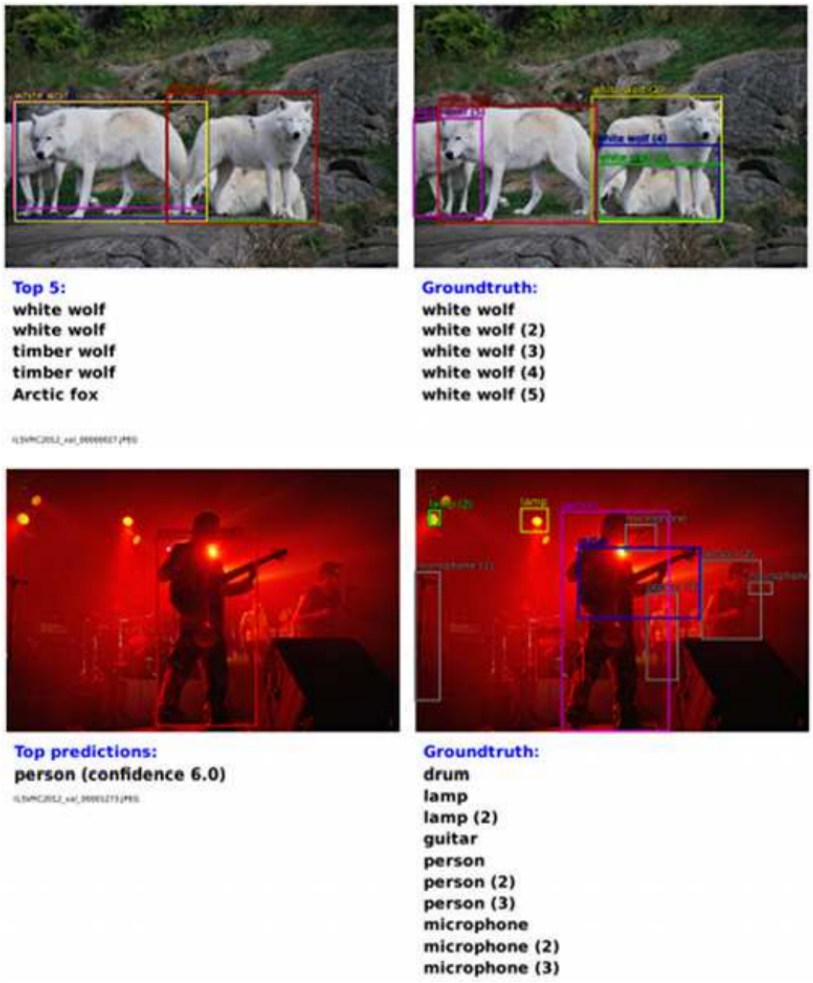


Figure 2.4: Examples of bounding boxes used from OverFeat [20]

One year later after seeing the ground breaking AlexNet reach a high scoring classification on a very large network of over a million images of a thousand classes (ImageNet), OverFeat focused on improving the architecture just shortly after. Using a similar architecture as AlexNet, they applied ideas brought from the past to improve the network's accuracy. Because there were more operations done due to the sliding window at multiple scales [20], it did not necessarily speed up the network, but it did improve the accuracy, i.e., the top-5 and top-1 error rates. This change to improve the accuracy was done internally within the CNN for each of

the layers but it did mention that by helping isolate specific objects within the data using these bounding boxes, the network was able to compute on better data after being pre-processed. The idea of pre-processing such data reduced the computational complexity and resulted in a lower scoring accuracy.

With OverFeat's novel approach to using a sliding window over multiple scales throughout the network and applying bounding boxes to objects within an image, it increased the accuracy of the network winning them top places in the 2013 ImageNet challenge. By finding such objects with their bounding boxes, computation throughout the network done on these bounding boxes allowed for lower scoring top-5 and top-1 accuracy.

Simonyan and Zisserman in 2014 [22] dwell into the idea of having even deeper nets than the previous winners of the ImageNet competition. The main contribution that these authors have done was to find whether an increased amount of layers within a network resulted in a better performing network. It is worth noting that this architecture scored first in localization and second in classification for the 2014 ImageNet challenge. To add to their contribution, they also found that by using a smaller 3×3 convolutional filter (or receptive field) across the entire network, the network had performed better as well. Although the amount of layers within their network increased the number of parameters, it was only a mere 8% increase from 11 layers to 19 layers (a difference from 133 million parameters to 144 million parameters.

To keep their architecture and novelty focused on just the layering of their convolutional neural network, they implemented their architecture that was similarly structured to previous state of the art winners of the ImageNet competition. As well, the training and validation was done in a similar manner. One thing that was noted to reduce the training epochs in comparison to previous ImageNet winners, was that they used pre-initialized training weights for their deeper networks. They had used the weights learned from their first network with 11 layers and pre-initialized their other deeper architectures' first couple of layers and the fully connected layers.

Their research notes that although increasing or decreasing convolutional filter sizes is not

new, using this method along with deeper networks is. The result of their findings show that by having more layers with smaller receptive fields, they are able to produce a better performing network in comparison to the more shallow and larger receptive field networks.

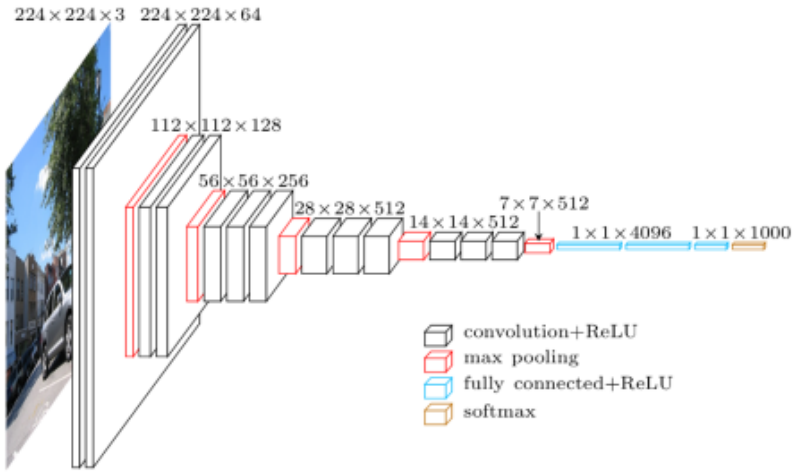


Figure 2.5: A visual representation of VGGNet-16's architecture.

With a simple idea of changing one aspect of a network's architecture, they are able to achieve better performing networks. Simonyan and Zisserman [22] found that by using pre-initialized layers, the network performed at a faster rate. By changing their initialization strategies, they are able to achieve their results faster and better with their deeper networks. implementing such tactics for initialization seems to lead to either faster training times that also improve performance.

The result of changing the depth of a convolutional neural network lead to breakthroughs for top-1 and top-5 error rates. Placing first and second for localization and classification respectively in the 2014 ImageNet competition using their findings of smaller convolutional filters (receptive fields) and consequently the initialization strategies implemented have shown massive improvement over previous state-of-the-art (SOTA) networks used in previous competitions.

Furthering the research done on deep neural networks, Szegedy [24] focused on the idea of even more layers in their network, while also implementing Lin's work for having a network in

a network. Their main goal with their network, which they called GoogLeNet (aka, Inception), was to optimize convolutional neural networks such that it reduces the amount of resources for computation as much as possible. They kept in mind that "in practice, the computational budget is always finite, [and] an efficient distribution of computing resources is preferred to an indiscriminate increase of size, even when the main objective is to increase the quality of results" [24]. What this means is that by focusing on reducing computation, neural networks can ultimately become better by being faster with less parameters. The GoogLeNet that was submitted for the 2014 had outstanding results that had 12× less parameters than AlexNet (submitted in 2012). Not only did GoogLeNet have less parameters, but a higher scoring top-5 accuracy of 6.67%.

The motivation to better networks was the fact that resources, i.e. GPUs, are finite. GPU architectures have a hard time computing non-uniform, sparse data so Szegedy tried to link having dense data for their network but at the same time keeping the data sparse in order to keep parameter numbers down. Dense data is easier to compute while sparse data is more intensive to compute due to the constant overhead of lookups and cache hits/misses for locating data. This was the definition of the Hebian principle: neurons that fire together, wire together. Combining this with "Inception" modules created the GoogLeNet.

The inception module is what allowed GoogLeNet to reduce their parameters. By clustering local data with different filter sizes and reducing the dimensionality (when needed), the network was able to grow at each stage without having a blow-up in computational complexity. Using the filter sizes of 1×1, 3×3 and 5×5 (by designer's choice) for local regions to larger patches, were used to avoid patch alignment issues. It was found that by using 1×1 filter reduction before the more expensive 3×3 and 5×5 convolutions, the complexity was reduced while keeping data as sparse as possible.

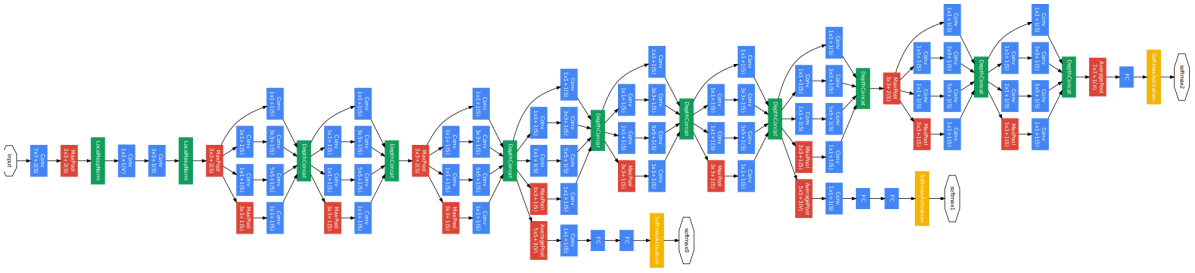


Figure 2.6: GoogLeNet's architecture [24]

Following through with previous networks novelties, GoogLeNet also found that dropout was used throughout and was essential to their networks high-scoring results. Concurrently, using Lin, Simonyan and Zisserman's VGG-Net idea of having a network within a network and having more layers respectively, gave insight to the inception model of going even deeper by adding additional layers and modules to the neural network for a deeper and wider network.

Although GoogLeNet did not change anything for initialization strategies, it was worth noting that their network beat the R-CNN done by [5], where they focused on locating "regions of interest" and then feeding these regions into a general CNN. There are many areas in which we can improve the network, whether it is pre-processing data, or within the network itself. Using different strategies, new novelties and adding them on top of each other only makes networks better and faster.

Reducing the computation complexity while maintaining similar or better results is the absolute goal for all researchers finding ways to better CNNs. GoogLeNet has made a good observation that resources are finite and the real goal isn't obtaining a high-scoring accuracy at any cost, but by obtaining these results with minimal resources. By adding more novelties to state-of-the-art networks and continuing previous architectures, it furthers the speed and accuracy of such networks. GoogLeNet, as of 2014, furthered research from previous state-of-the-art networks. As we continue the history of CNNs, new novelties will be added which will show improvements, whether it is minor or major, but ultimately focusing on reducing complexity and maintaining accuracy or improvement on accuracy.

With the recent trend for networks to go even deeper, ResNet managed to go from 2014's

winner, GoogLeNet, of 22 layers to Microsoft's ResNet of 152 layers [6]. It was notable that ResNet won first place for many competitions, most notably the 2015 ImageNet competition with a top-5 error rate of 3.57%.

The trend to go deeper started with the VGG-Net and was shown promising work that by adding more layers, networks are able to achieve better error rates. One question arose which Kaiming He sought to answer was, how much deeper can a network go? They found that although going deeper did achieve better accuracy, there came to a point where adding more depth saturated the accuracy.

In a small test Kaiming He ran using their "plain" 34-layer network [6], He showed that it had a higher training and test error than an 18-layer network. Conversely, with their ResNet network with the respective layers showed the opposite, where the more layers provided lower scoring accuracies. To solidify their findings, as they added more layers, the top-5 and top-1 error rates steadily decreased [6].

ResNet is useful because of its intuitive residual layers. These layers allow the input to "skip" over layers with what they called shortcut connections. This allows the inputs from each layer the ability to skip over layers and output to the next layer. This process is extremely useful because it does not add extra parameters or adds computational complexity to their network. Additional, the parameters (measured in FLOPS(float operations like multiply and add)) that were used within ResNet was only 18% of VGG-Net-19 (estimated at 19.6 billion FLOPS). By doing these skips, it allows for smooth propagation in both forward and backwards directions.

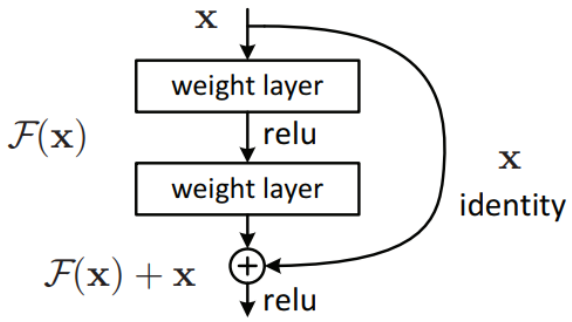


Figure 2.7: A residual module found in ResNet [6]

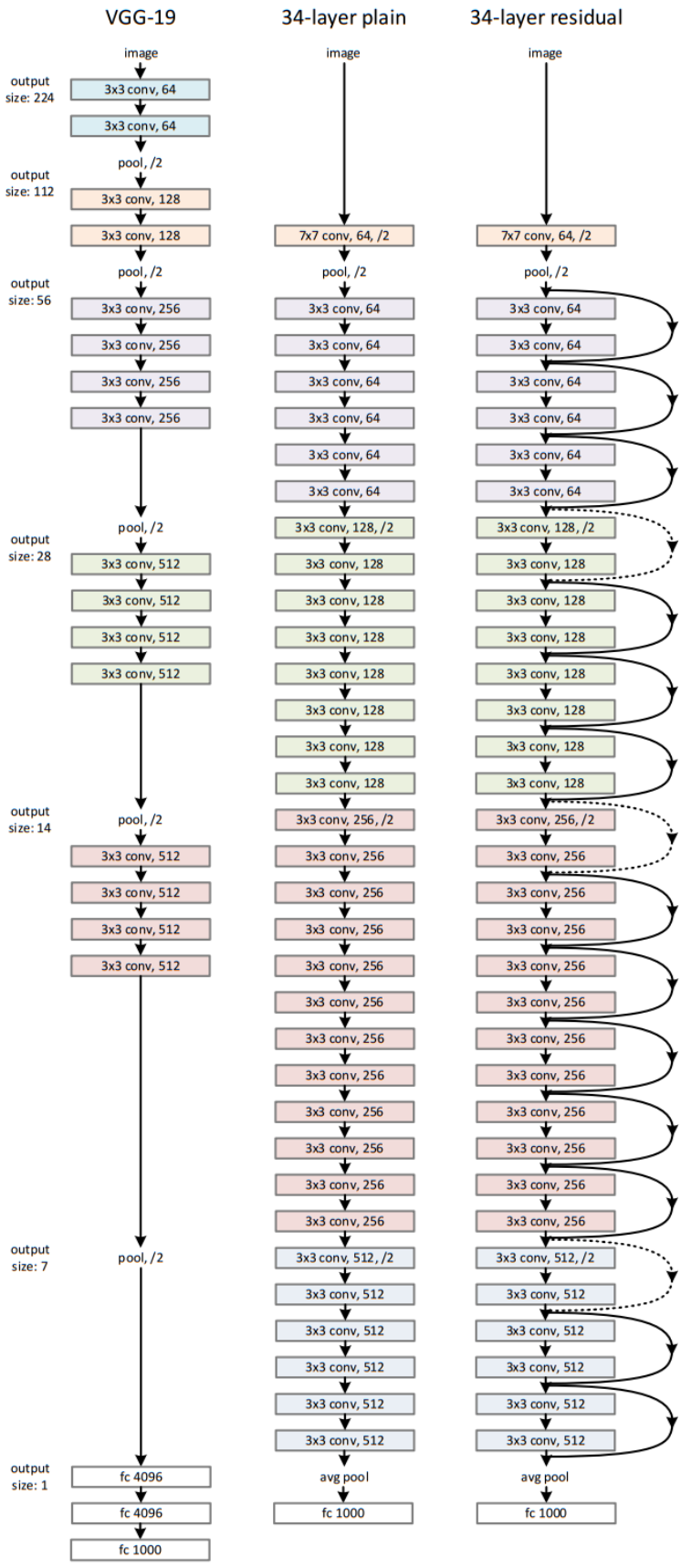


Figure 2.8: A comparison between VGGNet-19 and the 34-layer ResNet. The plain ResNet shows no "skips" being done which is also much slower to converge than the ResNet architecture with these "skips" included [6]

Using the basis of AlexNet up until GoogLeNet as a foundation for improvement, ResNet was the next step to speed up CNNs. There are still many areas of improvement that researchers are still finding today for which networks can gain more speed, reduce complexity, and/or increasing accuracy results. Initialization strategies were not mentioned and were likely not the focus at all for ResNet which gives this aspect of CNNs a large space to look into. By reducing the overhead of inputs, it should advance the field of CNNs even further.

The novelty of the residual layers allowed CNNs to improve the error rates for the future of neural networks. Although Kaiming He ignored any sort of initialization strategy, they used previous state-of-the-art networks as their basis to improve CNNs. Regardless, by finding a smoother transaction between layers within the network using their residual layers, it made CNNs better, and faster.

As the trend to make networks even deeper, DenseNet created networks going up to 250 layers. These deeper networks ultimately reduced training times and with DenseNet's competitive advantage, reduced a large amount of parameters required to achieve their top-5 and top-1 error rates. Although it was not a significant improvement for the ImageNet comparison, it is worth noting that their tests were not optimized and used ResNet's hyperparameters and settings to achieve their results in order to directly compare to the previous state-of-the-art. It was found that although not as accurate, it was more efficient. Smaller datasets used however have shown that DenseNet excels for reduced parameters and best scoring accuracies.

DenseNet brought forward a similar approach to ResNet where the inputs were able to feed forward through these "skip connections" (as discussed in ResNet) which were identity functions that passed inputs along the network without additional computation. DenseNet differs from ResNet in that each layer is able to skip to all preceding layers in the network. This is done in order to alleviate the vanishing gradient problem since it was found that by going deeper within networks, this problem becomes more apparent. Another difference that DenseNet compares with ResNet is that for all of the skip connections that can occur, the features learned are concatenated rather than summed. Other advantages of DenseNet are that

features learned can be reused instead of learning redundant features, and strengthen feature propagation to ultimately reduce the parameter list. The parameter list can be shortened by a factor of two times (table 2 of [8]) when comparing to previous state-of-the-art architectures such as Wide-ResNet, and FractalNet when using the CIFAR dataset.

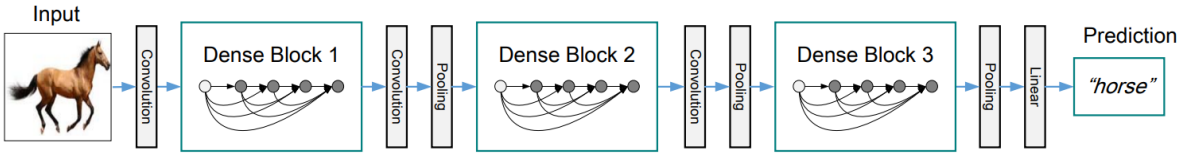


Figure 2.9: Huang's architecture of a small DenseNet architecture showing the skips within each DenseBlock to other layers in the network [8].

Furthering the network by diving deeper in the network for more additional layers resulted in better scores for the top-5 and top-1 error rates. By combating the vanishing gradient problem and keeping in mind training efficiency like GoogLeNet, DenseNet became a top performer in 2017. There was no intent to optimize their network because they directly wanted to challenge other state-of-the-art networks which leads for room to improve.

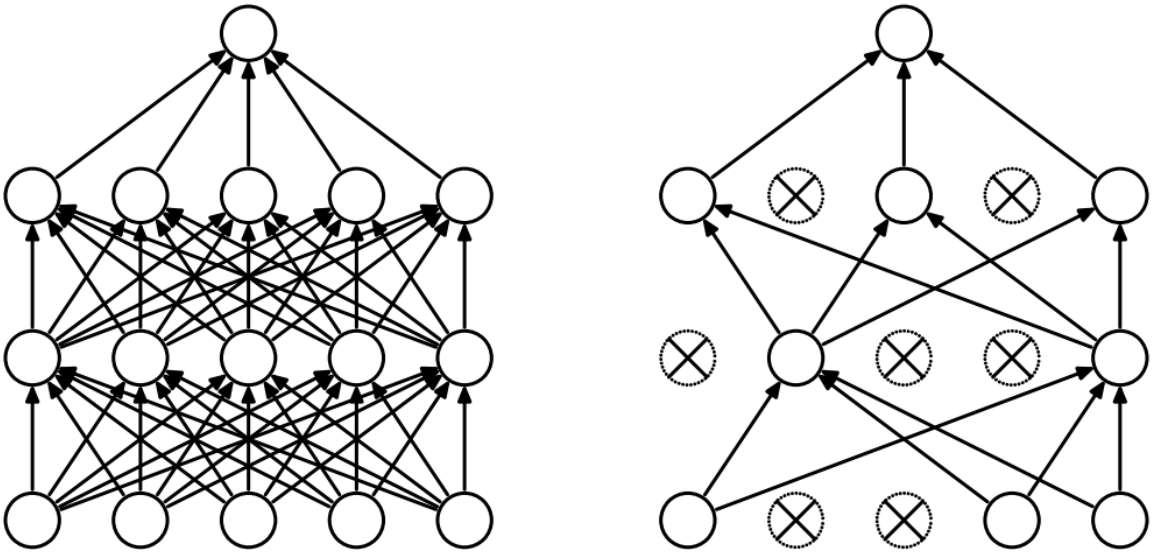
2.2 Noteworthy CNN Breakthroughs

This section will describe the most popular breakthroughs used to improve the quality of life of these CNNs with visualizations, deeper understandings of these networks and also increasing the training times and accuracy of classifying objects of CNNs.

2.2.1 Dropout

Following up from Hinton's original paper [12] to help defend their winning model, AlexNet, dropout was more thoroughly studied to show the effects of dropout. Dropout is the notion of dropping neurons within hidden and visible layers (including input). This is done by temporarily taking out a neuron along with the inputs and outputs (see Figure 2.10). Dropout can also be used as a hyperparameter by tuning the rate of dropping these neurons (between 0 and

1). The main goal of dropout is to reduce overfitting by breaking up dependencies between the neurons within the layers which lowers generalization errors.



(a) Standard Neural Net (b) After applying dropout.

Figure 2.10: Visually showing how dropout works showing the difference between having dropout and not having dropout [23].

Multiple test cases were done, from small datasets, like MNIST, to larger datasets like ImageNet. From these cases, it was found that dropout is a "general technique that is not specific to any domain" and inherently improves training accuracy by breaking up the co-adaptation within the hidden layers of the network. By adding in dropout to the network, it makes each neuron within the layers to become independent on other neurons in the layer(s). It was shown that adding dropout achieves state-of-the-art results and maximizing these results by using a dropout rate between $D = [0.4, 0.8]$, however, when used on smaller datasets, the gains were not as large. As a negative side effect of including dropout, it does add more computation to train such networks and increases the training time, which brings a trade-off between time versus accuracy.

Dropout is used as a tool to improve test accuracy, which was a major finding that is widely used across all state-of-the-art networks in the past and current. Although there is the inherent feature of more training time, the staggering increases in accuracy, especially for larger

datasets, are more than worth the trade-off.

2.2.2 Visualizing and Understanding CNN

Zeiler and Fergus' [26] work for creating a visualization technique was found to be very useful in order to debug their convolutional network. Relating their work to the previous state-of-the-art in 2012, AlexNet, they found improvements just by identifying the image activation maps found from their DeConvolutional Neural Network (which they called DeConvNet). This DeConvNet sole purpose in this work was to project feature activations back into the pixel space. To put it simply, a DeConvNet is the reverse of a traditional convolutional neural network where filters are applied, activations are used (ReLU), and then (optionally) pooling is used.

After studying the images obtained through their DeConvNet, they found multiple findings, all of which were tested and verified. Zeiler and Fergus found that features learned are not just random; and that they are indeed "intuitively desirable properties" such as edges, corners, grids, to faces and other classes as the layers go deeper. Other findings were as simple as looking at their results and seeing a difference in frequencies that were resolved just by changing strides to a lower value. In comparison, AlexNet used a stride, $s = 4$, Zeiler used $s = 2$, and the filter sizes from AlexNet used $F = 11 \times 11$, Zeiler used $F = 7 \times 7$).

Another analysis to test for image transformations and the effects lead to great results. They found that images are less prone to be falsely identified if images are shifted or scaled, however, images that are rotated have a hard time being identified. It was concluded that features that are learned are invariant to translations and scaling. Objects with rotational symmetry are invariant as well [26].

Not only did they use these DeConvNets to study the activations within the network, they studied occlusions within their tests which randomly cropped and grayed out areas in an image. They found that if the main (labelled) object is occluded, the probability to correctly guess the image significantly drops which shows that visualizations correspond to the image structure.

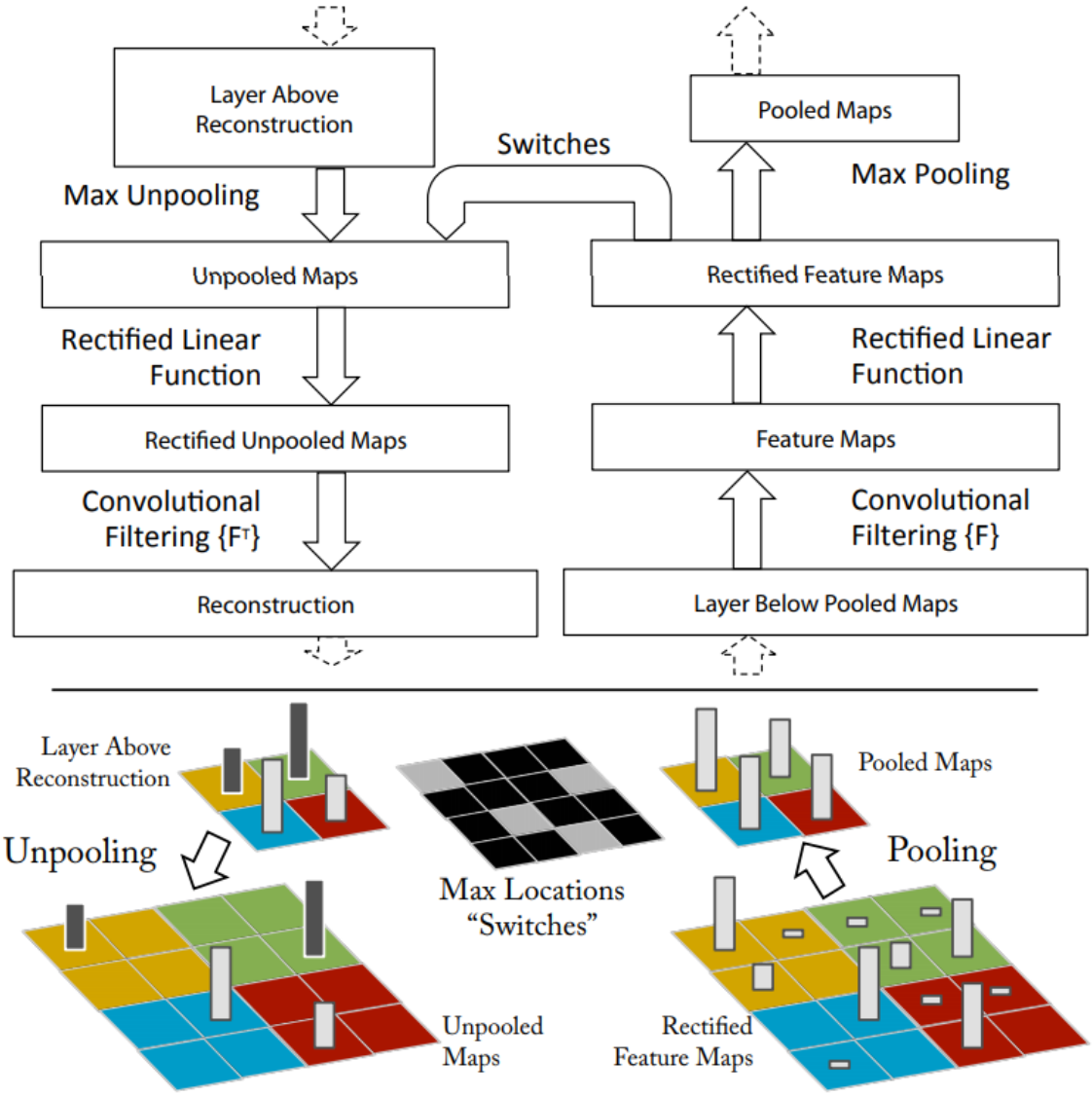


Figure 2.11: Zeiler shows how Deconvolution is done from [27].

Figure 2.12: Visualization of each layers from 1 to 5 of a modified AlexNet [26]. The original images are to the right of the kernel blocks.

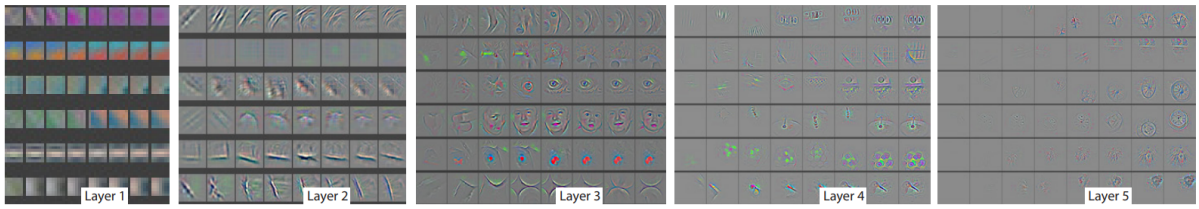


Figure 2.13: Visual representations of features learned through each layer organized by the number of epochs (from left to right [1,2,5,10,20,30,40,64]) [26].

One finding that was very worth going over was when they stated through their tests that the lower layers of the model can be seen to converge within a few epochs. However, the upper layers only develop after a considerable number of epochs(40 to 50), demonstrating the need to let the models train until they fully converge. This is a very good finding because as we learned through the history of CNNs, there are many ways we can better CNNs and providing a head start for the network to progress which can definitely lead to better features, and possibly impact the training time as well. This leads to the question to consider how important is learning within the first layer and whether earlier layers provide more impact than later layers.

2.3 Initialization Strategies for CNNs

This section will describe some of the ways researchers have tried to use to improve the learning phase of training by changing or implementing different ways to initialize the CNN.

2.3.1 Random search for hyper-parameter optimization (2012)

Bergstra and Bengio in 2012 [1] that in general, random hyper-parameter optimization is more efficient and in most cases more accurate than using a grid search. Grid search is very common and is still widely used as a safe way to find ideal parameters but the cost to use this safe method is high computation if the resolution of the parameter list is very fine. Grid search is easy because it is simple and easy to implement and is relatively reliable in a low dimensional

parameter space architecture. However, with a larger dimensional space for parameters, grid search suffers from "the curse of dimensionality". The reason is because adding parameters is exponential when it comes to finding the "right" value for hyper parameters. They also found that when training with a large dimensional space, some of the dimensions, if not most, are useless and offer minimal results to the final product which in turn takes up time and computation.

With random search taking random draws from the space input space as a regular grid, they show that it is not only more efficient, but very simple and easy to implement as well. They found that even with smaller dimension spaces, random search still exceeded grid search results in both low and high dimension spaces. The bonus to using random search as well is that when training with random search hyper parameter optimization, trials done through the iterations can be asynchronous, and be added or removed without consequence, and testing can be stopped at any moment. This is very useful in real situations where a power surge (or other real life issues) fails or more of your trials failed and did not complete.

Through multiple experiments that were done on multiple datasets, Bergstra and Bengio found that although the trials done with grid search and random search needed many trials to find a successful accuracy rate, random search competed with a higher score and required less iterations than grid search. There were only some cases in which random search was less dominant, but not by a large margin. Keep in mind that they had only tested on what is considered smaller datasets in reference to large datasets like ImageNet. Perhaps using this with the Gabor filter bank in [16]'s work, we can see an improvement.

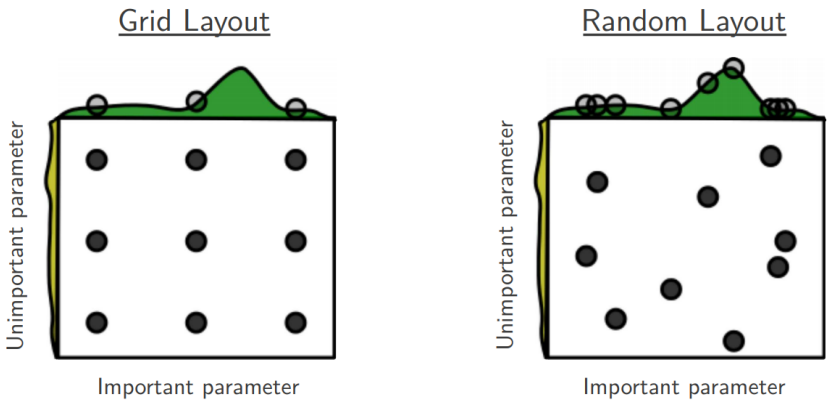


Figure 2.14: An example of how Grid versus Random optimization can affect parameterization [1]

2.3.2 Region Proposed Convolutional Neural Networks (R-CNN)

The R-CNN involves feeding in images that have already been localized and segmented with bounding boxes and therefore passed "good" data to the neural net. The R-CNN found that to produce major improvements in accuracy. This can be thought of as another pre-processing method or initialization method for convolutional neural networks.

In Girshick's first paper [5], he found that by extracting around 2000 region proposals per image using selective search and then warping the image by normalizing its dimensions before feeding into the CNN, it increased the mean average precision (mAP) of the VOC-2012 dataset by more than 30%, achieving a result of 53.3%. *mAP* calculates the average of the maximum precision of different recalls. Precision is the accuracy of an object being correct, while recall is the rate at which you can find the all of the positive objects. The technique to pre-process images before feeding into a network is not novel, but Girshick was able to create this region proposal effectively such that the proposals themselves had a large impact by selecting the most "right" regions.

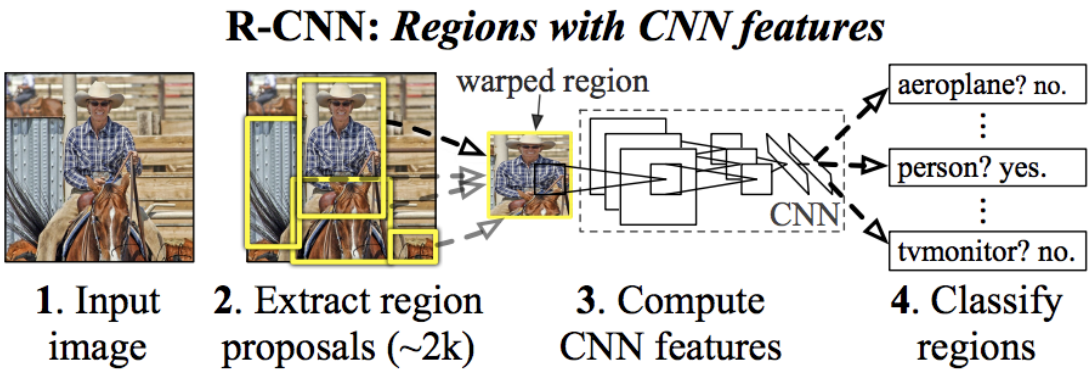


Figure 2.15: High level architecture showing how R-CNN works [5]

In later work, through 2 major iterations, they have reached a new R-CNN which they called, Faster R-CNN. The improvements were to find a new way to reduce the bottleneck of such region proposals. After restructuring, they found that by adding common convolutions to both the region proposals and the convolutional network, they were able to simultaneously predict objects, score them, and feed them through the network. This change resulted in a new high-scoring mAP of 70.4% for the VOC-2012 dataset. The best part, is that because of this change, the average time to process images was about 10ms (compared to 13 seconds on a GPU from their initial inception, R-CNN) and also used only 300 object proposals, 1700 less than their original network.

2.3.3 Median Filtering Forensics Based on CNNs

With the recent rise of CNNs since AlexNet, researchers explored using these networks in order to automatically find features rather than finding them manually. Chen [10] explored using CNNs alongside median filters to improve the field of forensics analysis. They created a CNN network that included the median filter as their first layer in their network. The benefits of using a median filter are for finding non-linearities and preserving edge information–which Gabor filter does as well. By using the median filter alongside the CNN network, image edges and textures can be suppressed and minimized which allows the CNN to learn features much

easier.

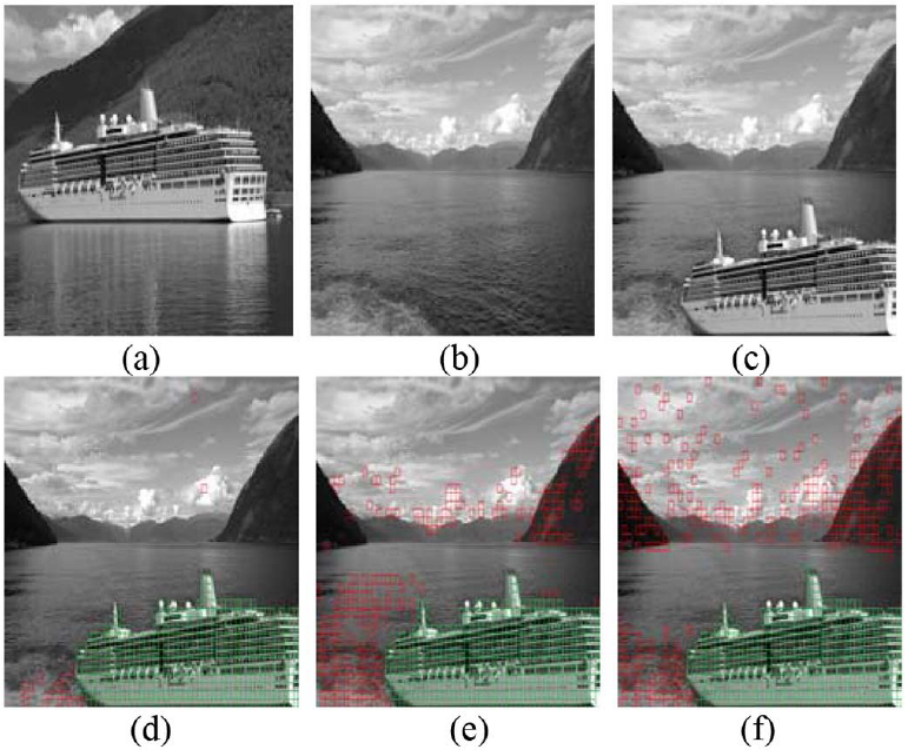


Figure 2.16: Chen (2015) [10] describes this images as (a) an image with a boat that has already been filtered with the median filter; in (b) showing a new image of a different background; in (c) showing the boat from (a) cropped into (b). The green regions show the true positives while false positives are marked in red. In (d) shows the median CNN working well with more true positives than false positives, while in (e) and (f) showing the Global Local Feature (GLF) and Auto Regressive (AR) methods with much more false positives than (d).

They ran two experiments in order to show that a median filter fed into a CNN network results in higher scoring accuracies than without using it. In their first experiment for finding fingerprints for small and compressed image blocks, it was found that by using their modified CNN, it improved the detection accuracy by 9.27%, from using a regular CNN with a result of 77.92% detection accuracy. As a bonus to this experiment, it beat the top 3 state-of-the-art methods for using median filters. In their second experiment for the cut-and-paste forgery detection, their method beat two other conventional methods (GLF and AR), shown in Figure 2.16.

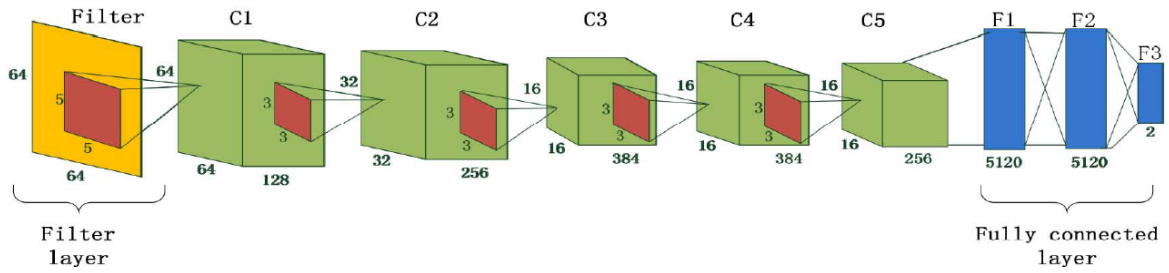


Figure 2.17: Architecture of a median initiated first layer CNN [10]

This new technique to implement a filtering layer for datasets is an ingenious way to isolate better features within the images. By implementing and modifying conventional CNNs and adding in the median filter as a first layer, Chen was able to produce better accuracies when comparing to previous state-of-the-art methods and in multiple experiments. Using Chen's ideas of using a pre-determined filter as the initial layer of the CNN, the next sections will introduce using the Gabor filters within the network, either in one layer or multiple layers.

2.4 CNNs using Gabor Filters

After exploring the major breakthroughs and the history of the CNNs, we will now explore Gabor variations that have been done on CNN networks. From the previous median filtered CNN network and the R-CNN network, we can see that images that are filtered for a purpose and then fed into the CNN can lead to much better results.

In 2003, Caldern [2] proposed networks to explore Dennis Gabor's filter using CNNs 9 years prior to AlexNet. Caldern proposed an early stage of the CNN architectures to add a Gabor feature extracting layer as the first layer within their network. They implement a 5 layer network: the first being their Gabor filter layer, and then alternating subsampling (pooling) layer and convolutional layers, until the final fully connected output layer. On top of this proposed GCNN architecture, they implemented a boosting method that considers the best result among different test cases. The boosting method that was described is another method like dropout that was useful for the dataset to generalize on test data better. The results showed

that from using the MNIST training dataset that their GCNN (with their proposed boosting protocol) achieved a 0.68% error percentage for classifying the incorrect image. This beat LeNet-5's results which gave an error percentage at 0.95%.

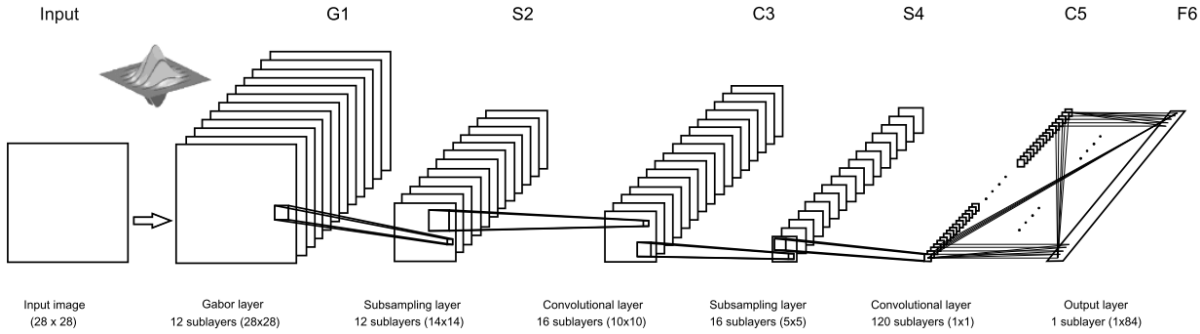


Figure 2.18: Architecture showing how the Gabor filter was used and implemented [2].

We can see that even in the earlier stages of the development of CNNs, Gabor filters can have an impact on CNNs. Although there was only a small difference in error percentage, the CNN architecture's back in 2003 are not nearly as good as they are in their current state (i.e. DenseNet, ResNet, VGGNet, etc.). Caldern created this GCNN in order to utilize Gabor filters benefits of extracting features and empowering these feature detectors through the use of feeding them through a CNN. Over time, Gabor filters alongside CNNs improved accuracy and training time.

Sarwar [19] furthered the research of other similar Gabor CNNs by implementing their own CNN using the Gabor filter in the first and second convolutional layers. The structure of their GCNN was as follows: 2 convolutional layers, each followed by a sub-sampling layer and finally ending off with a fully connected layer. Their proposed architecture was using a Gabor filter bank (which are a set of fixed Gabor kernels) in place of the first layer of their network. They also tested a variation of their network to include the Gabor filter bank as the second layer of their network. Their main objective was to find energy savings in terms of computation time by reducing the complexity of the network by exploiting error resilience.

Backpropagation, gradient computation, and weight updates within the layers are known to be highly expensive in terms of computation. By replacing the trained layers in the net-

work with fixed Gabor kernels, it means that the CNN does not have to use these expensive computation methods within their layers.

Configuration		**Accuracy Loss**	**Energy Savings**	**Training Time Reduction**
1st Conv. Layer Kernels	**2nd Conv. Layer Kernels**			
Trainable	Trainable	–	–	–
Fixed Gabor	Trainable	0.62%	20.70%	9.47%
Fixed Gabor	Fixed Gabor	5.85%	48.28%	42.48%
Fixed Gabor	Half Fixed Gabor & Half Trainable	1.14%	34.49%	22.30%

Table 2.1: Sarwar (2017) [19] showing the accuracy loss, energy savings, and training time reductions from testing on MNIST with 100 epochs using their GCNN architecture with the first and/or second layers replaced with a Gabor filter bank.

When testing on the MNIST dataset, their GCNN architecture found great improvements in terms of energy savings and training time reduction, at the cost of a 0.62% accuracy loss. Using their energy savings calculator that they implemented saved up to 20.70% computations and reducing the training time by 9.47%. These numbers were generated using the Gabor filter bank within the first layer of their architecture. Following this test, they proceeded to test using their first and second layers of their architecture to use the Gabor filter bank. It was found that although the energy savings and training time was reduced (48.28% and 53% respectively), the accuracy loss was far greater at 5.85%. They predicted that with a more complex dataset (i.e. ImageNet), similar numbers would be shown for energy savings and time reduction while impacting the accuracy loss even further. This can be seen in table 2.1.

2.4.1 Gabor Oriented Filters in CNNs

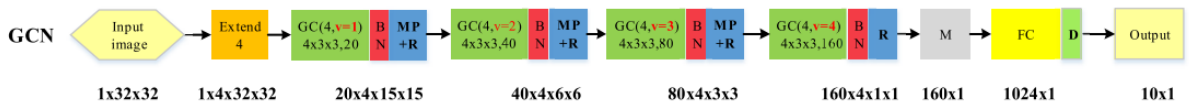


Figure 2.19: Luan (2017) [15] GCNN architecture showing a 5-layer network.

Luan [15] focused their work with GCNNs by implementing their Gabor filter bank on the entirety of the network's convolution layers as shown in Figure 2.19. Their main goal was to find a way to reduce the cost of training datasets by either reducing the training time or the amount of parameters used during the training. The network learns its traditional filters during the convolution phase and then the learned filters are modulated with a Gabor filter bank through an element by element product operation. This process creates what they have called a Gabor of filters (GoF) which are the filters that finds the features within the inputs. This process does not require additional parameters but enhances the features obtained from the dataset. Because of this additional GoF, the weight updating process via backpropagation for the learned filter is the only filter it needs to apply the updates to.

Method	**VGGNet**	**R-110**	**R-172**	**G-R-40**	**G-R-28**	**ORN-40**	**ORN-28**
# of Parameters	20.3M	1.7M	2.7M	2.2M	1.4M	2.2M	1.4M
Accuracy (%)	95.66	95.8	95.88	96.9	96.86	96.35	96.19

Table 2.2: Luan [15] showing the number of parameters used, and the accuracy from the SVHN dataset. G is short for GCNN and R is short for ResNet

Their tests were done on multiple datasets that include MNIST, CIFAR-10, CIFAR-100, SVHN, and more importantly, ImageNet-100. For MNIST, their accuracy using a 3×3 Gabor kernel bank was 0.63% using 0.25M (million) parameters. In contrast, their baseline CNN scored a 0.73% error rate using 3.08M parameters. For their next test using the SVHN dataset (table 2.2), the authors provided a modified ResNet using their GoFs applied. Two networks were created from this: a 28-layer GCNN-ResNet-28 and a 40-layer GCNN-ResNet-40. Both of these new networks created beat VGGNet and ResNet-110 using less parameters and scoring a higher accuracy. A similar pattern comes across on the CIFAR-10 and CIFAR-100 datasets as well as the ImageNet-100 dataset. For more of their results, refer to [15].

Ozbulak in 2018 [16] explored a novel use of the Gabor filter. His objective was to find whether transfer learning could be replaced by Gabor filter initialization instead. What this means is that instead of using pre-trained model and transferring the model to an unknown

dataset then fine tuning it, a Gabor filter bank can play the role of the pre-trained network. It was described in [25] that the low level features learned in the first layer resemble the Gabor filter naturally which can also be seen in layer 1 of Figure 2.12. Using this intuition, Ozbulak created a modified CNN network using a uniform distribution (a sample of a grid search) of Gabor filters as the first layer within the CNN.

MNIST	**Training**		**Test**	
Epoch	**Gabor**	**Glorot**	**Gabor**	**Glorot**
1	87.36	63.01	96.6	94.71
2	97.56	95.85	98.26	97.71
3	98.35	97.55	98.69	98.53
4	98.70	98.18	98.74	98.58
5	98.89	98.54	98.70	98.65
6	99.03	98.71	99.23	98.80
7	99.15	98.90	99.22	99.11
8	99.21	99.03	98.97	98.86
9	99.31	99.13	99.25	99.07
10	99.33	99.20	99.25	99.13

Table 2.3: The following results show the progression of training a custom 5-layer CNN using either the Gabor filter initialization or the Xavier distribution as the initialization method in the first layer using the MNIST dataset [16].

CIFAR-10	Training		Test	
Epoch	Gabor	Glorot	Gabor	Glorot
1	32.44	24.05	45.00	35.95
2	52.50	43.73	59.21	50.28
3	61.77	54.59	62.24	57.59
4	66.72	61.57	66.64	59.10
5	70.60	65.99	69.80	63.54
6	73.36	69.26	70.90	68.93
7	75.62	71.87	73.54	71.51
8	77.33	74.08	74.40	70.92
9	79.26	76.01	74.85	73.34
10	80.50	77.51	75.73	74.22

Table 2.4: The following results show the progression of training a custom 5-layer CNN using either the Gabor filter initialization or the Xavier distribution as the initialization method in the first layer using the CIFAR-10 dataset [16].

CIFAR-100	Training		Test	
Epoch	Gabor	Glorot	Gabor	Glorot
1	2.51	0.87	5.06	0.68
2	7.66	2.13	10.16	2.31
3	11.68	2.92	12.72	3.12
4	14.32	4.48	16.03	5.5
5	17.34	7.15	18.49	7.48
6	20.12	8.47	20.71	9.28
7	22.75	10.04	23.45	10.66
8	24.75	11.66	24.33	12.52
9	26.71	13.39	26.64	13.11
10	27.89	14.68	28.55	15.06

Table 2.5: The following results show the progression of training a custom 5-layer CNN using either the Gabor filter initialization or the Xavier distribution as the initialization method in the first layer using the CIFAR-100 dataset [16].

His results showed on three different datasets, MNIST, CIFAR-10, and CIFAR-100 that there was an improvement of approximately 2%, 1.5% and 13% respectively after running for 10 epochs for each dataset. These differences were compared with the Glorot distribution [16].

It was concluded that Gabor filters are indeed a good replacement for transfer learning. These results can be seen in the following tables 2.3, 2.4, and 2.5

As we progress through this thesis, we will learn more and describe the architecture of Ozbulak's work and his work will be used within our tests to further the usage of Gabor filters. Chapter 3.3 will define the architecture in detail and describe how Ozbulak created his network.

Chapter 3

Methodology

3.1 Datasets

In this section, we will describe the databases used within the GCNN detailed in Section 3.3.

For each database, we created a training and testing set using either the databases recommended splits (MNIST ,CIFAR-10, CIFAR-100) or a common data split ratio of 70% training, 15% validation, and 15% test sets. The training set we used to train the CNNs and the test set was used to report the performance of the CNN on unseen data. we used test data performance to compare the various initialization strategies.

3.1.1 MNIST

The first dataset that will be used and tested on is the MNIST database [13]. This collection of data is a subset of the NIST database and consists of fixed-sized grayscale images containing the numerical digits 0 to 9. The images are of size 28×28 pixels and have 60,000 training images and 10,000 test images.

Figure 3.1: A set of images generated from [13] showing 16 different numerical digits from 0 to 9.

3.1.2 CIFAR-10

The second dataset that will be used and tested on is the CIFAR-10 database [11]. This collection of data was collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The images are of size 32×32 pixels, are in full RGB color (3 channels to operate on), consisting of 10 classes. Each class consists of 6,000 images, for a total of 60,000 images. The training set is 50,000 images, while the test set consists of 10,000 images. Images in the dataset are mutually exclusive from one another which means that there is no overlap between other classes in the picture for complete consistency among training.

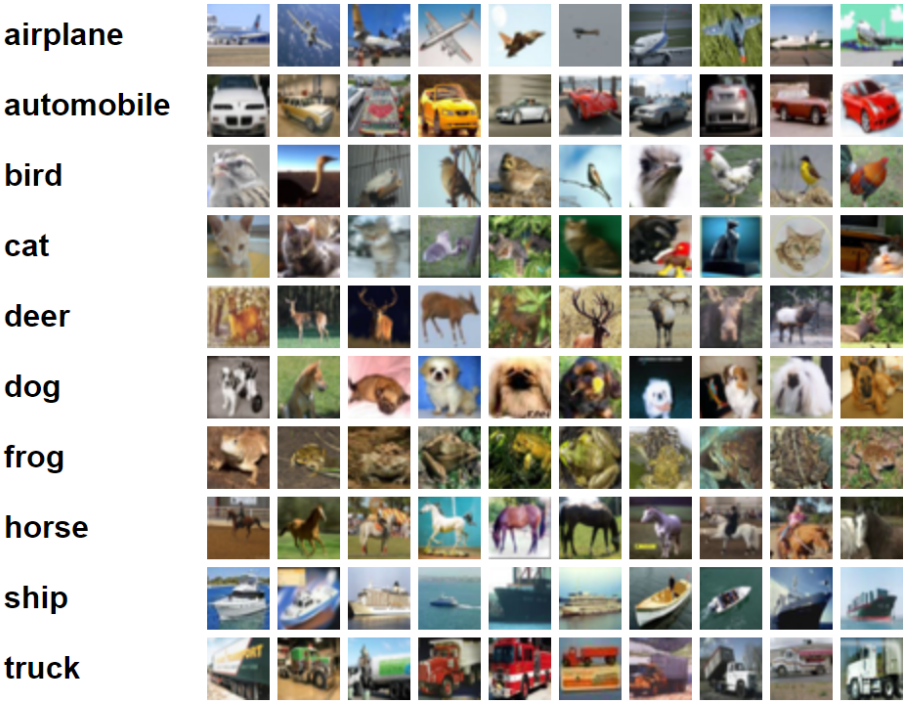


Figure 3.2: Images CIFAR-10 showing 10 different images of 10 different classes [**?**].

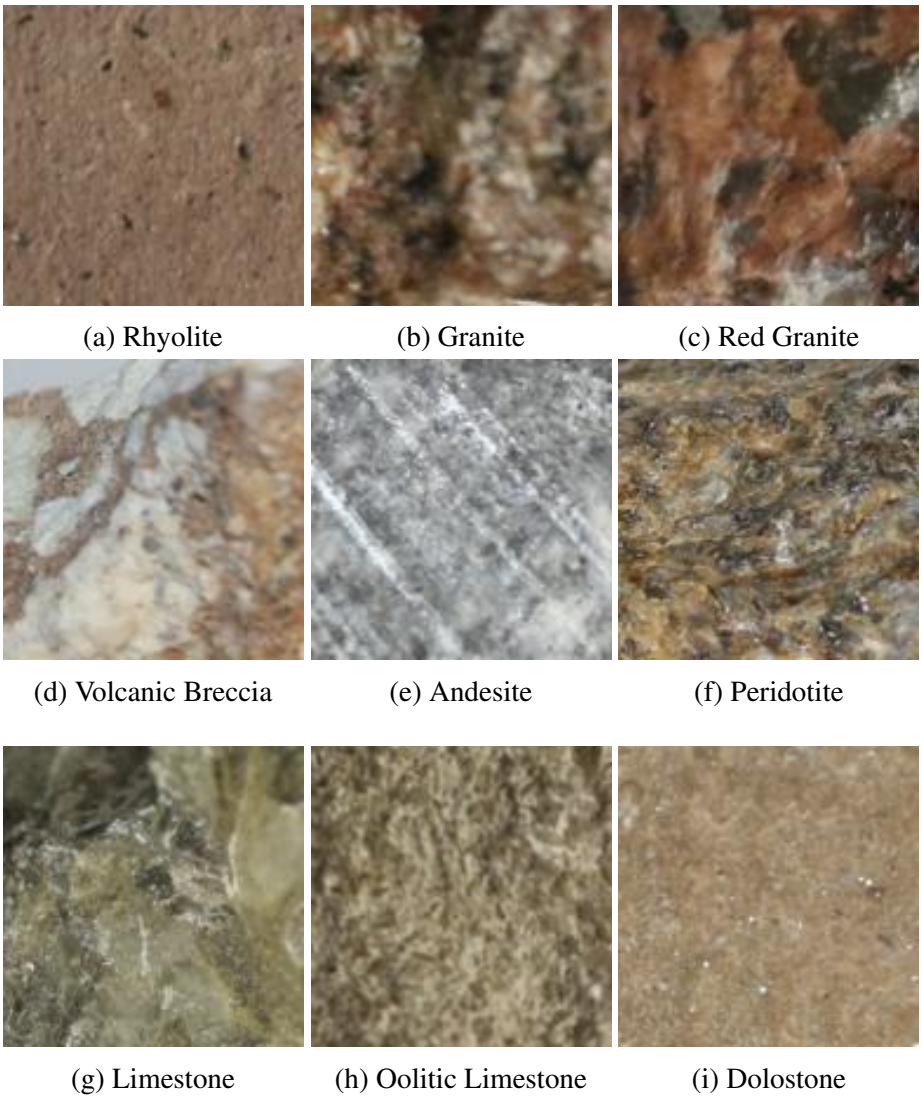
3.1.3 CIFAR-100

The third dataset created by Hinton and Krizhevsky is an extension of the CIFAR-10 dataset, the CIFAR-100 [11]. The only difference is that there are now 100 classes, where each class has 600 images for a total of 60,000 images. 50,000 images are for training, and 10,000 are for testing.

3.1.4 AlexisNet Rock Dataset

The fourth and final dataset that we will use is a custom generated dataset generated by Lei Shu [21]. The images were collected at Western University from a digital camera and an optical microscope. This dataset consists of 9 different rock classes for a total of about 80 images per class. The total amount of images is approximately 700 images. To make up for the very small dataset, Lei applies image augmentation to the original image. The notion of augmentation for images is applying translations, rotations, blurring and scaling to an original image to create

an entirely new image. These images are added to the dataset which increases the total number of Lei's database to 2,233 images. The images were split into a training and testing using a 80%/20% split.



(a) Rhyolite (b) Granite (c) Red Granite

(d) Volcanic Breccia (e) Andesite (f) Peridotite

(g) Limestone (h) Oolitic Limestone (i) Dolostone

Figure 3.3: A simple rock database consisting of 9 different rock types used in AlexisNet described in Section 3.3.2.

3.2 Gabor Filter Initialization

In this section, we will discuss how the Gabor filter bank was created and how it will be used within the CNN structure defined in the next section. We will describe the parameters used and how they are initialized. By default, when using Keras framework, the default initializer for the convolutional layers is an Xavier (or sometimes referred as Glorot) uniform distribution. When we are using the Gabor filter initilization, we are using the Gabor filter bank in place of the Xavier distribution. This method is also used and compared with [16]'s results. Below we will describe how the Gabor filter bank is created using 2 methods, grid-search and randomization. As a reference for a visualization, refer to Figure 2.14.

3.2.1 Gabor Grid-Search Initialization

Parameter	**Range**
ksize	5×5
sigma	[2,21]
lambda	[8,100]
theta	[0,360]
gamma	[0,300]
psi	[0,360]

Table 3.1: Ozbulak shows the ranges and values given for the Gabor filter used in his GCNN [16]. The *ksize* represents the size of the Gabor filter (or sometimes referred to as kernel). The *sigma* represents the standard deviation of the Gaussian envelope. The *lambda* represents the wavelength of the sinusoidal factor. The *theta* represents the orientation of the normal to the parallel strips of the Gabor functoin. The *gamma* represents the spatial aspect ratio and spcifies the ellipticity of the Gabor function. Finally, the *psi* represents the phase offset between the bands.

The idea of finding parameters using grid-search is not a novelty, however Ozbulak thought that it was a good starting place to use it within his custom 5-layer network because of the natural strength of feature extraction while using the filter. The notion of a grid-search is to

find parameters at the cost of brute forcing all possible choices within a test. This can be done by using a fixed interval within the range of a parameter. For example, in a parameter, p, given a range, $R = [0, 100]$, and using an interval value $I = 1$, there would be 100 possible choices to choose from. The Gabor filter has many parameters that can be tuned, and finding the best initialization strategy for the CNN can be time-consuming, and can cost a lot of computation. The parameters that can be tuned are the kernel size, sigma, lambda, theta, gamma, and psi ($ksize, \sigma, \lambda, \theta, \gamma$, and ψ). These parameters are described in Section 1.2.3. Ozbulak randomly selected a finite range for each of these parameters listed in table 3.1.

These ranges were used as the minimum and maximum data points using an interval that was dependent on how many number of filters was used in the first layer. In Ozbulak's case, it was 96 filters (or intervals). Therefore, for each parameter, it was divided into 96 equal intervals and each interval for each parameter created 1 Gabor filter for a total of 96 Gabor filters. This is important to distinguish because it is not a perfect grid search, but a cheap method to find the bulk of the grid search. Otherwise, their would not be enough space in the hardware to store 96 filters with 96 intervals of 5 parameters. Another note to keep in mind is that the Gabor filter's parameter's ranges are dependent on how many filters was used in the first layer. We use this as a study for comparing Gabor filter bank sizes, $F = [32, 64, 96, 128]$. In the next section, for each test case, it will describe how many filters were used to initialize and create the Gabor filter bank.

The grid-search initialization method [16] was used as the preferred initialization method over transfer learning and was trained on MNIST, CIFAR-10, and CIFAR-100. This Gabor filter bank was only used in layer 1 because [19] found that there was a small return of improvement for using the Gabor filter in consequent layers and in some cases reduced the performance of the CNN.

LEI

3.2.2 Gabor Random Initialization

Similar to the Gabor grid-search method described above, the random initialization spreads the parameters values randomly across the ranges per parameter with no specific distribution. Using the same parameter ranges that [16] had used, we can directly compare whether grid-search or randomization is better for initializing the CNN. Using what Bergstra and Bengio had found in their research [1], random-search for parameterization was found to be faster, and in most cases, more efficient when training a network which we will see in the next section.

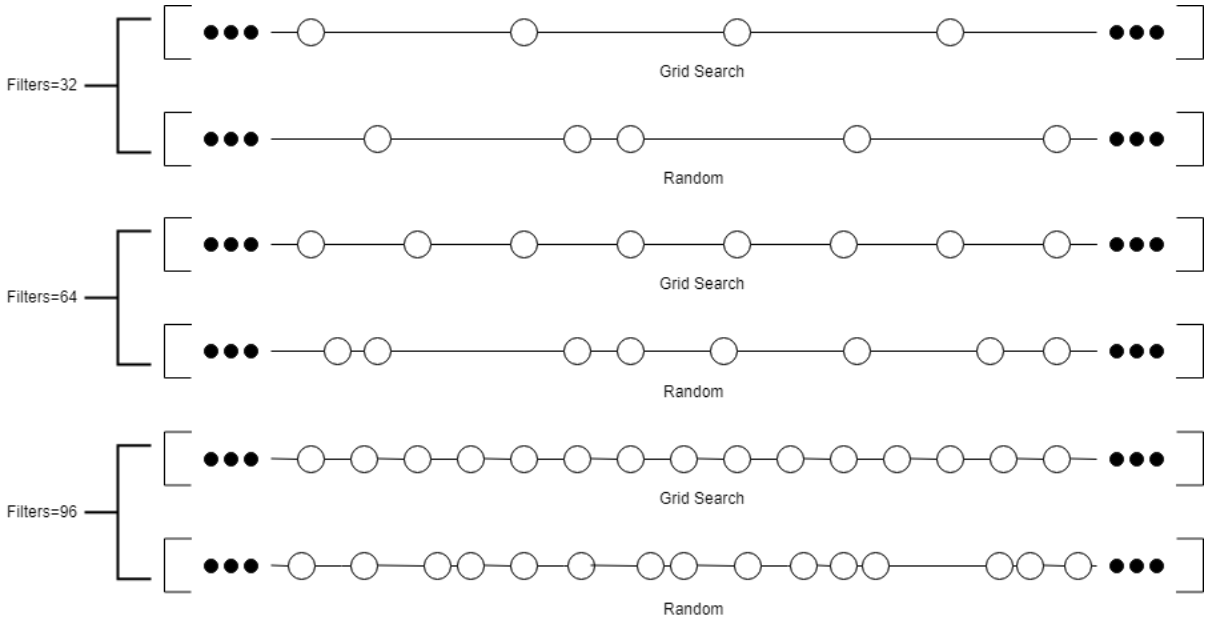


Figure 3.4: An example of how grid search versus random search is done for a given parameter. Grid search will offer a uniformity but can miss certain parameterizations, while randomness can fill those holes. Here, we can see if we add more density to the search, we can fill more holes, but will require more resources to do so.

3.3 Gabor CNN Structure (GCNN)

Below we will be describing two different architectures used for different datasets. The AlexisNet described is a smaller network with a different architecture, but instead of an Xavier normal distribution, we will use our Gabor initialization method instead.

3.3.1 MNIST, CIFAR-10, CIFAR-100 GCNN Architecture

Similarly to Ozbulak's architecture [16], we use the same architecture as well for a direct comparison on whether different initialization methods affect the final results.

The architecture is an AlexNet-like architecture and is as follows: the first layer is a convolutional layer that containes 96 filters of size 5×5. Each of these filters is then activated by a Rectified Linear Unit (ReLU). This is perhaps the most important layer in the architecture as we will change the initialization of this layer from the default Xavier uniform distribution to our Gabor filter initialization. The second layer is a convolutional layer that has 96 filters of size 1×1 and uses ReLU activation as well. The data then gets passed through a max-pooling filter of size 3×3 with a stride of 2. Once pooled, the data goes through a third convolutional layer of 192 filters of size 5×5. Once again, these channels are activated by ReLU and are fed into the fourth convolutional layer. The fourth layer has 192 filters of size 1×1 and are activated by ReLU. The data then gets fed through another max pooling layer of size 3×3 with a stride of 2. Once pooled, the data gets fed into the fifth convolutional layer where there are 192 filters of size 3×3 and activated by ReLU. Once activated, it gets fed into the sixth convolutional layer with 192 filters of size 1×1 and activated by ReLU. The seventh and final convolutional layer has 10 filters of size 1×1 and is pushed through an average pooling layer of size 6×6. The output from here is then fed into a fully connected layer using a softmax activation for classification. Dropout is used during training before the softmax activation (Figure 3.5). All convolutional layers aside from the first layer are initialized with the Xavier uniform distribution for the parameters. Biases are all initialized with zero. The optimization algorithm that was used throughout is the stochastic gradient descent (SGD). The chosen hyperparameters for learning rate, momentum and learning rate decay are 0.01, 0.9, and 0.0005 respectively. The loss function is based off of the Categorical cross entropy.

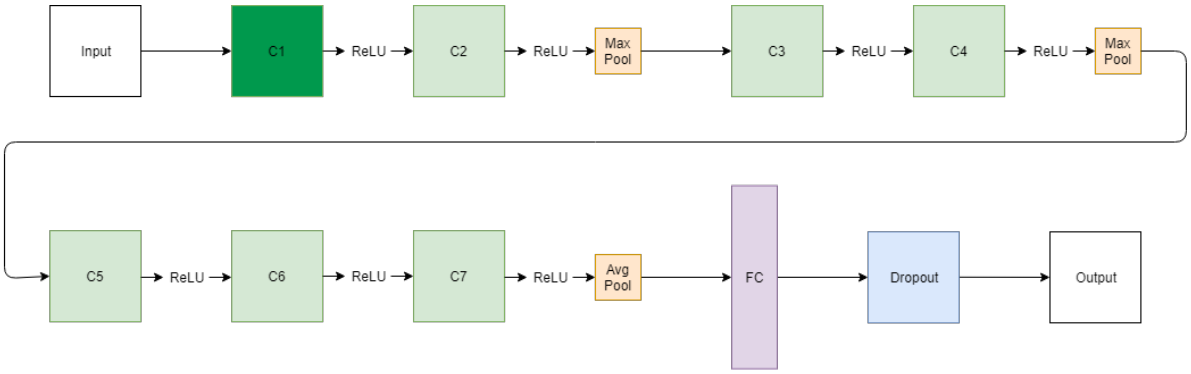


Figure 3.5: Architecture of the GCNN used during training for MNIST, CIFAR-10, CIFAR-100. C1 to C7 are the convolutional layers, and FC is the fully connected layer. C1 is the only layer in the network that has the ability to change how it is initialized with either a grid-search, random, or an Xavier uniform distribution.

As we will see in the next section, the first layer is important in the architecture because we are able to manipulate the initialization and the number of filters that are used during the training phase. The number of filters that can be used is important because of how many Gabor filters are created from it. Recall that we are going to use a grid search versus a random-search to see whether randomness improves the results from training and/or speeds up the training process. We also include dropout at the end of the network in some of our tests for most of the experiments.

3.3.2 AlexisNet GCNN Architecture

From AlexisNet [17], the author performed multiple tests to optimize his small CNN architecture. In the first experiment, Pascual was determining the optimal number of filters to be used within the convolutional layers that provided the best results. It was found that having a smaller number of filters was more optimal as there were less parameters to train and maintaining a higher accuracy over the other tests with larger number of filters.

Using his methodical approach to find his optimal setup, we will be using his latest architecture that consists of 3 convolutional layers, followed by a fully connected layer. After each convolutional layer, the data is max-pooled before getting fed into the next convolutional layer.

Between each convolutional layer it is activated by ReLU. The pooling layers are all of size 2×2 and each of the filter kernel sizes within the convolutions are of size 3×3. Before using softmax towards the end, there is a dropout layer of 50%. Refer to the following Figure 3.6 for a visual representation of the architecture.

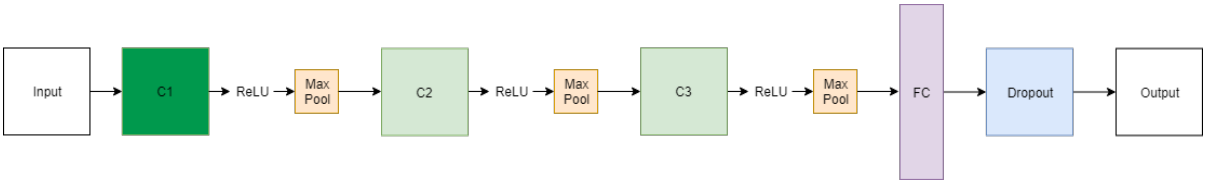


Figure 3.6: Architecture of AlexisNet used during training for his custom dataset described in Section 3.1.4. C1 to C3 are the convolutional layers, and FC is the fully connected layer. C1 is the only layer in the network that has the ability to change how it is initialized with either a grid-search, random, or an Xavier uniform distribution. In the original paper [17], the initialization method he had used was the Xavier uniform distribution

By using this test, we are able to see how well the Gabor filter performs on a small, custom dataset, and not just the standard datasets that are readily available. It is also worth noting that the architecture is not a complete replica of our given architecture, but a much simpler one. The tests will be done solely based on the first layer with different initialization strategies used. We will compare the results from using various filter sizes, different Gabor initializations and dropout, and compare these directly with the author's results that uses the Xavier distribution among a fixed filter size, $F = 32$, in the first layer.

Experiment	**Model**	**Training**		**Standard Deviation**	
		Accuracy	**Loss**	**Accuracy**	**Loss**
A	1 layer, 32 filters	0.9363	0.1851	0.0518	0.0184
	1 layer, 64 filters	0.9080	0.2499	0.0663	0.1963
	1 layer, 96 filters	0.8933	0.2827	0.0207	0.2860
B	2 layer, 32 filters	0.9700	0.0763	0.0262	0.0637
	3 layer, 32 filters	0.9960	0.0160	0.0055	0.0195

Table 3.2: We see the results from using Pascual's network described in Section 3.3.2 to obtain his results when training on the rock dataset (Section 3.1.4 [17]). The results obtained are the average of 10 trials using the same hyperparameters in all trials.

Chapter 4

Testing and Results

In this chapter, we design and describe how each experiment will be done, and what results they should provide. To clear up potential confusions, we are going to describe what each experiment, test and trial are. In each experiment, there are t number of tests that have n number of trials and e number of epochs. In the following chapter, the results will be shown for each section (A, B, C, D, E, F). Each section will describe the experiment, show a summary of the results, include example plots from tests within the experiment and will conclude with a discussion. At the end, there will be a summary for all of the experiments and our findings. All results are rounded to 4 decimal places, and for values that are less than 1/10000, we use scientific notation (for variance and standard deviation mostly). We use 4 decimal places to show more precision because in some cases, some of the results taken are similar (i.e., MNIST results are very close to each other). Please note that the term "uniform" is used as a grid-search and vice-versa throughout this section.

In the first 2 experiments, we will be exploring each parameter used in the Gabor filter. We will conclude whether parameters have a larger or lower impact on the CNN. We will also explore the impact of dropout, and changing the resolution of the parameter space for each parameter in the Gabor filter.

The next 4 experiments, we will explore using the grid search or the random search initial-

ization methods for the Gabor filter bank that is used in the first layer. Here we test using 4 different datasets with different complexity (colour versus grayscale, small set of classes versus large set of classes, etc.). The goal here in these experiments is to see how the initialization affects each dataset and show which method of initialization is better per dataset. We do these different initializations to see whether if random search is faster than grid search for the Gabor filter bank used in the CNN.

Following each experiment from C and onwards, we summarize our findings. In each case, we use a coloured header to classify findings. We and show confirmations with yellow highlight and new results with purple highlight.

Summary of Experiments
A: Importance of Gabor Parameters Training on MNIST, CIFAR-10
B: Search of a Single Parameter in the Gabor Filter Training on MNIST, CIFAR-10
C: Grid Search Vs. Random Search with Shuffled Datasets Training on MNIST, CIFAR-10, CIFAR-100
D: Grid Search Vs. Random Search with Non-Shuffled Datasets Training on MNIST, CIFAR-10, and CIFAR-100
E: Grid Search Vs. Random Search with a Non-Trainable First Layer and Non-Shuffled Datasets Training on MNIST, CIFAR-10, and CIFAR-100
F: Grid Search Vs. Random Search with a Simple Rock Database

Table 4.1: A summary of all of the experiments from A to F for this thesis.

4.1 A: Importance of Gabor Parameters Training on MNIST, CIFAR-10

4.1.1 Experiment Setup

In this first experiment, we describe the architecture of the Gabor filter bank used to see the impact of each parameter. The Gabor filter bank created will be static across all tests where each parameter is tested, and changed once per test. For example, in one test, the parameter list will look like the following: $[\sigma = x, \lambda = 1, \theta = 0, \gamma = 0, \psi = 0]$, where x is a static number given

from the closed interval of $I = [a, b]$, which describes the minimum value at a and maximum value at b of that parameter. This filter generated will be the same across all filters in the Gabor filter bank. The architecture described in Section 3.3 will be used. Each test trains for 10 epochs for a total of 5 tests. The filter size never changes for this experiment at 96 filters in layer 1. For CIFAR-10, we introduce more channels (colours) for computation.

The Gabor filter bank will be static for each test and during training, the generated Gabor filters will be able to learn through training. The datasets used will be shuffled per trial and the final results will be taken as an average of the trials done. The values chosen are to resemble the values given from [16] in order to compare with in the upcoming tests. Additionally, we will be testing how dropout affects the final results. We will test 3 different dropouts of 0%, 25% and 50% ($D = [0, 0.25, 0.50]$). The parameters selected are based off of [16].

Parameter	**Interval**	**Chosen Values**
Sigma, σ	[2, 22]	[2, 6, 10, 14, 18, 22]
Lambda, λ	[0, 100]	[0, 20, 40, 60, 80, 100]
Theta, θ	[0, 360]	[0, 60, 120, 180, 240, 300]
Gamma, γ	[0, 300]	[0, 60, 120, 180, 240, 300]
Psi, ψ	[0, 360]	[0, 60, 120, 180, 240, 300]

Table 4.2: A list of the parameters that can be tuned within the Gabor Filter. The ranges given are inclusive. For the sake of using an even interval, the ranges are slightly tuned. As a reminder, Section 1.2.3 describes each parameter within the Gabor filter. Note that for Theta and Psi, calculating at 0° and 360° is the same result.

For MNIST, CIFAR-10, and CIFAR-100, the each test will have 10 epochs for 5 trials. As a reminder, each trial will have the dataset shuffled so that no trial is similar. The number of Gabor kernels created will be fixed at 96 filters. Each test is done once per parameter change. For example, in one test run, $\sigma = 2$ while other variables in the Gabor filter are fixed. The next test, σ will be a fixed increment and the other variables are fixed. Each parameter tests 6 different values within the range given and the increments between each test are fixed (for

example, σ's given interval $I = [2, 22]$, the values chosen will be $I = [2, 6, 10, 14, 18, 22]$ inclusive. The tables following this section shows the best results in bold.

Test Case	Dropout	Ksize	Sigma	Lambda	Theta	Gamma	Psi
A: Sigma	[0,0.25,0.50]	5×5	[2,22]	50	0	150	0
B: Lambda	[0,0.25,0.50]	5×5	10	[0,100]	0	150	0
C: Theta	[0,0.25,0.50]	5×5	10	50	[0,300]	150	0
D: Gamma	[0,0.25,0.50]	5×5	10	50	0	[0,300]	0
E: Psi	[0,0.25,0.50]	5×5	10	50	0	150	[0,300]

Table 4.3: Each tests from A through E listed, one parameter will change while the other parameters will remain static. Dropout has 3 test cases, $D = [0, 0.25, 0.50]$ while the other parameters (σ, λ, θ, γ, ψ) changes with 6 evenly divided intervals between the maximum and the minimum inclusive. Therefore in each test case, we have a total of 18 tests. This experiment has a grand total of 90 tests.

4.1.2 Results

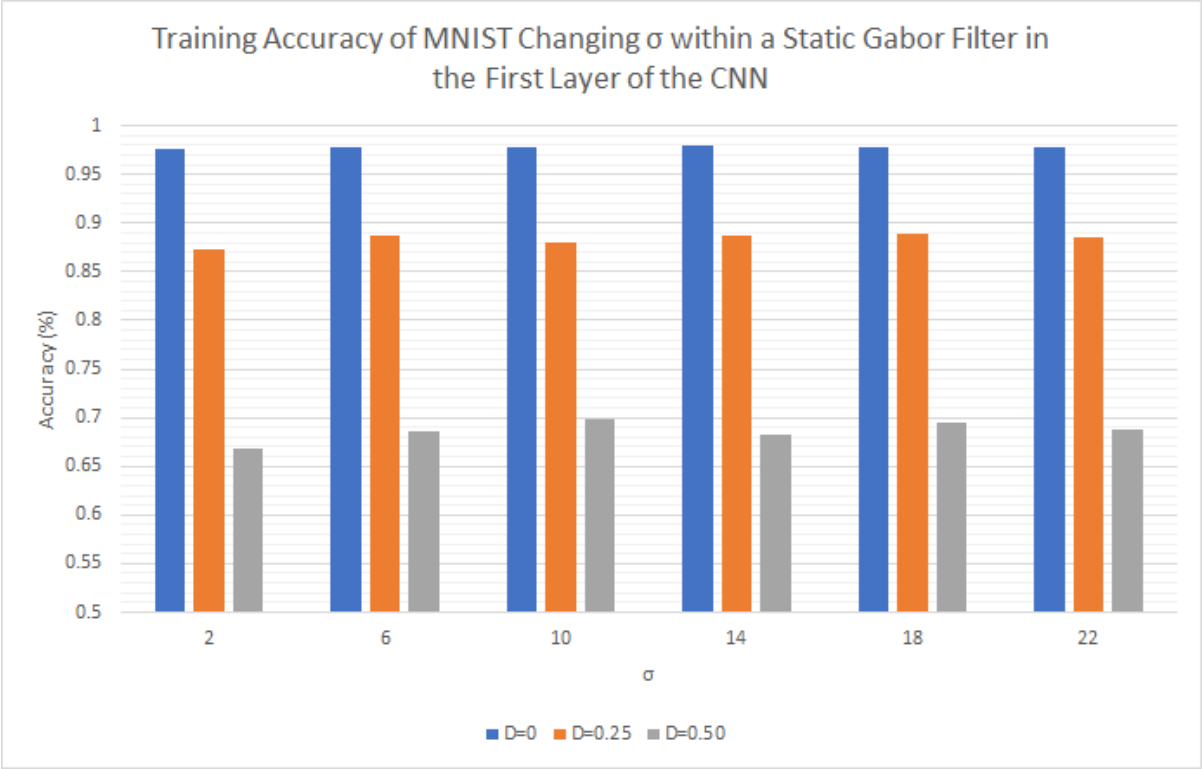
Sigma, σ

Gabor Parameters	Dataset	Dropout	Sigma	Training		Validation	
				Accuracy	Loss	Accuracy	Loss
$ksize = 5 \times 5$ $lambda = 50$ $theta = 0$ $gamma = 150$ $psi = 0$	MNIST	0	2	0.9756	0.0797	0.9802	0.0649
			6	0.9772	0.0746	0.9790	0.0682
			10	0.9783	0.0701	0.9824	0.0576
			14	**0.9789**	**0.0695**	**0.9808**	**0.0598**
			18	0.9784	0.0713	0.9785	0.0662
			22	0.9780	0.0727	0.9806	0.0631
		0.25	2	0.8731	0.3679	0.9715	0.1133
			6	0.8867	0.3295	0.9741	0.0975
			10	0.8806	0.3295	0.9741	0.1050
			14	0.8875	0.3362	0.9737	0.1021
			18	**0.8897**	**0.3129**	**0.9741**	**0.0900**
			22	0.8860	0.3242	0.9732	0.0969
		0.50	2	0.6686	0.8465	0.9624	0.2138
			6	0.6867	0.8187	0.9599	0.2194
			10	**0.6977**	**0.7933**	**0.9626**	**0.1988**
			14	0.6829	0.8121	0.9606	0.2073
			18	0.6951	0.7923	0.9663	0.1815
			22	0.6880	0.7934	0.9604	0.2041

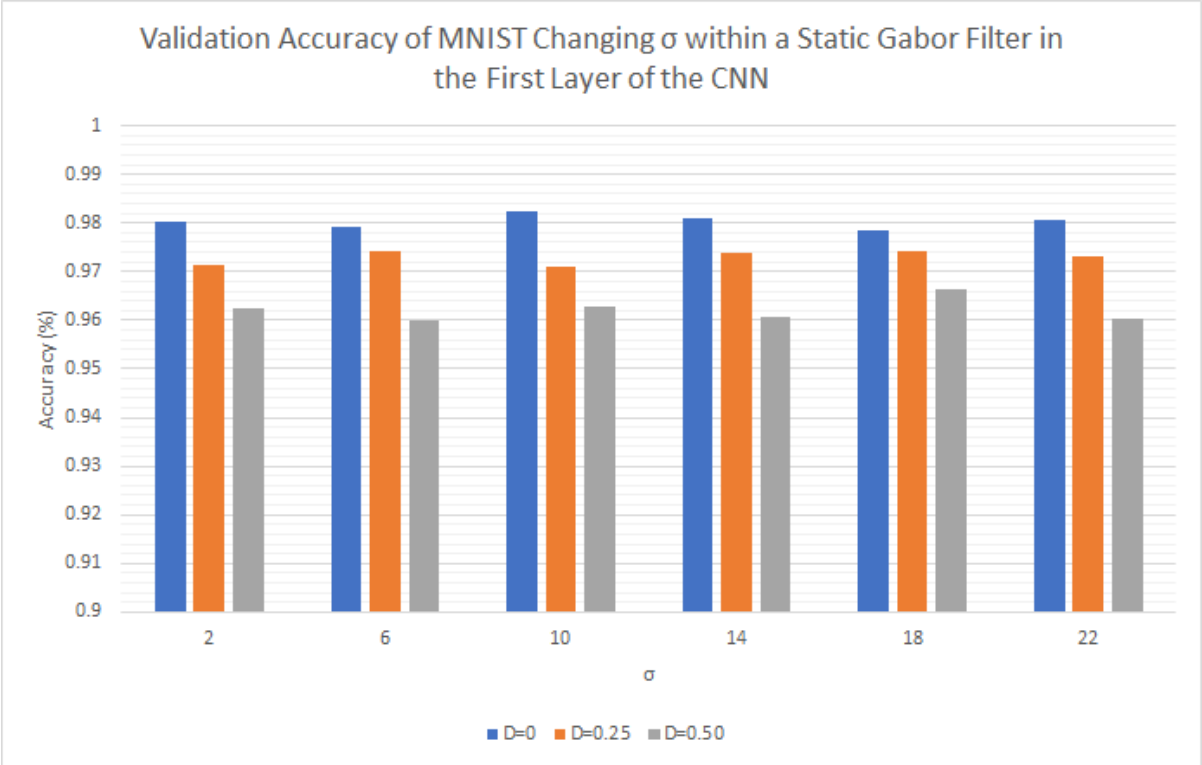
Table 4.4: Running a simple test showing a static first layer using a Gabor filter bank, testing on the MNIST dataset. The layers are able to learn during training. In this test, σ is the only variable that is changing in a range given as $[2, 6, 10, 14, 18, 22]$ inclusive. σ changes 6 times for each different dropout given.

Gabor Parameters	Dataset	Dropout	Sigma	Training		Validation	
				Accuracy	Loss	Accuracy	Loss
$ksize = 5 \times 5$ $lambda = 50$ $theta = 0$ $gamma = 150$ $psi = 0$	CIFAR-10	0	2	0.4315	1.5970	0.4320	1.5918
			6	0.3618	1.7474	0.3544	1.7667
			10	0.4061	1.6529	0.4172	1.6335
			14	0.4277	1.6035	0.4295	1.5932
			18	**0.4335**	**1.5939**	**0.4262**	**1.6137**
			22	0.4230	1.6151	0.4256	1.6120
		0.25	2	0.0975	1.9121	0.3546	1.8238
			6	0.2676	1.9656	0.3346	1.8238
			10	0.2669	1.9761	0.3401	1.8746
			14	0.2770	1.9587	0.3308	1.8690
			18	0.2561	1.9932	0.3134	1.9030
			22	**0.2831**	**1.9342**	**0.3498**	**1.8346**
		0.50	2	0.1871	2.1287	0.2696	2.0327
			6	**0.2113**	**2.0948**	**0.3019**	**1.9916**
			10	0.1679	2.1740	0.2189	2.0988
			14	0.1709	2.1768	0.2335	2.1105
			18	0.1780	2.1582	0.2448	2.0777
			22	0.1582	2.1928	0.2187	2.1346

Table 4.5: Running a simple test showing a static first layer using a Gabor filter bank, testing on the CIFAR-10 dataset. The layers are able to learn during training. In this test, σ is the only variable that is changing in a range given as $[2, 6, 10, 14, 18, 22]$ inclusive. σ changes 6 times for each different dropout given.

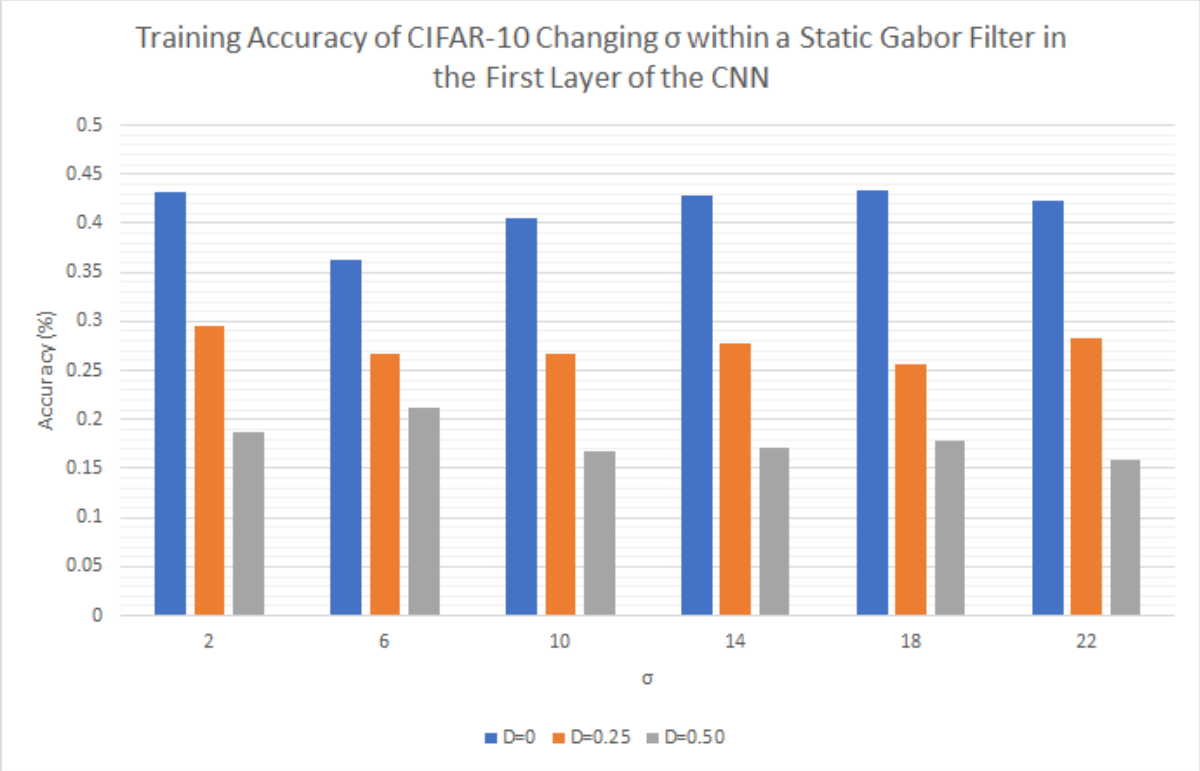


(a)

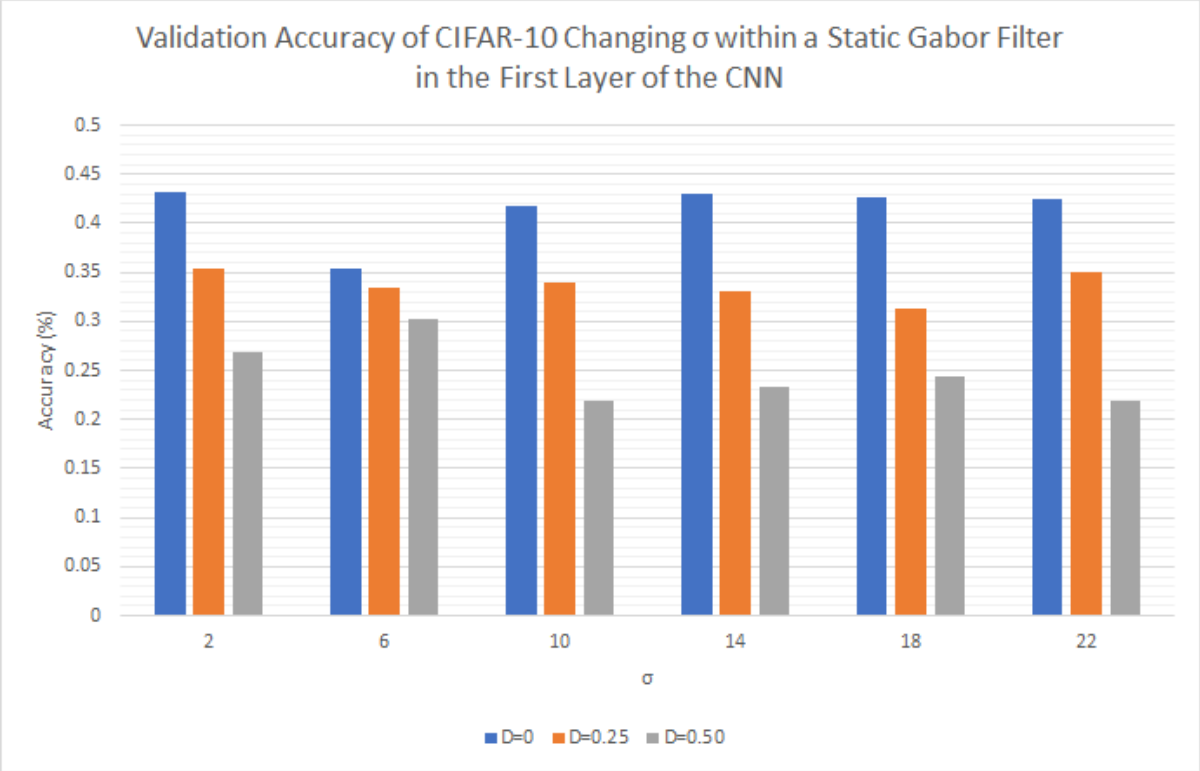


(b)

Figure 4.1: Here we see a different representation of Table 4.4 showing the training and validation accuracy of the GCNN using MNIST for σ. Each colored bar in the legend shows the dropout rate used during training.



(a)



(b)

Figure 4.2: Here we see a different representation of Table 4.5 showing the training and validation accuracy of the GCNN using CIFAR-10 for σ. Each colored bar in the legend shows the dropout rate used during training.

Discussion

Training and validation accuracy drops when the CNN introduces dropout. We can see that the error increases, as well, but with dropout, it should workout in the long term in terms of generalization when training to avoid overfitting the dataset; however, this is not the case. Overfitting is the result of training a dataset to correctly identify the dataset's classes, but when introduced to new external data, the CNN will have a harder time identifying. There appears to be a very small pattern in terms of improvement where the results become greater as σ increases and then decreases almost like a Gaussian curve. We can see this in Table 4.4 where the values increase up until the bolded results and then decreases. This pattern appears in the MNIST dataset more than the CIFAR-10 dataset. This could be the result of how random the datasets are, especially when CIFAR-10 introduces colour in their dataset.

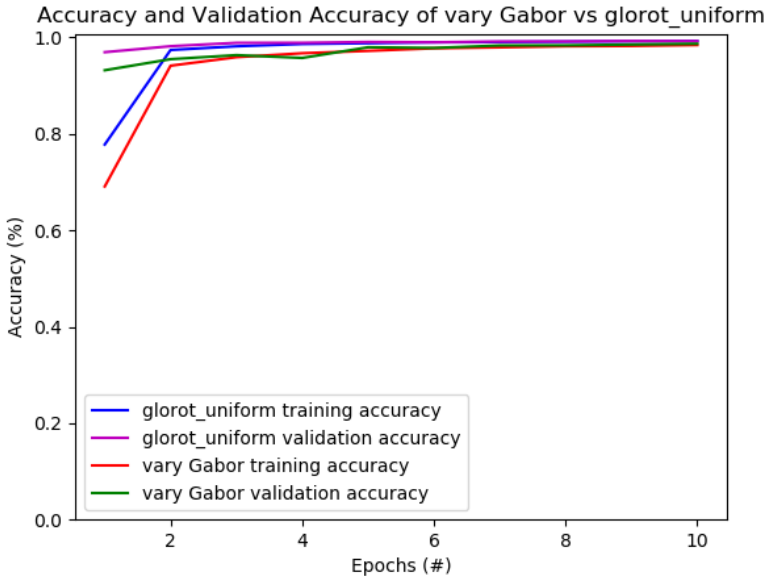


Figure 4.3: A test showing the training results per epoch for when $\sigma = 18$ and dropout, $D = 0$, when using the Gabor filter as initialization in the first layer. In comparison we can see how a Xavier initialization competes. In this static experiment, MNIST is the dataset being used to train on. The legend item that says 'vary Gabor' signifies the results from when $\sigma = 18$.

We can see that for a simpler dataset, i.e., MNIST, σ has a low impact on the final results, but when we introduce colour (CIFAR-10 dataset), we can see that σ has much more

of an affect. There is no specific pattern that can be seen on the CIFAR-10 dataset. As mentioned previously, this is likely due to how random the datasets can appear to the network, or alternatively, adding colour adds more complexity and makes training harder, more sporadic.

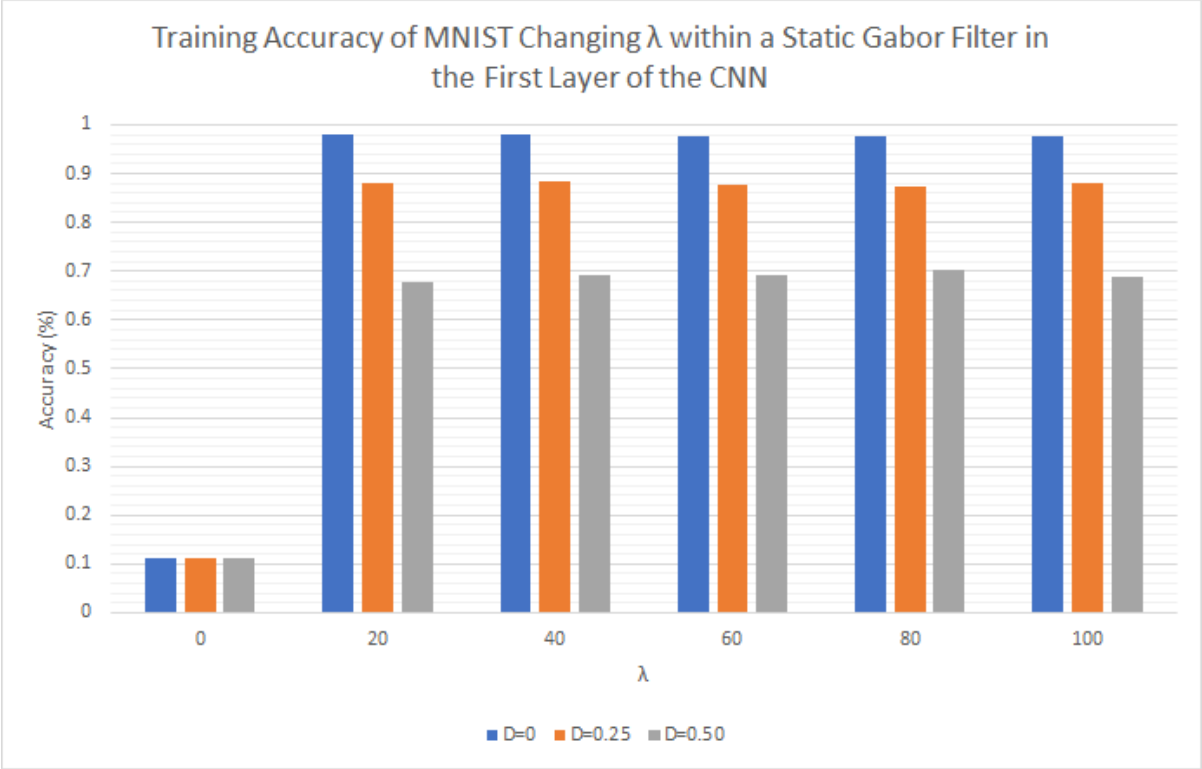
Lambda, λ

Gabor Parameters	Dataset	Dropout	Lambda	Training		Validation	
				Accuracy	Loss	Accuracy	Loss
$ksize = 5 \times 5$ $sigma = 10$ $theta = 0$ $gamma = 150$ $psi = 0$	MNIST	0	0	0.1124	2.3012	0.1135	2.3010
			20	0.9784	0.0706	0.9812	0.0608
			40	**0.9787**	**0.0697**	**0.9811**	**0.0603**
			60	0.9778	0.0719	0.9762	0.0749
			80	0.9783	0.0718	0.9821	0.0582
			100	0.9777	0.0724	0.9797	0.0644
		0.25	0	0.1124	2.3012	0.1135	2.3010
			20	0.8798	0.3488	0.9728	0.1012
			40	**0.8825**	**0.3370**	**0.9747**	**0.1000**
			60	0.8753	0.3662	0.9706	0.1165
			80	0.8750	0.3582	0.9744	0.0996
			100	0.8789	0.3606	0.9706	0.1219
		0.50	0	0.1124	2.3012	0.1135	2.3010
			20	0.6779	0.8412	0.9624	0.2027
			40	0.6907	0.8065	0.9642	0.1946
			60	0.6918	0.8102	0.9666	0.1967
			80	**0.7029**	**0.7945**	**0.9629**	**0.2031**
			100	0.6894	0.8064	0.9662	0.2002

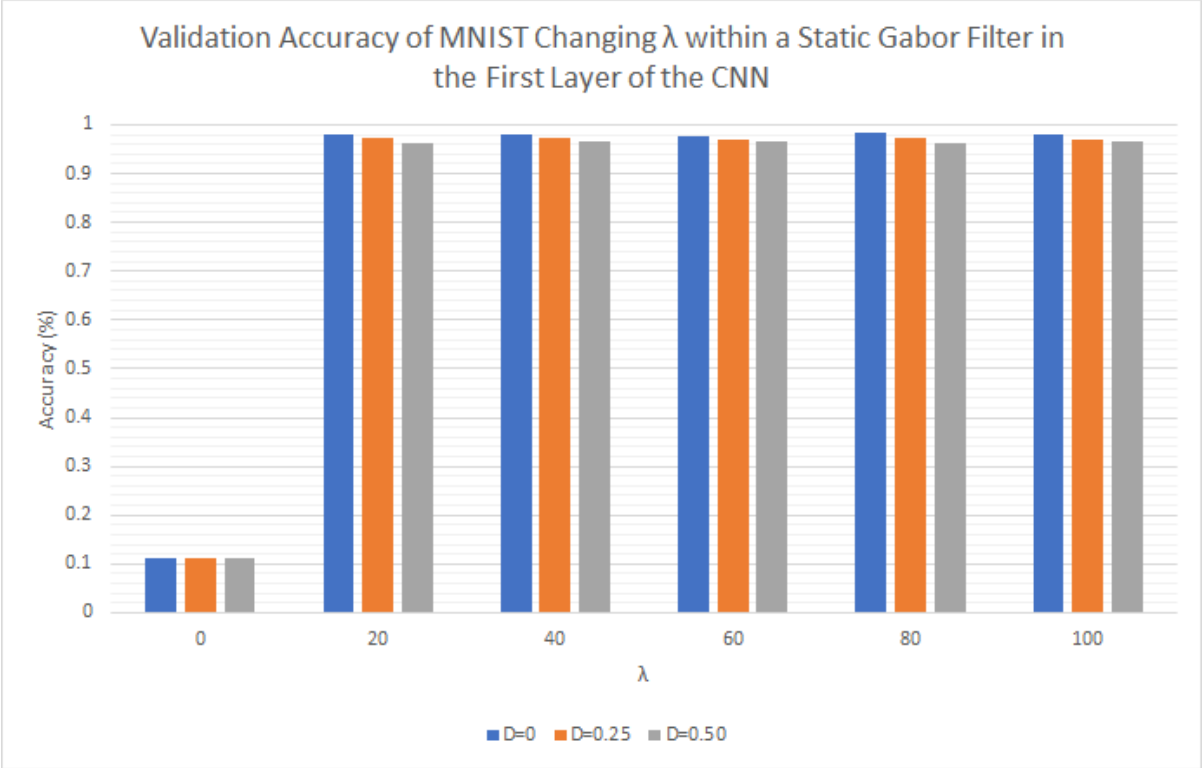
Table 4.6: Running a simple test showing a static first layer using a Gabor filter bank, testing on the MNIST dataset. The layers are able to learn during training. In this test, λ is the only variable that is changing in a range given as $[0, 20, 40, 60, 80, 100]$ inclusive. λ changes 6 times for each different dropout given.

Gabor Parameters	Dataset	Dropout	Lambda	Training		Validation	
				Accuracy	Loss	Accuracy	Loss
$ksize = 5 \times 5$ $sigma = 10$ $theta = 0$ $gamma = 150$ $psi = 0$	CIFAR-10	0	0	0.0981	2.3026	0.1000	2.3026
			20	0.4055	1.6482	0.4147	1.6396
			40	0.4078	1.6541	0.4127	1.6469
			60	**0.4082**	**1.6479**	**0.4041**	**1.6549**
			80	0.3989	1.6733	0.3913	1.6904
			100	0.3993	1.6659	0.4018	1.6611
		0.25	0	0.0975	2.3026	0.1000	2.3026
			20	**0.2981**	**1.9014**	**0.3593**	**1.7958**
			40	0.2404	2.0244	0.2921	1.9402
			60	0.2547	1.9945	0.3023	1.9145
			80	0.2901	1.9168	0.3514	1.8089
			100	0.2367	2.0250	0.2883	1.9403
		0.5	0	0.0975	2.3026	0.1000	2.3026
			20	0.1603	2.2097	0.2167	2.1619
			40	**0.2093**	**2.1000**	**0.3075**	**1.9852**
			60	0.1449	2.2081	0.2040	2.1510
			80	0.1716	2.1724	0.2328	2.0989
			100	0.1736	2.1763	0.2432	2.1038

Table 4.7: Running a simple test showing a static first layer using a Gabor filter bank, testing on the CIFAR-10 dataset. The layers are able to learn during training. In this test, λ is the only variable that is changing in a range given as $[0, 20, 40, 60, 80, 100]$ inclusive. λ changes 6 times for each different dropout given.

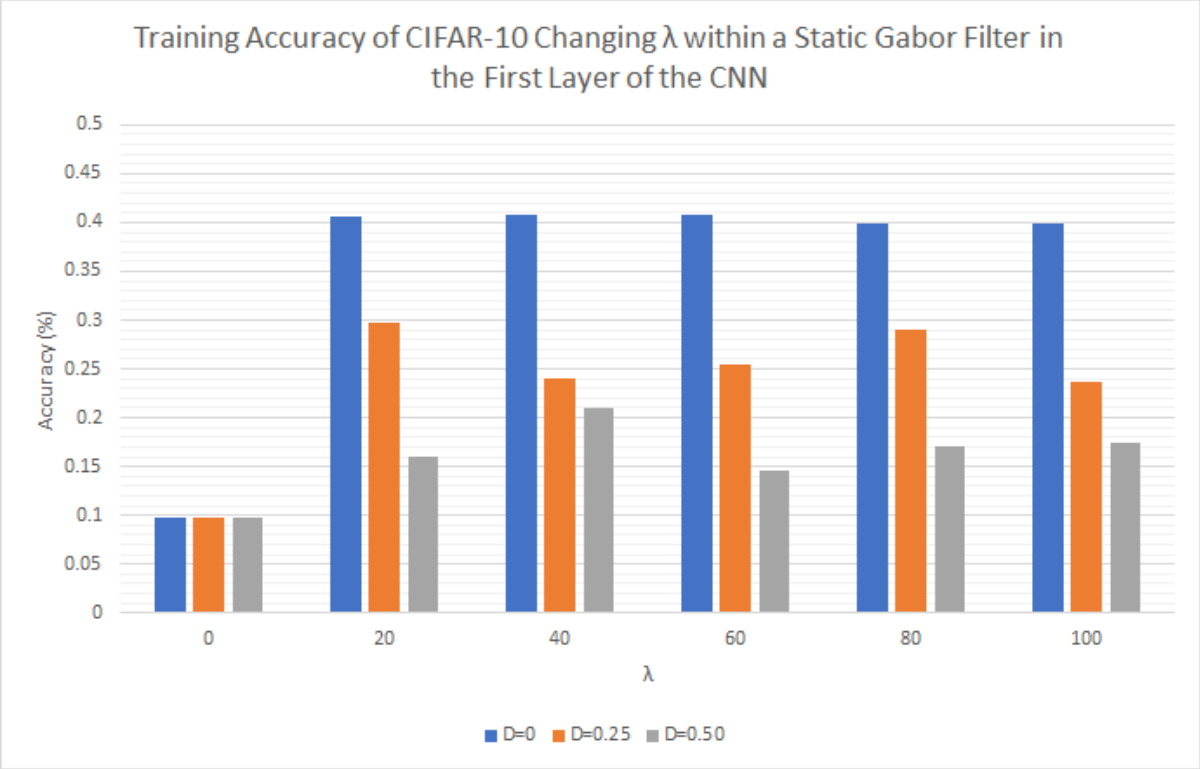


(a)

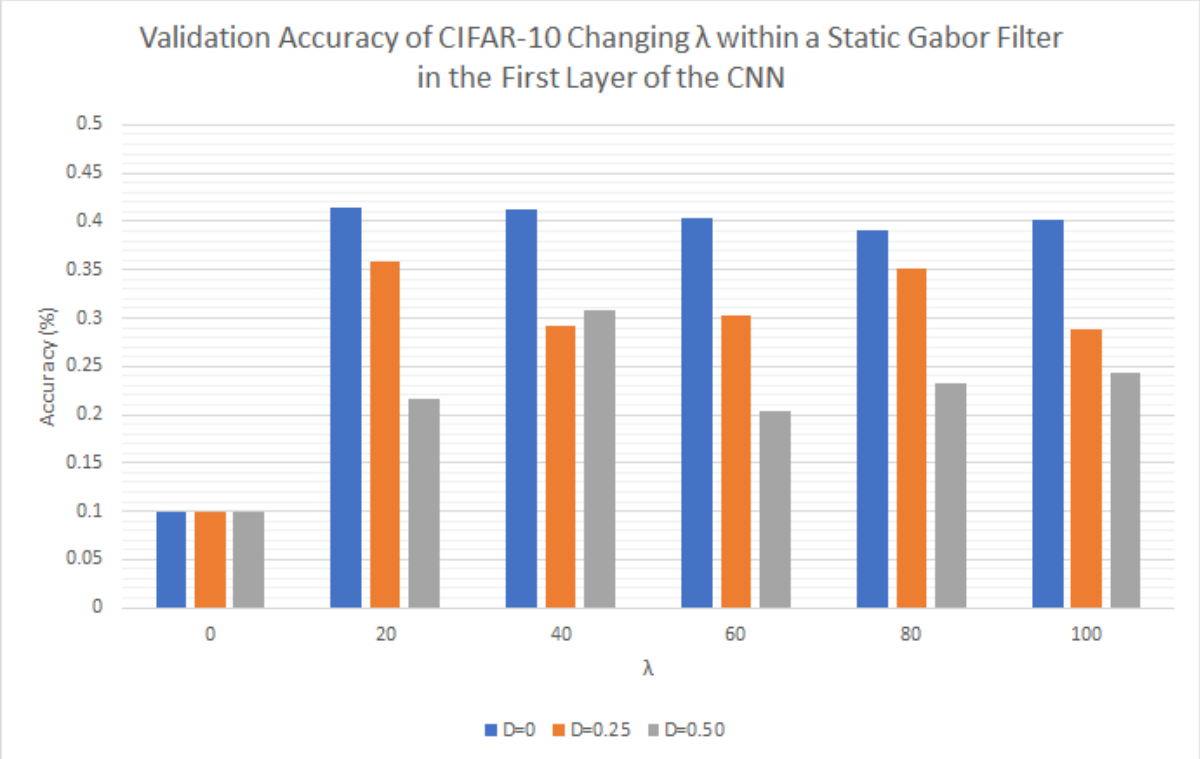


(b)

Figure 4.4: Here we see a different representation of Table 4.6 showing the training and validation accuracy of the GCNN using MNIST for λ. Each colored bar in the legend shows the dropout rate used during training.



(a)



(b)

Figure 4.5: Here we see a different representation of Table 4.7 showing the training and validation accuracy of the GCNN using CIFAR-10 for λ. Each colored bar in the legend shows the dropout rate used during training.

Discussion

The highest scoring results here are bolded in Tables 4.6 and 4.7. We can see that the results are very close to each other for the MNIST dataset and sporadic once again like the σ results for the CIFAR-10 dataset. For both of the datasets, λ tends to be the best between $[40, 80]$. When dropout is introduced like σ, we can see that the results drop in performance in both datasets, but drops more significantly for the CIFAR-10 dataset.

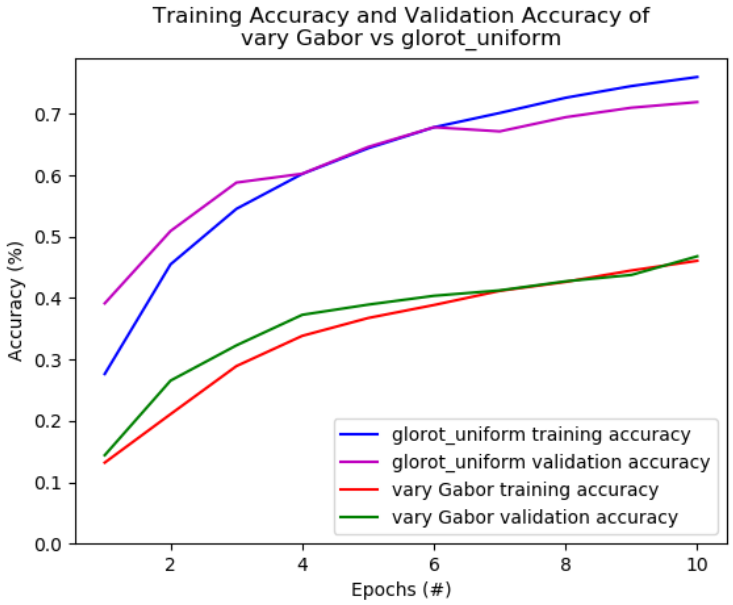


Figure 4.6: A test showing the training results per epoch for when $\lambda = 60$ and dropout, $D = 0$, when using the Gabor filter as initialization in the first layer. In comparison we can see how a Xavier initialization competes. In this static experiment, CIFAR-10 is the dataset being used to train on. The legend item that says 'vary Gabor' signifies the results from when $\lambda = 60$.

There are no patterns that can be extracted from these results. One thing we can learn from using λ is that when $\lambda = 0$, we can see from the results that a 0 value should never be used. The network has trouble achieving to learn when there is no value given to *lambda* for both MNIST and CIFAR-10. This is likely due to the 2-D formula given for the Gabor filter in Section 1.2.3, where λ must be a value greater than 0. Otherwise, the resulting formula should not function at all. Due to the dataset being shuffled in CIFAR-10, the results for λ has a harder time to be consistent but the results are all reasonably within a range that shows no significant impact on

the results as a whole.

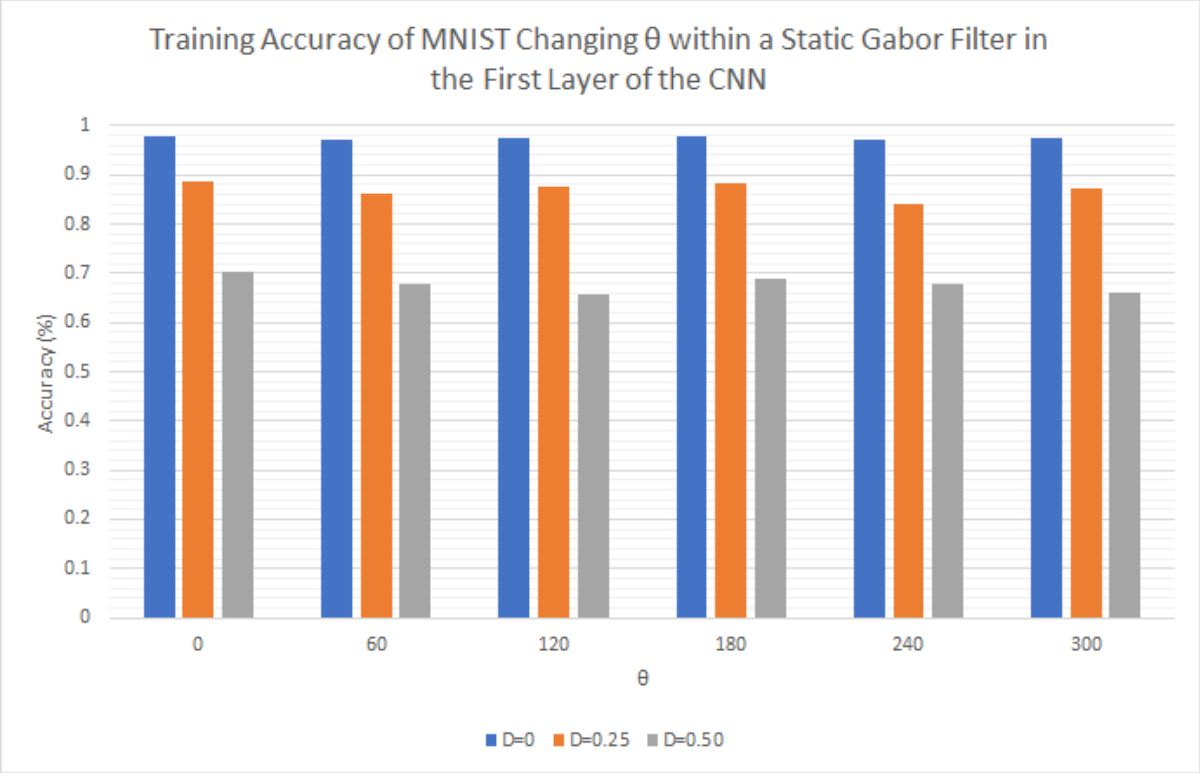
Theta, θ

Gabor Parameters	Dataset	Dropout	Theta	Training		Validation	
				Accuracy	Loss	Accuracy	Loss
$ksize = 5 \times 5$ $sigma = 10$ $lambda = 50$ $gamma = 150$ $psi = 0$	MNIST	0	**0**	**0.9787**	**0.0702**	**0.9803**	**0.0634**
			60	0.9720	0.0940	0.9767	0.0785
			120	0.9749	0.0835	0.9771	0.0737
			180	0.9783	0.0711	0.9794	0.0651
			240	0.9732	0.0886	0.9777	0.0715
			300	0.9736	0.0885	0.9756	0.0777
		0.25	**0**	**0.8883**	**0.3299**	**0.9753**	**0.0954**
			60	0.8627	0.4016	0.9682	0.1283
			120	0.8773	0.3581	0.9704	0.1100
			180	0.8839	0.3341	0.9743	0.0947
			240	0.8410	0.4330	0.9671	0.1358
			300	0.8715	0.3753	0.9714	0.1149
		0.5	**0**	**0.7017**	**0.8104**	**0.9657**	**0.2046**
			60	0.6770	0.8434	0.9557	0.2539
			120	0.6579	0.8815	0.9610	0.2402
			180	0.6894	0.8132	0.9615	0.2190
			240	0.6769	0.8273	0.9651	0.1973
			300	0.6593	0.8813	0.9602	0.2362

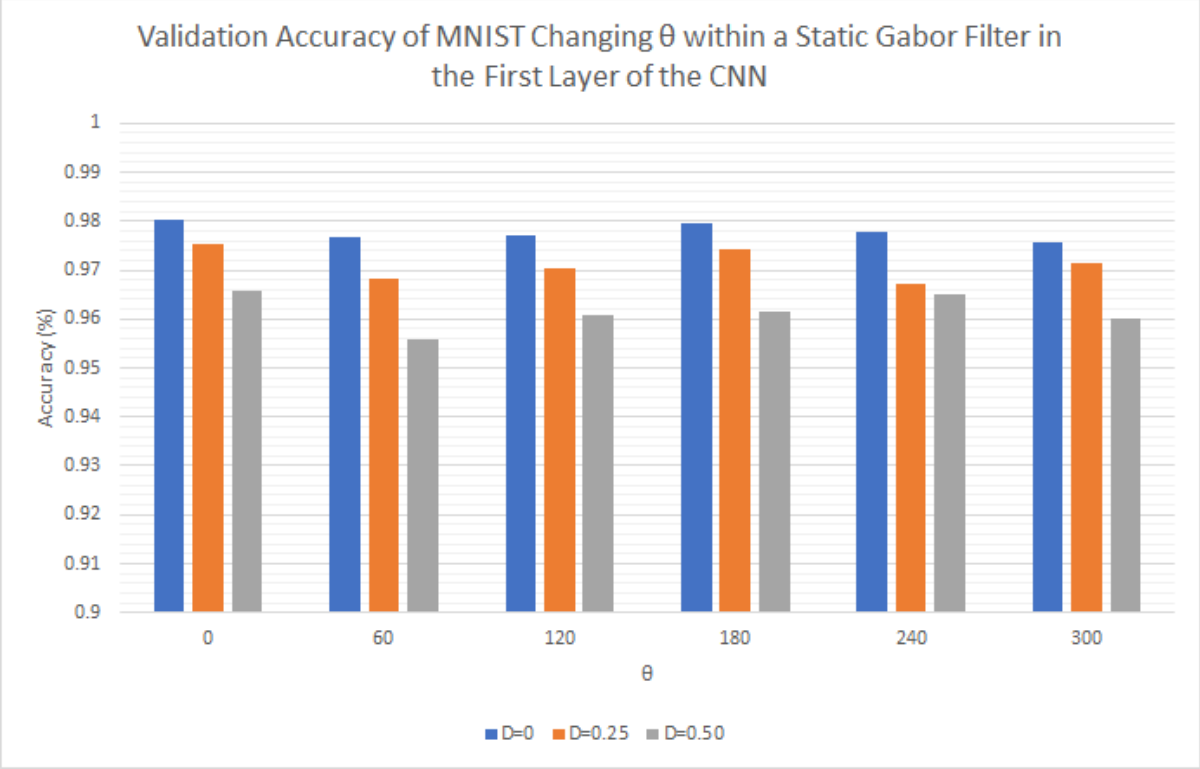
Table 4.8: Running a simple test showing a static first layer using a Gabor filter bank, testing on the MNIST dataset. The layers are able to learn during training. In this test, θ is the only variable that is changing in a range given as $[0, 60, 120, 180, 240, 300]$ inclusive. θ changes 6 times for each different dropout given.

Gabor Parameters	Dataset	Dropout	Theta	Training		Validation	
				Accuracy	Loss	Accuracy	Loss
$ksize = 5 \times 5$ $sigma = 10$ $lambda = 50$ $gamma = 150$ $psi = 0$	CIFAR-10	0	0	0.4106	1.6504	0.4199	1.6226
			60	0.4579	1.4883	0.4697	1.4493
			120	0.4666	1.4663	0.4751	1.4520
			180	0.3564	1.7564	0.3594	1.7467
			240	**0.4775**	**1.4377**	**0.4877**	**1.4192**
			300	0.4631	1.4785	0.4704	1.4620
		0.25	0	0.2435	2.0116	0.2992	1.9358
			60	0.3141	1.8347	0.3910	1.7004
			120	0.3241	1.8053	0.3932	1.6748
			180	0.2374	2.0302	0.2780	1.9829
			240	**0.3330**	**1.7656**	**0.4052**	**1.6456**
			300	0.3245	1.7883	0.4016	1.6699
		0.5	0	0.1547	2.2069	0.1984	2.1633
			60	0.2317	2.0155	0.3254	1.8722
			120	0.2352	1.9988	0.3352	1.8672
			180	0.1626	2.1864	0.2171	2.1262
			240	**0.2400**	**1.9741**	**0.3418**	**1.8409**
			300	0.2271	2.0102	0.3196	1.8627

Table 4.9: Running a simple test showing a static first layer using a Gabor filter bank, testing on the CIFAR-10 dataset. The layers are able to learn during training. In this test, θ is the only variable that is changing in a range given as $[0, 60, 120, 180, 240, 300]$ inclusive. θ changes 6 times for each different dropout given.

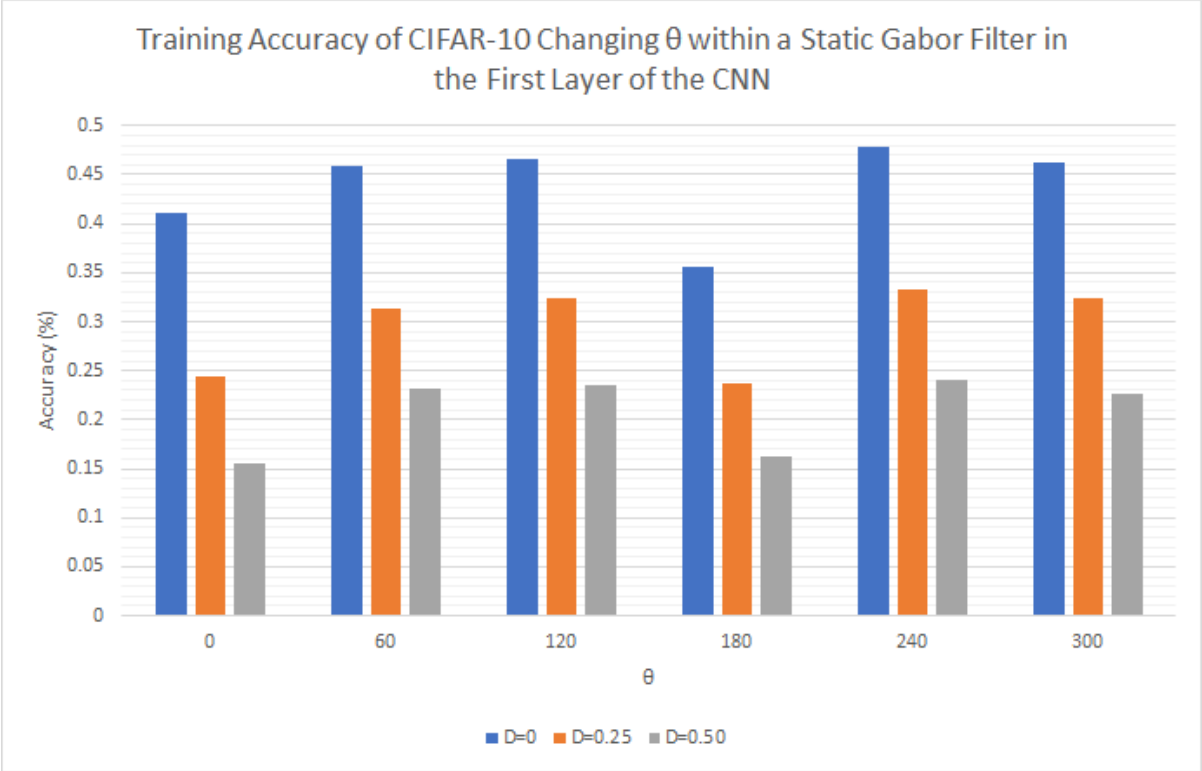


(a)

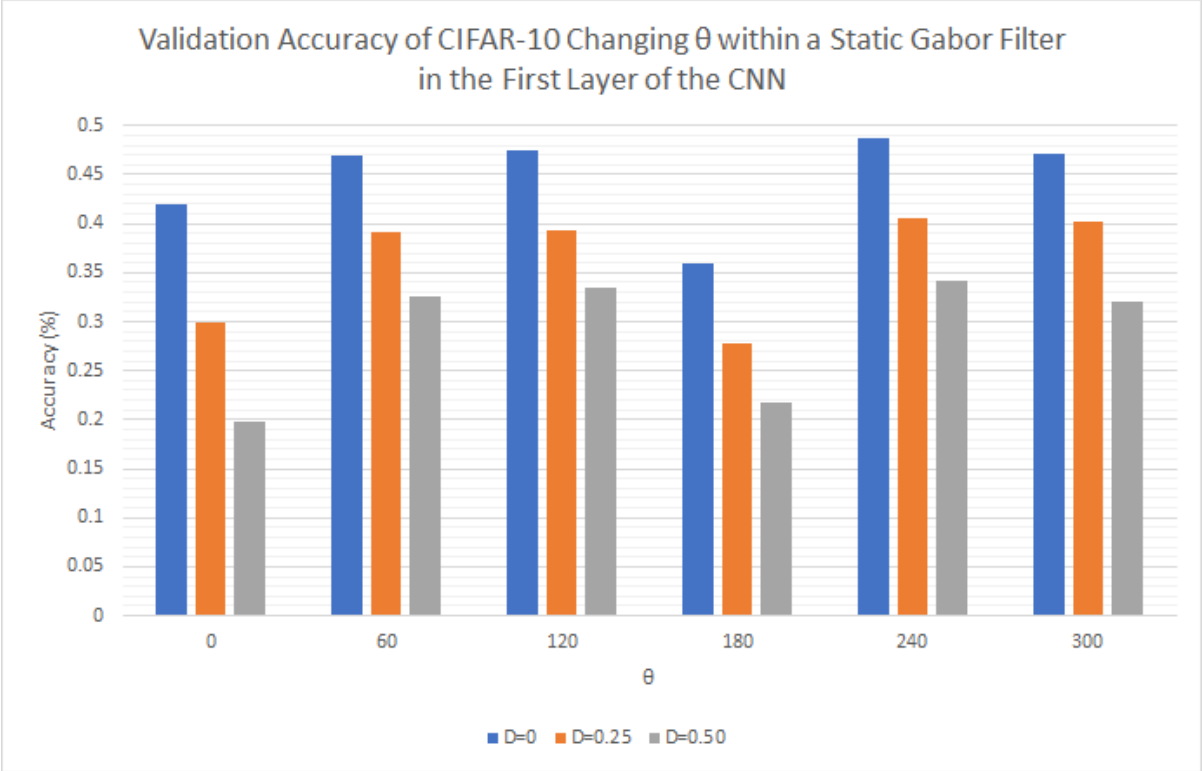


(b)

Figure 4.7: Here we see a different representation of Tables 4.8 showing the training and validation accuracy of the GCNN using MNIST for θ. Each colored bar in the legend shows the dropout rate used during training.



(a)



(b)

Figure 4.8: Here we see a different representation of Table 4.9 showing the training and validation accuracy of the GCNN using CIFAR-10 for θ. Each colored bar in the legend shows the dropout rate used during training.

Discussion

When we tune θ for the MNIST dataset, and when we include dropout, the results tend to drop. We can see that there is no pattern that we can recognize and the results are very close to one another. It appears that when $\theta = 0$, it shows the highest results in MNIST and when $\theta = 240$ for CIFAR-10, the results are the highest. Perhaps due to the complexity of multiple channels, rotating the filter (example in Figure 1.7) for one channel is much easier to obtain than rotating for 3 different channels (like in CIFAR-10 dataset). This could also mean that when using different angles of the Gabor filter, CIFAR-10 reacts more, especially because the objects within the images are not just numbers but cars, animals, buildings, etc.

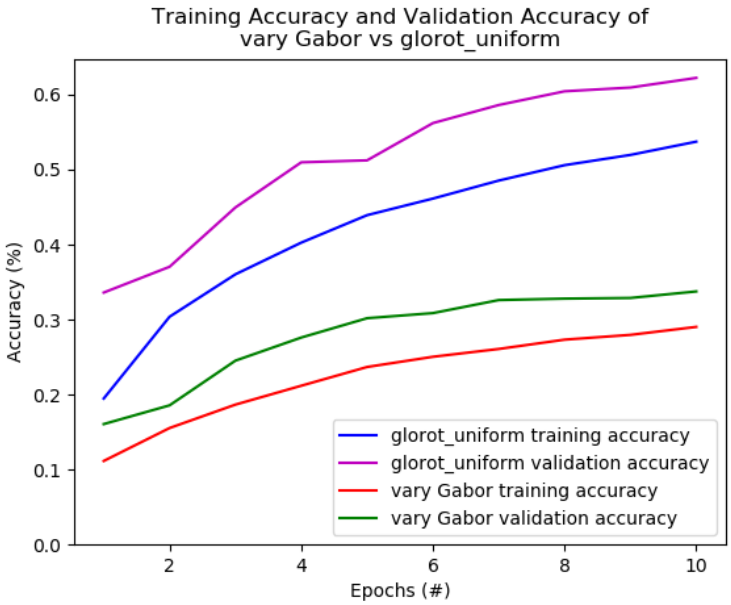
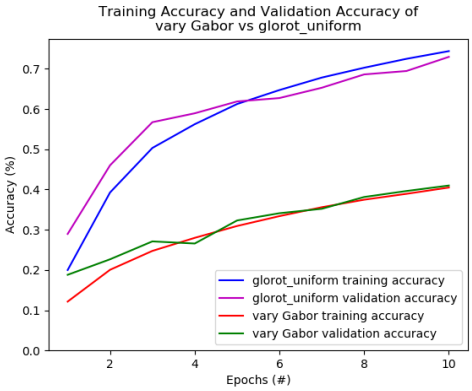


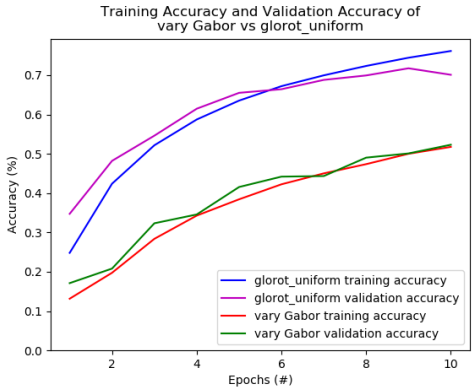
Figure 4.9: A test showing the training results per epoch for when $\theta = 0$ and dropout, $D = 0.25$, when using the Gabor filter as initialization in the first layer. In comparison we can see how a Xavier initialization competes. In this static experiment, CIFAR-10 is the dataset being used to train on. The legend item that says 'vary Gabor' signifies the results from when $\theta = 0$.

In both datasets, MNIST and CIFAR-10, the results are random as we increase θ. We can say the same as σ and λ where the parameter does not greatly affect the results so long as the values exist. Due to the nature of the Gabor filter, by rotating the Gabor filters, we can extract different features at each frequency which is believed to be why the Gabor filter is said to be

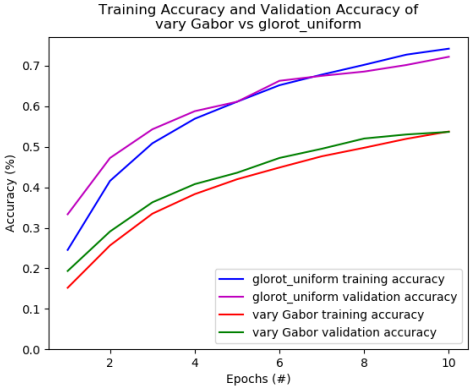
a good alternative from transfer learning [16]. The results drop the same from the previous 2 tests with σ and λ for both datasets.



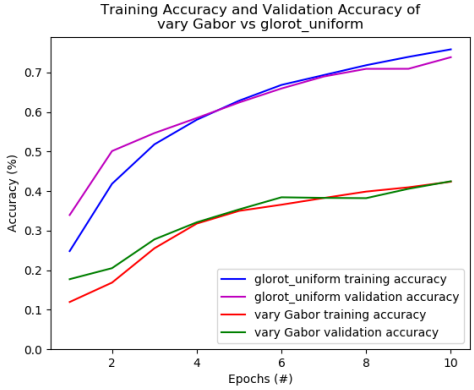
(a) $\theta = 0$



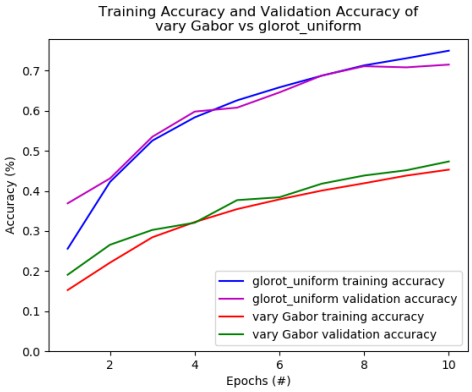
(b) $\theta = 60$



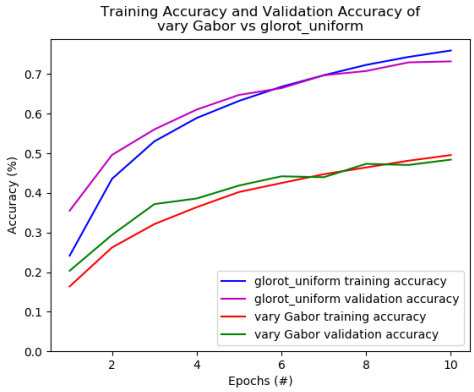
(c) $\theta = 120$



(d) $\theta = 180$



(e) $\theta = 240$



(f) $\theta = 300$

Figure 4.10: 6 different variations of training on CIFAR-10 focusing on θ, where $\theta = [0, 60, 120, 180, 240, 300]$ from (a) through (f). The other Gabor parameters are static, where $\sigma = 10$, $\lambda = 50$, $\gamma = 150$, $\psi = 0$ for all variations of the Gabor filter bank. There is no dropout added to the network

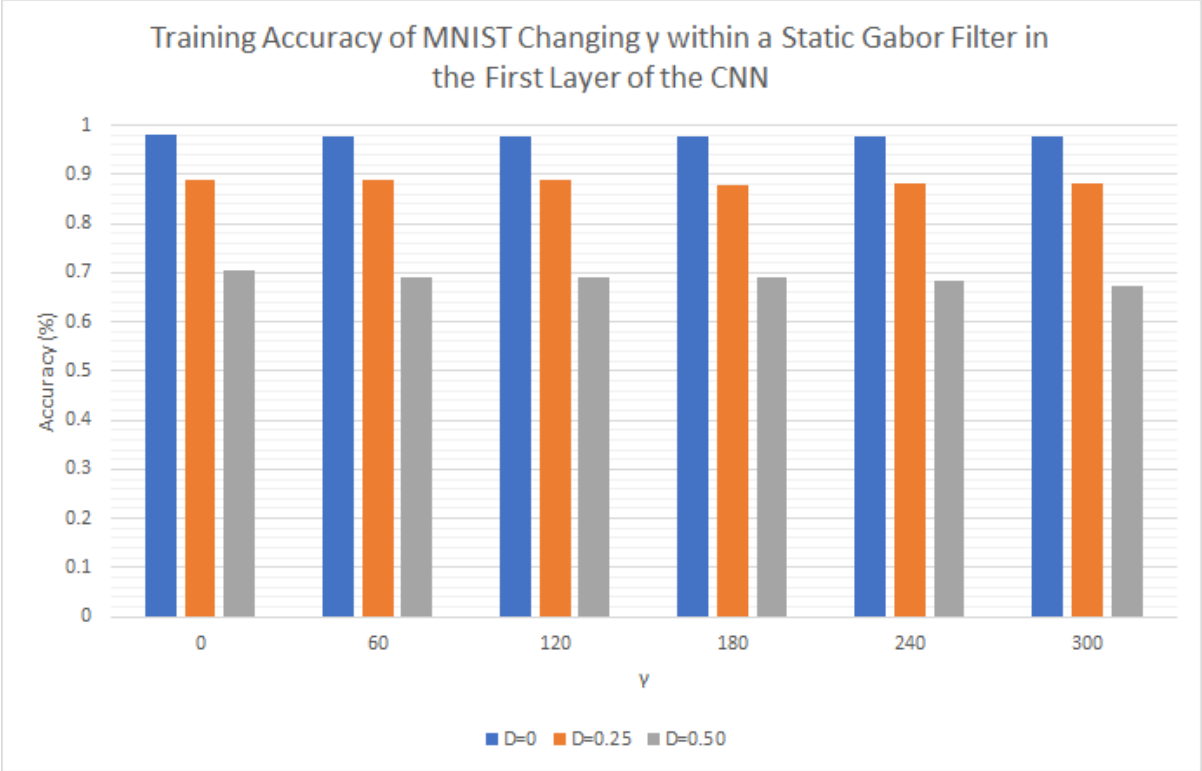
Gamma, γ

Gabor Parameters	Dataset	Dropout	Gamma	Training		Validation	
				Accuracy	Loss	Accuracy	Loss
$ksize = 5 \times 5$ $sigma = 10$ $lambda = 50$ $theta = 0$ $psi = 0$	MNIST	0	**0**	**0.9808**	**0.0628**	**0.9786**	**0.0669**
			60	0.9779	0.0724	0.9814	0.0599
			120	0.9777	0.0724	0.9796	0.0659
			180	0.9773	0.0736	0.9803	0.0612
			240	0.9781	0.0712	0.9798	0.0651
			300	0.9776	0.0734	0.9799	0.0656
		0.25	0	0.8879	0.3287	0.9728	0.1016
			60	0.8892	0.3220	0.9766	0.0915
			120	**0.8897**	**0.3283**	**0.9729**	**0.1034**
			180	0.8769	0.3481	0.9726	0.1006
			240	0.8832	0.3360	0.9741	0.0945
			300	0.8834	0.3422	0.9717	0.1048
		0.5	**0**	**0.7041**	**0.7990**	**0.9617**	**0.2101**
			60	0.6918	0.8105	0.9660	0.2081
			120	0.6919	0.8055	0.9622	0.2023
			180	0.6908	0.8172	0.9605	0.2128
			240	0.6847	0.8094	0.9633	0.1931
			300	0.6737	0.8535	0.9609	0.2306

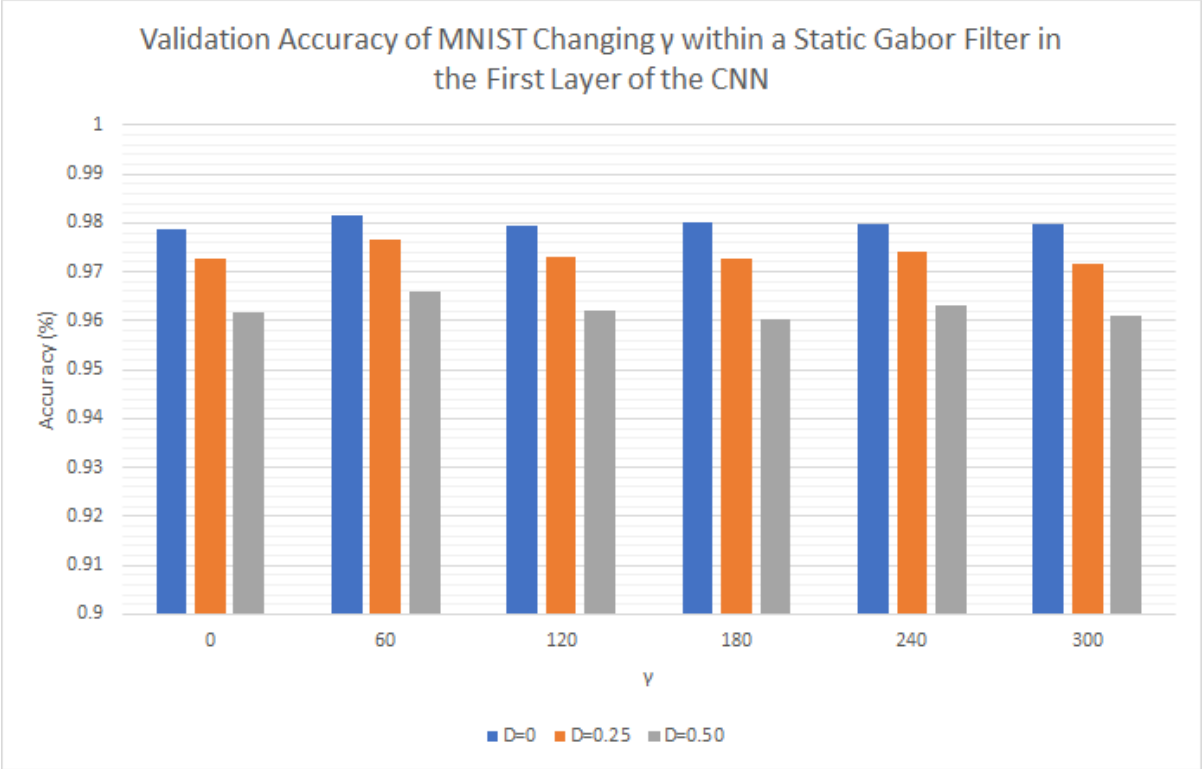
Table 4.10: Running a simple test showing a static first layer using a Gabor filter bank, testing on the MNIST dataset. The layers are able to learn during training. In this test, γ is the only variable that is changing in a range given as $[0, 60, 120, 180, 240, 300]$ inclusive. γ changes 6 times for each different dropout given.

Gabor Parameters	Dataset	Dropout	Gamma	Training		Validation	
				Accuracy	Loss	Accuracy	Loss
$ksize = 5 \times 5$ $sigma = 10$ $lambda = 50$ $theta = 0$ $psi = 0$	CIFAR-10	0	0	0.0982	2.3026	0.1000	2.3026
			60	**0.4279**	**1.6067**	**0.4272**	**1.6145**
			120	0.4285	1.6015	0.4301	1.5953
			180	0.4069	1.6570	0.4041	1.6559
			240	0.4205	1.6195	0.4216	1.6144
			300	0.3521	1.7696	0.3398	1.7873
		0.25	0	0.0979	4.1272	0.1002	4.7433
			60	0.2689	1.9685	0.3354	1.8564
			120	**0.2952**	**1.9152**	**0.3560**	**1.8138**
			180	0.2723	1.9555	0.3394	1.8479
			240	0.2275	2.0638	0.2753	1.9882
			300	0.2467	2.0047	0.2983	1.9205
		0.5	0	0.1245	2.2561	0.1521	2.2269
			60	**0.1992**	**2.1114**	**0.2702**	**2.0227**
			120	0.1593	2.2106	0.2077	2.1591
			180	0.1956	2.1250	0.2756	2.0359
			240	0.1810	2.1504	0.2397	2.0795
			300	0.1775	2.1671	0.2441	2.0873

Table 4.11: Running a simple test showing a static first layer using a Gabor filter bank, testing on the CIFAR-10 dataset. The layers are able to learn during training. In this test, γ is the only variable that is changing in a range given as $[0, 60, 120, 180, 240, 300]$ inclusive. γ changes 6 times for each different dropout given.

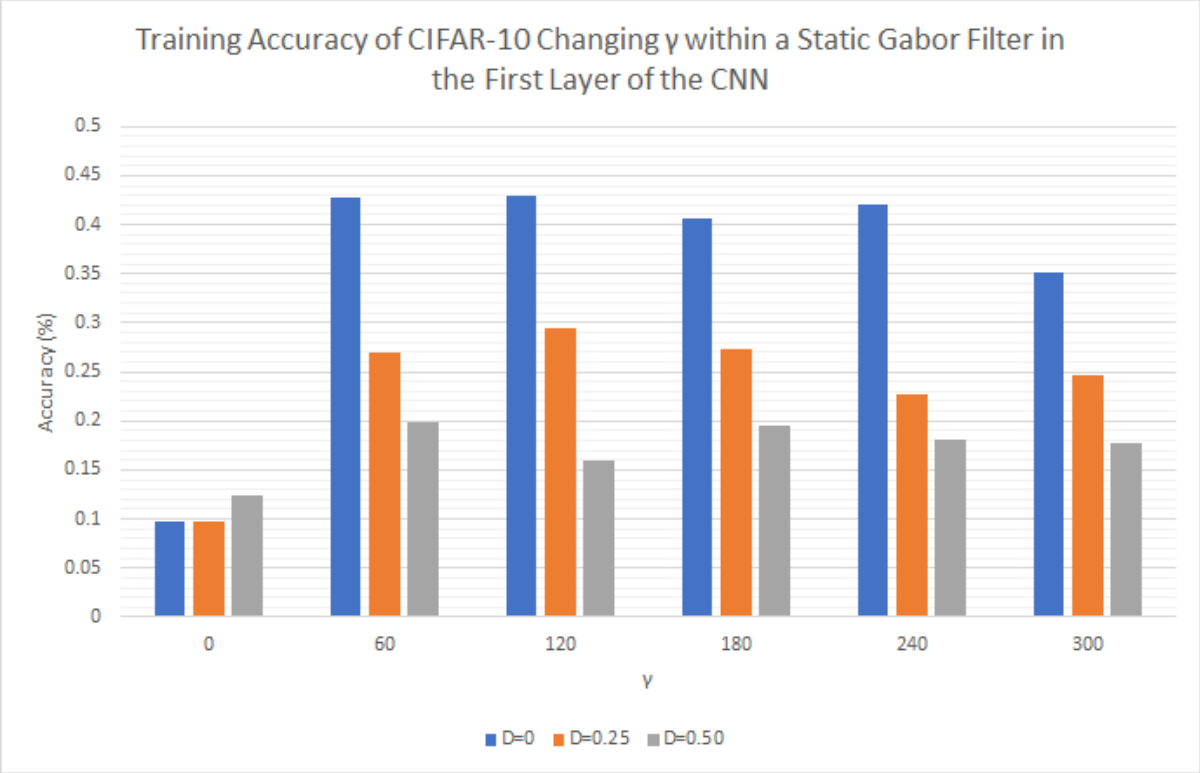


(a)

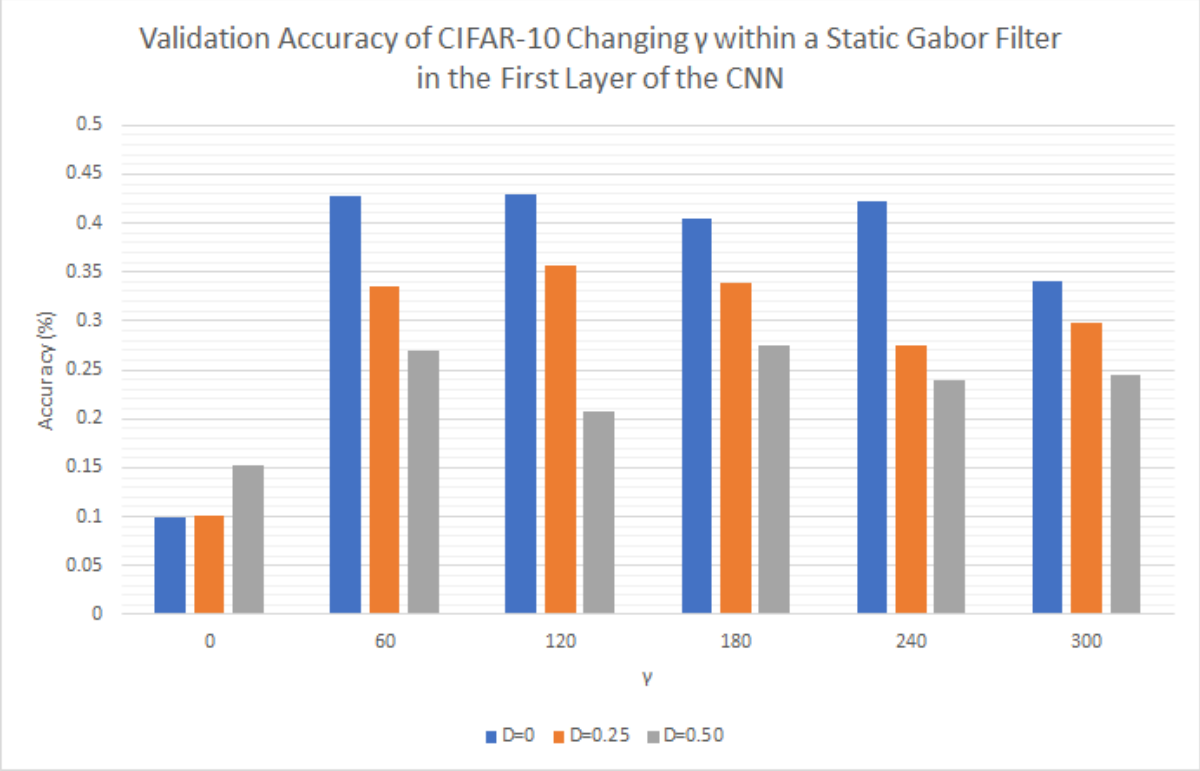


(b)

Figure 4.11: Here we see a different representation of Tables 4.8 showing the training and validation accuracy of the GCNN using MNIST for γ. Each colored bar in the legend shows the dropout rate used during training.



(a)



(b)

Figure 4.12: Here we see a different representation of Table 4.9 showing the training and validation accuracy of the GCNN using CIFAR-10 for γ. Each colored bar in the legend shows the dropout rate used during training.

Discussion

Gamma seems to have almost no effect on the MNIST dataset. The results seem to gradually go lower as we increase γ. The trend for adding dropout will reduce the results for both datasets. When we look at the CIFAR-10 dataset however, the response we get seem to change. $\gamma = 0$ does not fair well but as we increase the values, the results increase and remain steady. γ wants to be from the range of [60, 120] and outside of this range the results drop (but not significantly) for CIFAR-10 but for MNIST, the results are very close to each other.

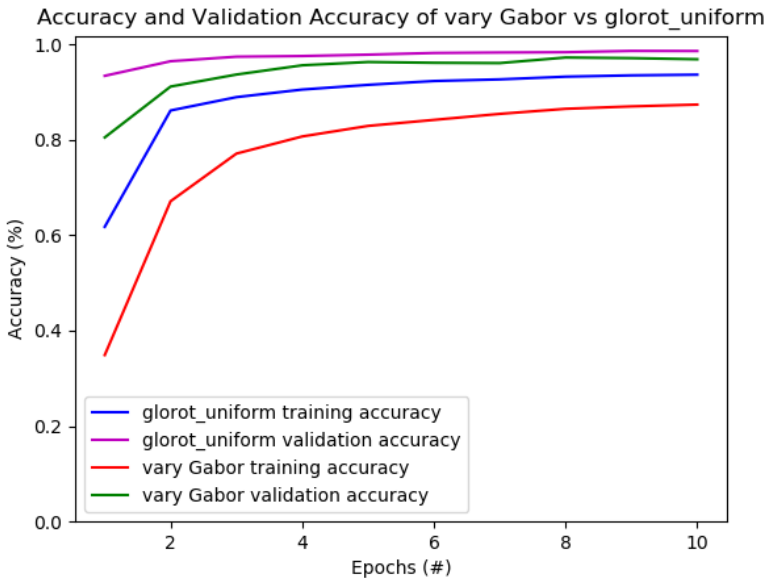


Figure 4.13: A test showing the training results per epoch for when $\gamma = 120$ and dropout, $D = 0.25$, when using the Gabor filter as initialization in the first layer. In comparison we can see how a Xavier initialization competes. In this static experiment, CIFAR-10 is the dataset being used to train on. The legend item that says "vary Gabor" signifies the results from when $\gamma = 120$.

Like the other parameters tested, it seems that there is no huge impact for having γ except for when testing with MNIST, $\gamma = 0$ should not be used. This is likely due to the added complexity of colour.

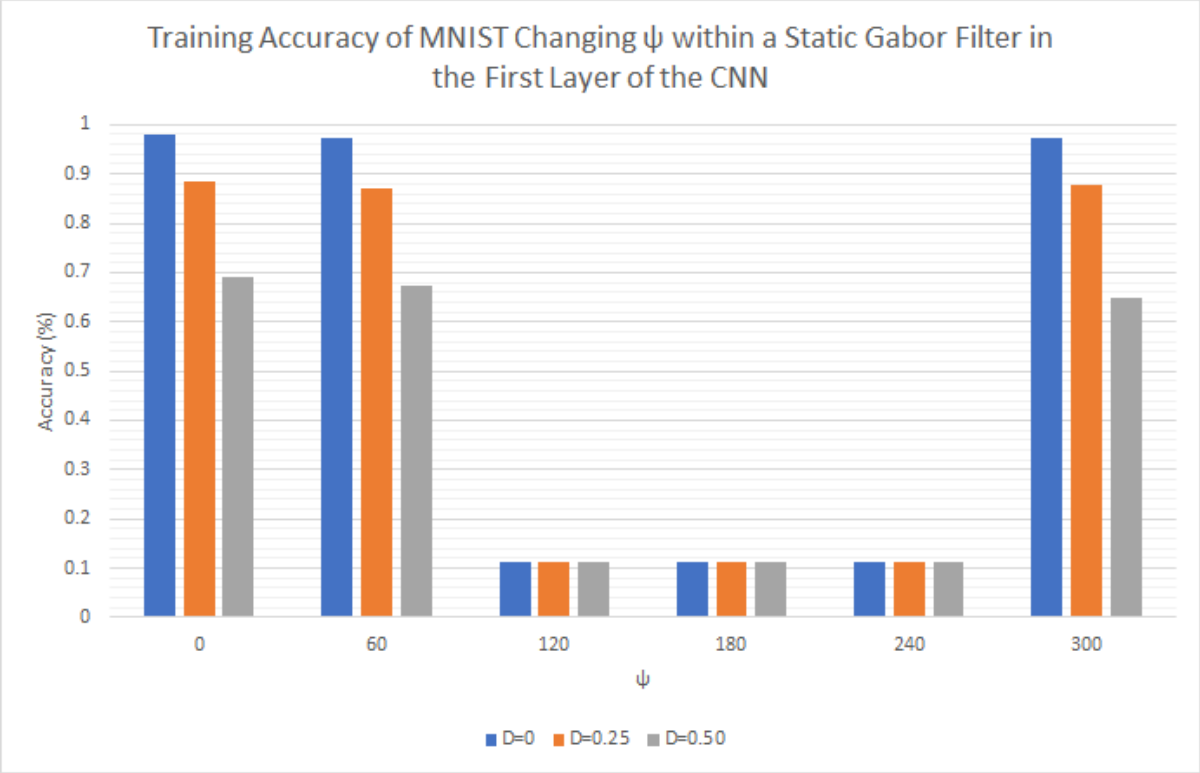
Psi, ψ

Gabor Parameters	Dataset	Dropout	Psi	Training		Validation	
				Accuracy	Loss	Accuracy	Loss
$ksize = 5 \times 5$ $sigma = 10$ $lambda = 50$ $theta = 0$ $gamma = 150$	MNIST	0	**0**	**0.9782**	**0.0719**	**0.9768**	**0.0769**
			60	0.9730	0.0873	0.9746	0.0808
			120	0.1124	2.3012	0.1135	2.3010
			180	0.1124	2.3012	0.1135	2.3010
			240	0.1124	2.3012	0.1135	2.3010
			300	0.9735	0.0871	0.9780	0.0708
		0.25	**0**	**0.8860**	**0.3266**	**0.9725**	**0.0997**
			60	0.8705	0.3867	0.9672	0.1303
			120	0.1124	2.3012	0.1135	2.3010
			180	0.1124	2.3012	0.1135	2.3010
			240	0.1124	2.3012	0.1135	2.3010
			300	0.8763	0.3616	0.9687	0.1168
		0.5	**0**	**0.6922**	**0.7809**	**0.9628**	**0.1785**
			60	0.6715	0.8472	0.9579	0.2370
			120	0.1124	2.3012	0.1135	2.3010
			180	0.1124	2.3012	0.1135	2.3010
			240	0.1124	2.3012	0.1135	2.3010
			300	0.6483	0.9060	0.9549	0.2682

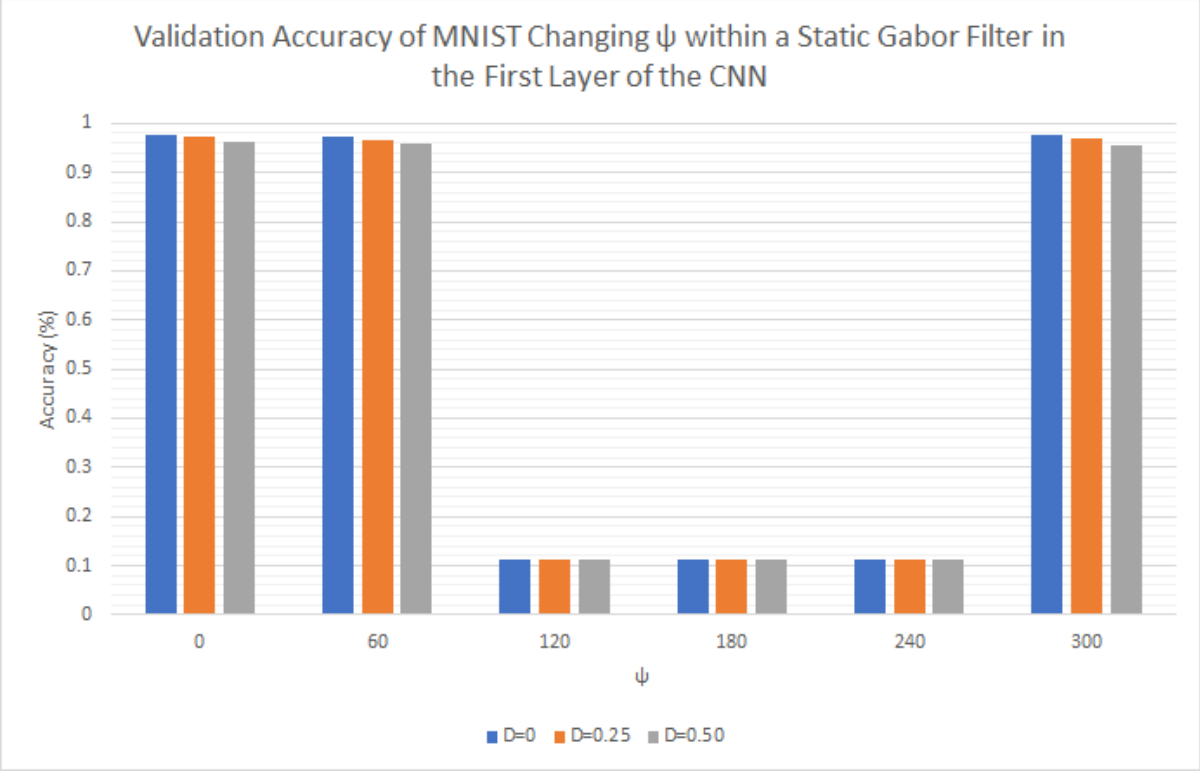
Table 4.12: Running a simple test showing a static first layer using a Gabor filter bank, testing on the MNIST dataset. The layers are able to learn during training. In this test, ψ is the only variable that is changing in a range given as $[0, 60, 120, 180, 240, 300]$ inclusive. ψ changes 6 times for each different dropout given.

Gabor Parameters	Dataset	Dropout	Psi	Training		Validation	
				Accuracy	Loss	Accuracy	Loss
$ksize = 5 \times 5$ $sigma = 10$ $lambda = 50$ $theta = 0$ gamma=150	MNIST	0	0	0.3834	1.7045	0.3925	1.6903
			60	0.4191	1.6293	0.4196	1.6343
			120	0.0973	2.3026	0.1000	2.3026
			180	0.0973	2.3026	0.1000	2.3026
			240	0.0981	2.3026	0.1000	2.3026
			300	**0.4198**	**1.6189**	**0.4300**	**1.6013**
		0.25	0	0.2102	2.0855	0.2527	2.0165
			60	0.2915	1.9097	0.3617	1.8041
			120	0.0965	2.3026	0.1000	2.3026
			180	0.0979	2.3026	0.1000	2.3026
			240	0.0965	2.3026	0.1000	2.3026
			300	**0.2960**	**1.8953**	**0.3595**	**1.7946**
		0.5	0	0.1892	2.1578	0.2511	2.0836
			60	0.1925	2.1228	0.2722	2.0299
			120	0.0976	2.3026	0.1000	2.3026
			180	0.0974	2.3026	0.1000	2.3026
			240	0.0972	2.3026	0.1000	2.3026
			300	**0.2200**	**2.0731**	**0.3106**	**1.9545**

Table 4.13: Running a simple test showing a static first layer using a Gabor filter bank, testing on the CIFAR-10 dataset. The layers are able to learn during training. In this test, ψ is the only variable that is changing in a range given as $[0, 60, 120, 180, 240, 300]$ inclusive. ψ changes 6 times for each different dropout given.

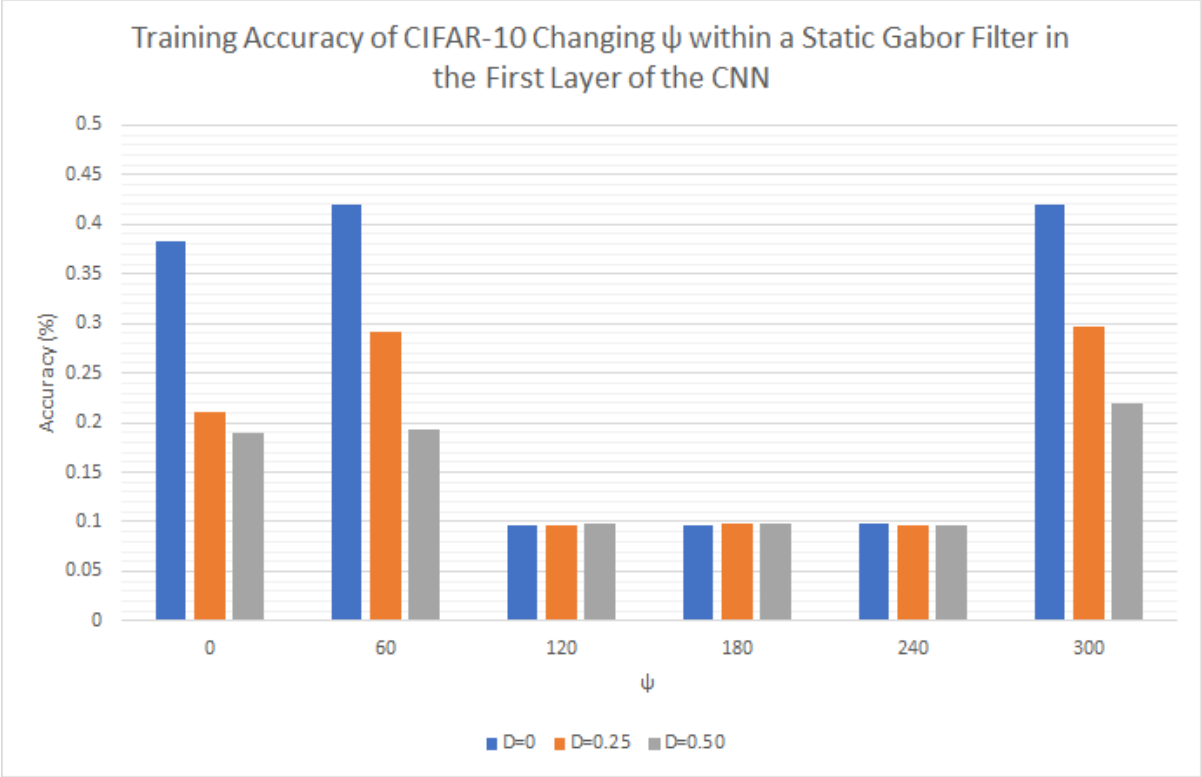


(a)

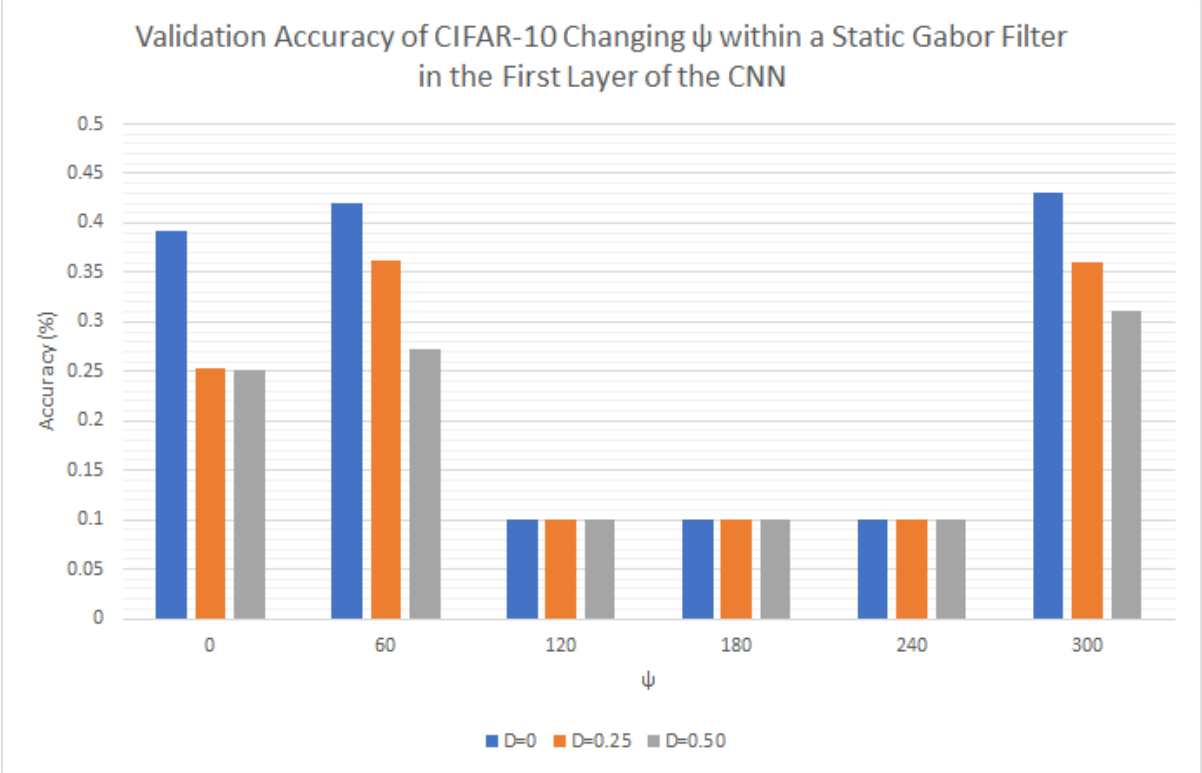


(b)

Figure 4.14: Here we see a different representation of Tables 4.8 showing the training and validation accuracy of the GCNN using MNIST for ψ. Each colored bar in the legend shows the dropout rate used during training.



(a)



(b)

Figure 4.15: Here we see a different representation of Table 4.9 showing the training and validation accuracy of the GCNN using CIFAR-10 for ψ. Each colored bar in the legend shows the dropout rate used during training.

Discussion

We can see from the Tables 4.12 and 4.13 that there is a range the ψ prefers. Values that are within the ranges $[120, 240]$ are values that ψ should never be in. The network has a very hard time training within these intervals. Otherwise, ψ follows the same trend as the other parameters where adding dropout tends to drop the results. When ψ is not within the range mentioned previously, the results are similar in that aspect and does not affect the dataset significantly, so long as it exists. The dataset chosen does not make a different but we must note that when training on CIFAR-10, the results drop significantly. This is likely due to the complexity of adding colour to the network.

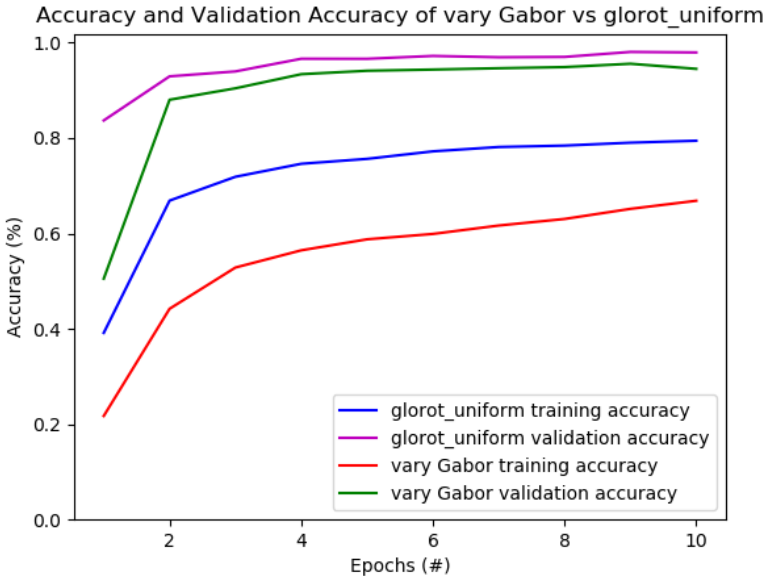


Figure 4.16: A test showing the training results per epoch for when $\psi = 0$ and dropout, $D = 0.5$, when using the Gabor filter as initialization in the first layer. In comparison we can see how a Xavier initialization competes. In this static experiment, MNIST is the dataset being used to train on. The legend item that says 'vary Gabor' signifies the results from when $\psi = 0$.

4.1.3 Summary

The parameters in the Gabor filter are responsive based on the 2-D mathematical expression given in Section 1.2.3. When σ, λ and γ are 0, the network fails to converge and does not train

(or does not train well). When $\gamma = 0$ for testing on MNIST, we see that the network is able to converge, but when testing on CIFAR-10, we see that it cannot converge at all. We also learned that when ψ is within a range of $[120, 240]$, the network fails to provide any meaningful results as well.

Because the Gabor filter depends on its rotational behaviour to find features,θ should have the most impact but in our test using single parameters for θ offer no noticeably different impact than other parameters. One thing that was found however, was that θ has higher results overall than other parameters in the CIFAR-10 dataset. The MNIST dataset is much too simple to compare the significance from other parameters.

We can also say that when training on a more complex dataset like CIFAR-10 versus MNIST, the results tend to drop in both testing and validation when there are more channels (colour).

In conclusion, σ, λ, and γ should not be equal to 0 ever. ψ should not be in a range between $[120 - 240]$ as far as this experiment shows. θ and ψ have the most impact overall . As for all other parameters, as long as they are > 0, they will now a lower impact than no impact at all.

Parameter	**Impact Overall**	**Best Found Value(s)**	**Worst Found Value(s)**
Sigma	Low	[10-18]	0
Lambda	Low	[20-80]	0
Theta	High	0	N/A
Gamma	Low	> 0	0
Psi	High	0, 300	[120-240]

Table 4.14: A summary of the impact of the parameters for using a static variable for 96 filters given in the first layer (i.e., A trial has 96 filters consisting of values of $\sigma = 2$, $\lambda = 40$, $\theta = 0$, $\gamma = 120$, and $\psi = 0$). The parameters that have a larger impact overall are θ and ψ. The other paremeters offer no noticeable impact to the final results. Values that appear to provide higher results are listed in the "best found value(s)" column and values that provide lower results are listed in the "worst found value(s)" column.

4.2 B: Search of a Single Parameter in the Gabor Filter Training on MNIST, CIFAR-10

4.2.1 Experiment Setup

For the second experiment, we will be using a Gabor filter bank that assumes a grid search for one variable at a time. The other variables will remain static (like experiment A). The values that the parameters will take are values within Table 4.2. Within the ranges given, the parameters will be equally divided by the number of filters generated. For example, if there are 32 filters that layer 1 uses and the range given is between $[1, 32]$ inclusive, there will be 32 values equally distributed (i.e., $I = [1, 2, 3, ..., 32]$). Each test will have 10 epochs per trial with a total of 5 trials.

The Gabor filter bank will change how many filters are generated per test. There will be 3 different variations in layer 1 that will have either 32, 64, or 96 Gabor kernels which will impact how spaced out the grid search is within each variable (refer to example given above). Additionally, dropout will be added to each variation.

MNIST, CIFAR-10, and CIFAR-100 are used in each of the 5 trials and each test is shuffled so that none are the same. Like the first experiment, the averages of each test will be recorded which includes the training accuracy and loss, and the validation accuracy and loss.

Test Case	# of Filters	Dropout	Sigma	Lambda	Theta	Gamma	Psi
A: Sigma	[16,32,48,64,80,96]	[0,0.25,0.5]	[2,22]	50	0	150	0
B: Lambda	[16,32,48,64,80,96]	[0,0.25,0.5]	10	[0,100]	0	150	0
C: Theta	[16,32,48,64,80,96]	[0,0.25,0.5]	10	50	[0,300]	150	0
D: Gamma	[16,32,48,64,80,96]	[0,0.25,0.5]	10	50	0	[0,300]	0
E: Psi	[16,32,48,64,80,96]	[0,0.25,0.5]	10	50	0	150	[0,300]

Table 4.15: Each tests from A through E listed, one parameter will change will the other parameters will remain static. Dropout has 3 test cases, 0%, 25%, and 50%. The other parameters (σ, λ, θ, γ, ψ) will be distributed evenly by dividing the maximum and the minimum (inclusive) by the number of filters. Therefore in each test case, we have a total of 18 tests. This experiment has a grand total of 90 tests.

4.2.2 Results

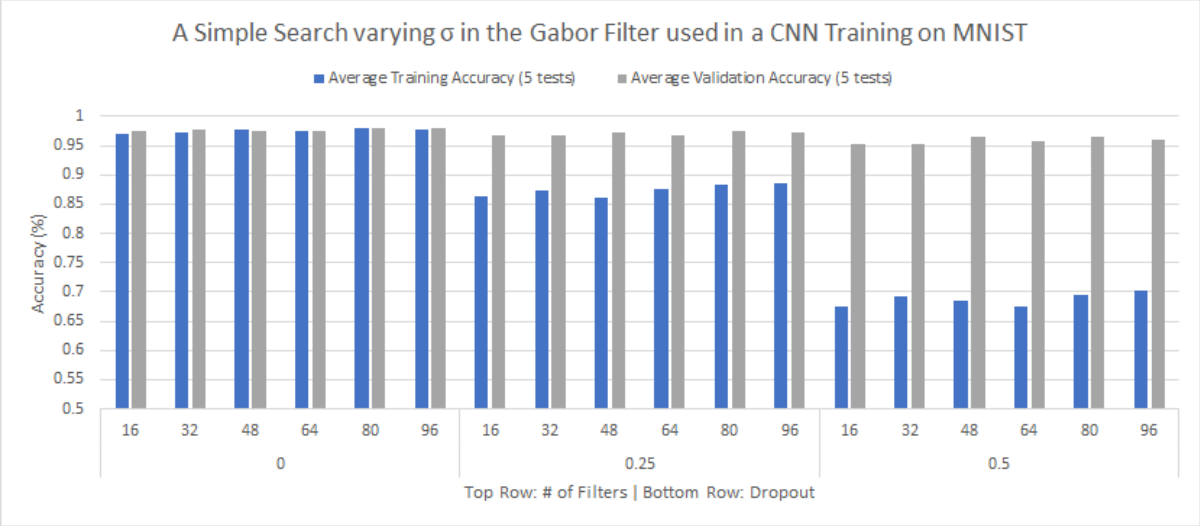
Sigma, σ

Gabor Parameters	Dataset	Dropout	# of Filters	Training		Validation	
				Accuracy	Loss	Accuracy	Loss
$ksize = 5 \times 5$ $sigma = vary$ $lambda = 50$ $theta = 0$ $gamma = 150$ psi=0	MNIST	0	16	0.9708	0.0962	0.9746	0.0795
			32	0.9735	0.0882	0.9778	0.0717
			48	0.9763	0.0782	0.9755	0.0776
			64	0.9750	0.0813	0.9738	0.0822
			80	**0.9790**	**0.0695**	**0.9787**	**0.0674**
			96	0.9778	0.0728	0.9810	0.0627
		0.25	16	0.8636	0.3985	0.9669	0.1281
			32	0.8733	0.3665	0.9678	0.1214
			48	0.8601	0.3877	0.9732	0.1097
			64	0.8748	0.3685	0.9670	0.1199
			80	0.8838	0.3391	0.9740	0.0993
			96	**0.8865**	**0.3304**	**0.9726**	**0.1046**
		0.5	16	0.6743	0.8745	0.9523	0.2651
			32	0.6916	0.8408	0.9532	0.2498
			48	0.6849	0.8280	0.9650	0.2006
			64	0.6744	0.8606	0.9572	0.2356
			80	0.6948	0.7848	0.9654	0.1783
			96	**0.7033**	**0.7893**	**0.9608**	**0.1994**

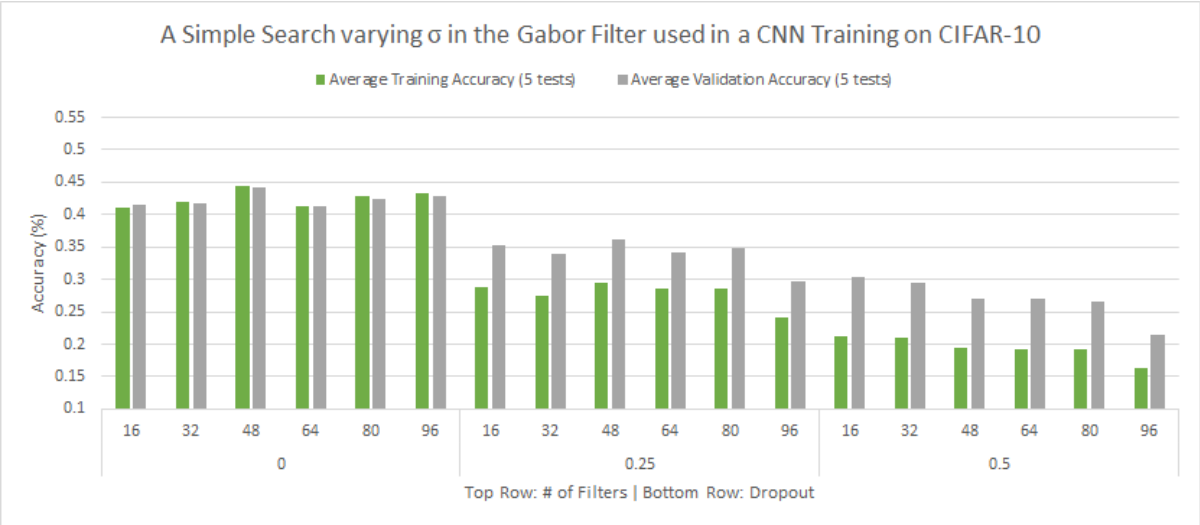
Table 4.16: Running a simple test showing a varying, given first layer using a Gabor filter bank, testing on the MNIST dataset. The layers are able to learn during training. In this test, σ is the only variable that is given a range changing from the intervals given in Table 4.2. The test here is dependent on the number of filters given in the first layer ranging from $[16, 32, 48, 64, 80, 96]$ inclusive. Sigma changes 6 times based on the number of filters for each different dropout given.

Gabor Parameters	Dataset	Dropout	# of Filters	Training		Validation	
				Accuracy	Loss	Accuracy	Loss
$ksize = 5 \times 5$ $sigma = vary$ $lambda = 50$ $theta = 0$ $gamma = 150$ $psi = 0$	CIFAR-10	0	16	0.4110	1.6471	0.4162	1.6305
			32	0.4198	1.6273	0.4181	1.6313
			48	**0.4441**	**1.5618**	**0.4419**	**1.5571**
			64	0.4125	1.6441	0.4137	1.6456
			80	0.4282	1.6042	0.4250	1.6129
			96	0.4319	1.5954	0.4293	1.5994
		0.25	16	0.2892	1.9167	0.3529	1.8191
			32	0.2756	1.9519	0.3391	1.8459
			48	**0.2943**	**1.9156**	**0.3614**	**1.8123**
			64	0.2858	1.9335	0.3420	1.8310
			80	0.2867	1.9252	0.3490	1.8135
			96	0.2409	2.0216	0.2969	1.9354
		0.5	**16**	**0.2122**	**2.0982**	**0.3033**	**1.9842**
			32	0.2100	2.0939	0.2937	1.9855
			48	0.1954	2.1150	0.2706	2.0196
			64	0.1927	2.1428	0.2700	2.0558
			80	0.1925	2.1313	0.2669	2.0390
			96	0.1635	2.1823	0.2153	2.1114

Table 4.17: Running a simple test showing a varying, given first layer using a Gabor filter bank, testing on the CIFAR-10 dataset. The layers are able to learn during training. In this test, σ is the only variable that is given a range changing from the intervals given in Table 4.2. The test here is dependent on the number of filters given in the first layer ranging from $[16, 32, 48, 64, 80, 96]$ inclusive. Sigma changes 6 times based on the number of filters for each different dropout given.



(a)



(b)

Figure 4.17: Here we see a different representation of Tables 4.16 and 4.17 showing the training and validation accuracy of the GCNN using MNIST and CIFAR-10 for σ. Each colored bar in the legend shows the training and validation results. The Gabor filter bank has dropout and a fixed number of filters.

Discussion

compare top results in both mnist and cifar. is there a pattern? are the results close to each other? conclude if using a lower filter count makes more sense or not

For the MNIST dataset, there appears to be a pattern where if we increase the number of

filters for σ, the results are better. In this case, we have higher scores when there are more filters but in CIFAR-10, the opposite appears to happen (except for when dropout is 0%. In 4 of the 6 different scenarios of dropout, the trend seems to want to increase the results performance if there are more filters, i.e., more variations of the Gabor filter. When dropout is added in the mix, results drop for both datasets. This happens more in the CIFAR-10 database but is not as noticeable in the MNIST database. This is, like experiment A, likely due to the added complexity of colours. The results are all within a close range of each other so we can say that by using less filters, we use less parameters and as a result obtain similar performance.

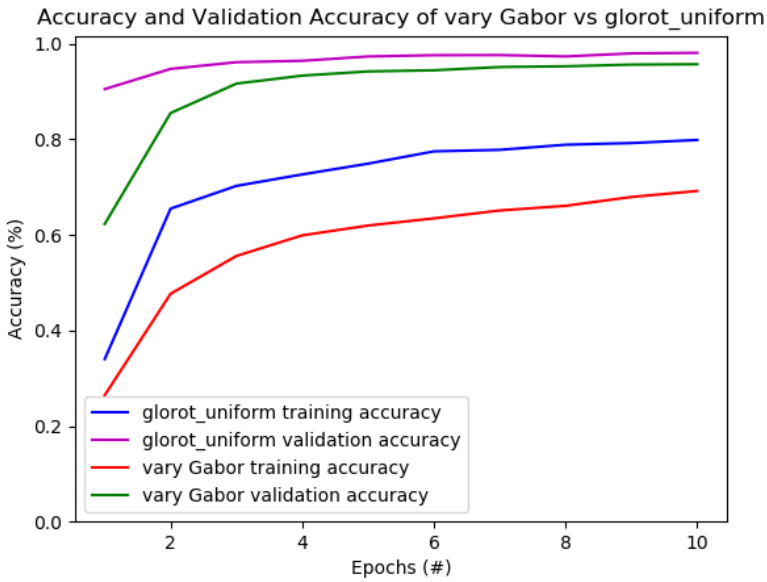


Figure 4.18: A test showing the training results per epoch for σ when the number of filters, $F = 96$ and dropout, $D = 0.5$, when using the Gabor filter as initialization in the first layer. In comparison we can see how a Xavier initialization competes. In this experiment for using a grid search for one parameter, MNIST is the dataset being used to train on. The legend item that says 'vary Gabor' signifies using σ as the parameter being tested on.

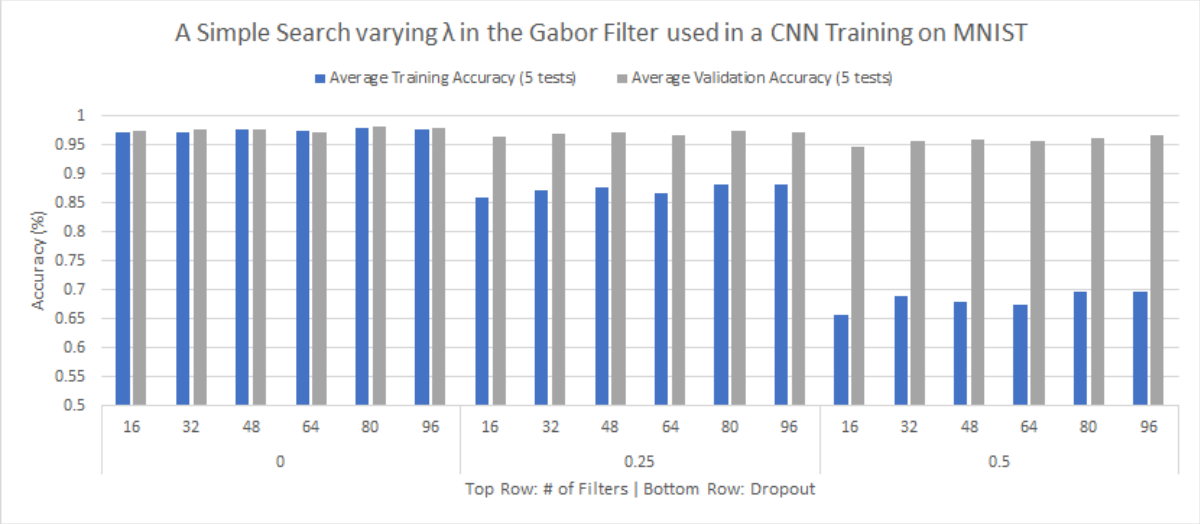
Lambda, λ

Gabor Parameters	Dataset	Dropout	# of Filters	Training		Validation	
				Accuracy	Loss	Accuracy	Loss
$ksize = 5 \times 5$ $sigma = 10$ $lambda = vary$ $theta = 0$ $gamma = 150$ $psi = 0$	MNIST	0	16	0.9712	0.0959	0.9737	0.0834
			32	0.9730	0.0883	0.9778	0.0707
			48	0.9771	0.0754	0.9774	0.0701
			64	0.9745	0.0837	0.9711	0.0903
			80	**0.9802**	**0.0649**	**0.9815**	**0.0596**
			96	0.9781	0.0712	0.9791	0.0641
		0.25	16	0.8591	0.4136	0.9652	0.1399
			32	0.8711	0.3766	0.9688	0.1177
			48	0.8780	0.3556	0.9721	0.1014
			64	0.8660	0.3933	0.9669	0.1273
			80	**0.8828**	**0.3432**	**0.9753**	**0.0933**
			96	0.8812	0.3566	0.9732	0.0996
		0.5	16	0.6562	0.9125	0.9479	0.3092
			32	0.6884	0.8337	0.9575	0.2421
			48	0.6796	0.8524	0.9601	0.2452
			64	0.6745	0.8380	0.9581	0.2303
			80	0.6958	0.8103	0.9620	0.2168
			96	**0.6967**	**0.7836**	**0.9673**	**0.1629**

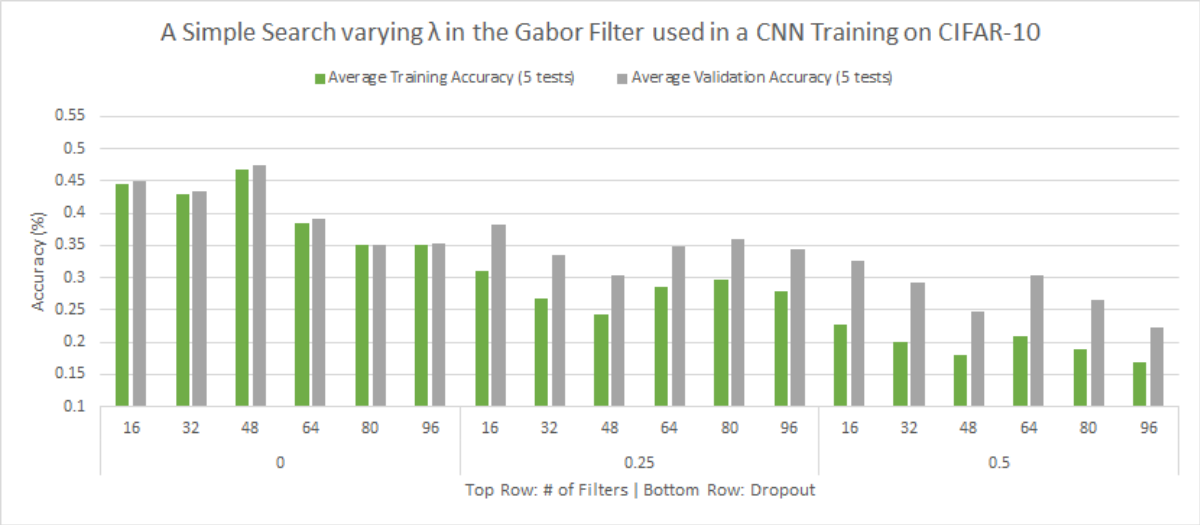
Table 4.18: Running a simple test showing a varying, given first layer using a Gabor filter bank, testing on the MNIST dataset. The layers are able to learn during training. In this test, λ is the only variable that is given a range changing from the intervals given in Table 4.2. The test here is dependent on the number of filters given in the first layer ranging from $[16, 32, 48, 64, 80, 96]$ inclusive. Lambda changes 6 times based on the number of filters for each different dropout given.

Gabor Parameters	Dataset	Dropout	# of Filters	Training		Validation	
				Accuracy	Loss	Accuracy	Loss
$ksize = 5 \times 5$ $sigma = 10$ $lambda = vary$ $theta = 0$ $gamma = 150$ $psi = 0$	CIFAR-10	0	16	0.4454	1.5413	0.4504	1.5242
			32	0.4295	1.5954	0.4342	1.5851
			48	**0.4684**	**1.4984**	**0.4745**	**1.4832**
			64	0.3834	1.7147	0.3920	1.7000
			80	0.3499	1.7724	0.3500	1.7747
			96	0.3507	1.7704	0.3534	1.7674
		0.25	**16**	**0.3100**	**1.8516**	**0.3822**	**1.7262**
			32	0.2685	1.9555	0.3341	1.8457
			48	0.2431	2.0343	0.3046	1.9286
			64	0.2859	1.9281	0.3489	1.8330
			80	0.2959	1.9017	0.3599	1.7996
			96	0.2799	1.9393	0.3444	1.8353
		0.5	**16**	**0.2263**	**2.0518**	**0.3259**	**1.9344**
			32	0.2010	2.1097	0.2920	2.0014
			48	0.1795	2.1630	0.2482	2.0866
			64	0.2083	2.1125	0.3035	2.0025
			80	0.1880	2.1368	0.2663	2.0396
			96	0.1689	2.1775	0.2228	2.1058

Table 4.19: Running a simple test showing a varying, given first layer using a Gabor filter bank, testing on the CIFAR-10 dataset. The layers are able to learn during training. In this test, λ is the only variable that is given a range changing from the intervals given in Table 4.2. The test here is dependent on the number of filters given in the first layer ranging from $[16, 32, 48, 64, 80, 96]$ inclusive. Lambda changes 6 times based on the number of filters for each different dropout given.



(a)



(b)

Figure 4.19: Here we see a different representation of Tables 4.18 and 4.19 showing the training and validation accuracy of the GCNN using MNIST and CIFAR-10 for λ. Each colored bar in the legend shows the training and validation results. The Gabor filter bank has dropout and a fixed number of filters.

Discussion

For the MNIST dataset, like the σ test, we see the same pattern where the training on MNIST is more accurate as we add more filters (Table 4.18). It appears to show a progression where the number of filters influence the performance but as a reminder, with more filters we add

more complexity and parameters to the entire network. The results are very close to each other, for example, in a test running Dropout, $D = 0$, the difference between having 16 filters and 96 filters is 0.71%. This means that we can sacrifice the number of filters for MNIST to obtain reasonable results. That said, MNIST is a very simple dataset.

For CIFAR-10, like experiment A, we can see that it follows a similar pattern where having a lower number of filters results in a higher performance. By adding more filters, the performance drops gradually as well. We can say that using less filters likely produces better results. The complexity of colour in CIFAR-10 is likely the factor that reduces the performance of the CNN. Both MNIST and CIFAR-10 produce similar performance results.

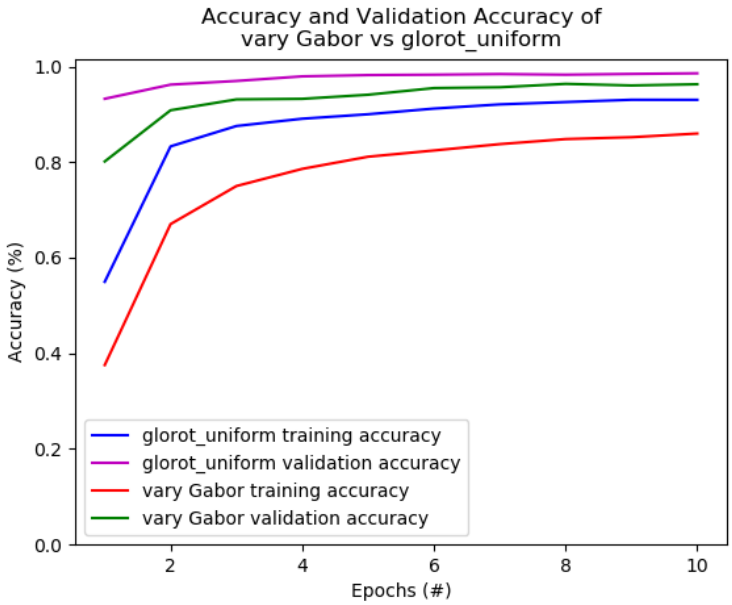


Figure 4.20: A test showing the training results per epoch for λ when the number of filters, $F = 64$ and dropout, $D = 0$, when using the Gabor filter as initialization in the first layer. In comparison we can see how a Xavier initialization competes. In this experiment for using a grid search for one parameter, MNIST is the dataset being used to train on. The legend item that says 'vary Gabor' signifies using λ as the parameter being tested on.

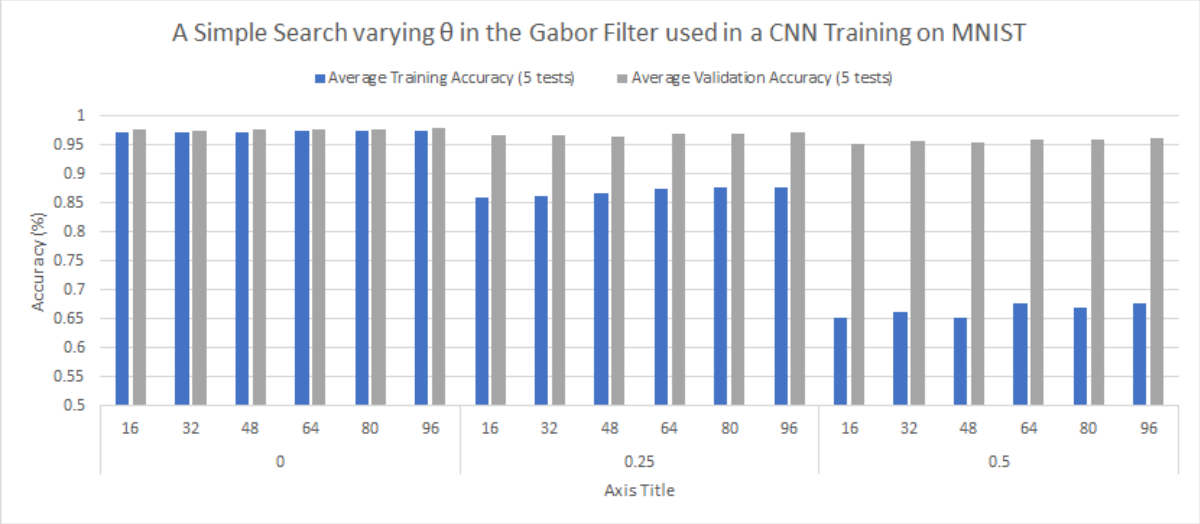
Theta, θ

Gabor Parameters	Dataset	Dropout	# of Filters	Training		Validation	
				Accuracy	Loss	Accuracy	Loss
$ksize = 5 \times 5$ $sigma = 10$ $lambda = 50$ $theta = vary$ $gamma = 150$ $psi = 0$	MNIST	0	16	0.9726	0.0900	0.9763	0.0773
			32	0.9731	0.0897	0.9759	0.0760
			48	0.9733	0.0896	0.9765	0.0763
			64	0.9748	0.0844	0.9774	0.0713
			80	0.9735	0.0879	0.9778	0.0725
			96	**0.9753**	**0.0814**	**0.9797**	**0.0649**
		0.25	16	0.8607	0.3961	0.9674	0.1338
			32	0.8626	0.4006	0.9665	0.1320
			48	0.8661	0.3853	0.9653	0.1309
			64	0.8740	0.3738	0.9701	0.1147
			80	0.8766	0.3697	0.9704	0.1167
			96	**0.8767**	**0.3610**	**0.9719**	**0.1084**
		0.5	16	0.6535	0.8978	0.9530	0.2744
			32	0.6629	0.8880	0.9565	0.2544
			48	0.6532	0.9102	0.9540	0.2922
			64	0.6762	0.8431	0.9606	0.2272
			80	0.6694	0.8576	0.9598	0.2243
			96	**0.6773**	**0.8427**	**0.9616**	**0.2204**

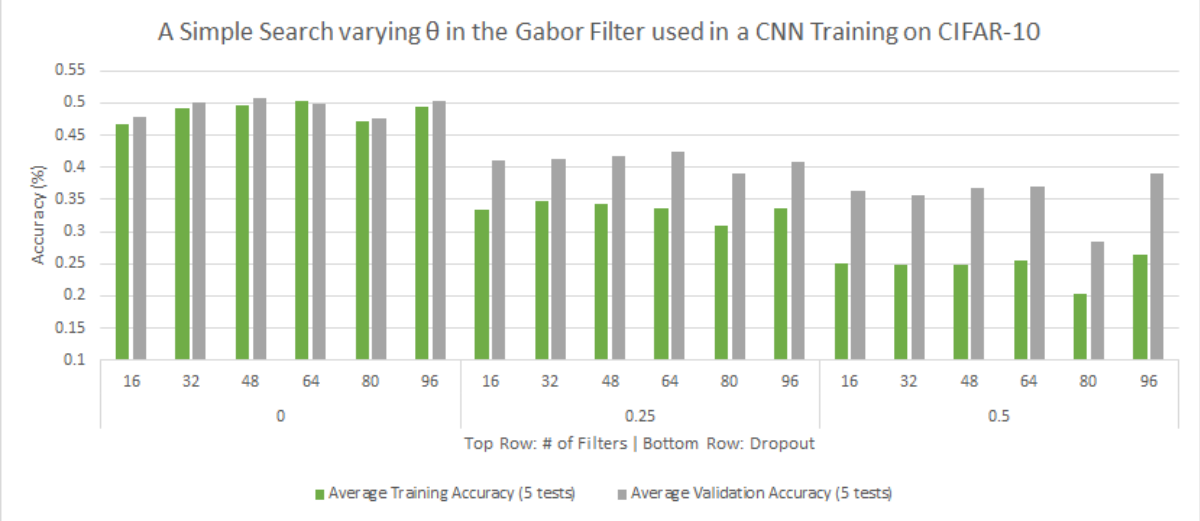
Table 4.20: Running a simple test showing a varying, given first layer using a Gabor filter bank, testing on the MNIST dataset. The layers are able to learn during training. In this test, θ is the only variable that is given a range changing from the intervals given in Table 4.2. The test here is dependent on the number of filters given in the first layer ranging from $[16, 32, 48, 64, 80, 96]$ inclusive. Theta changes 6 times based on the number of filters for each different dropout given.

Gabor Parameters	Dataset	Dropout	# of Filters	Training		Validation	
				Accuracy	Loss	Accuracy	Loss
$ksize = 5 \times 5$ $sigma = 10$ $lambda = 50$ $theta = vary$ $gamma = 150$ $psi = 0$	CIFAR-10	0	16	0.4659	1.4668	0.4786	1.4494
			32	0.4917	1.4017	0.5009	1.3729
			48	0.4966	1.3855	0.5078	1.3617
			64	**0.5028**	**1.3705**	**0.4980**	**1.3781**
			80	0.4714	1.4690	0.4766	1.4578
			96	0.4936	1.3918	0.5020	1.3579
		0.25	16	0.3340	1.7648	0.4099	1.6414
			32	**0.3474**	**1.7284**	**0.4117**	**1.6189**
			48	0.3422	1.7339	0.4167	1.6174
			64	0.3369	1.7614	0.4236	1.6228
			80	0.3095	1.8337	0.3895	1.7039
			96	0.3366	1.7618	0.4089	1.6477
		0.5	16	0.2494	1.9352	0.3630	1.7780
			32	0.2487	1.9577	0.3568	1.8067
			48	0.2474	1.9645	0.3679	1.8047
			64	0.2546	1.9435	0.3693	1.7963
			80	0.2041	2.0790	0.2850	1.9720
			96	**0.2634**	**1.9163**	**0.3904**	**1.7354**

Table 4.21: Running a simple test showing a varying, given first layer using a Gabor filter bank, testing on the CIFAR-10 dataset. The layers are able to learn during training. In this test, θ is the only variable that is given a range changing from the intervals given in Table 4.2. The test here is dependent on the number of filters given in the first layer ranging from $[16, 32, 48, 64, 80, 96]$ inclusive. Theta changes 6 times based on the number of filters for each different dropout given.



(a)



(b)

Figure 4.21: Here we see a different representation of Tables 4.20 and 4.21 showing the training and validation accuracy of the GCNN using MNIST and CIFAR-10 for θ. Each colored bar in the legend shows the training and validation results. The Gabor filter bank has dropout and a fixed number of filters.

Discussion

For this test, we see that the results from testing on MNIST and CIFAR-10 are much higher on average than the previous 2 tests. We also see the same similarities when testing on MNIST where the results perform better with more filters. The biggest difference we can see is that

when we test on the CIFAR-10, the results do not make any noticeable pattern except for having higher results than previous tests.

Using CIFAR-10 as the dataset, it is likely to have better scores when using more filters. This may be because of the rotational benefits of the Gabor filter extracting different features within the dataset. We can see this trend continue from the first experiment where θ obtains higher results than other parameters. Adding dropout to both training sets follow the trend of reducing the performances in both training and validation.

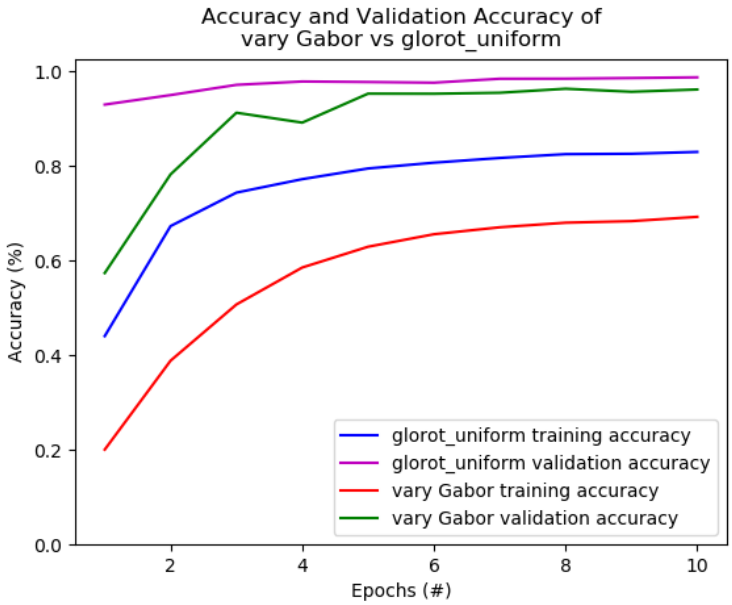


Figure 4.22: A test showing the training results per epoch for θ when the number of filters, $F = 96$ and dropout, $D = 0.5$, when using the Gabor filter as initialization in the first layer. In comparison we can see how a Xavier initialization competes. In this experiment for using a grid search for one parameter, MNIST is the dataset being used to train on. The legend item that says 'vary Gabor' signifies using θ as the parameter being tested on.

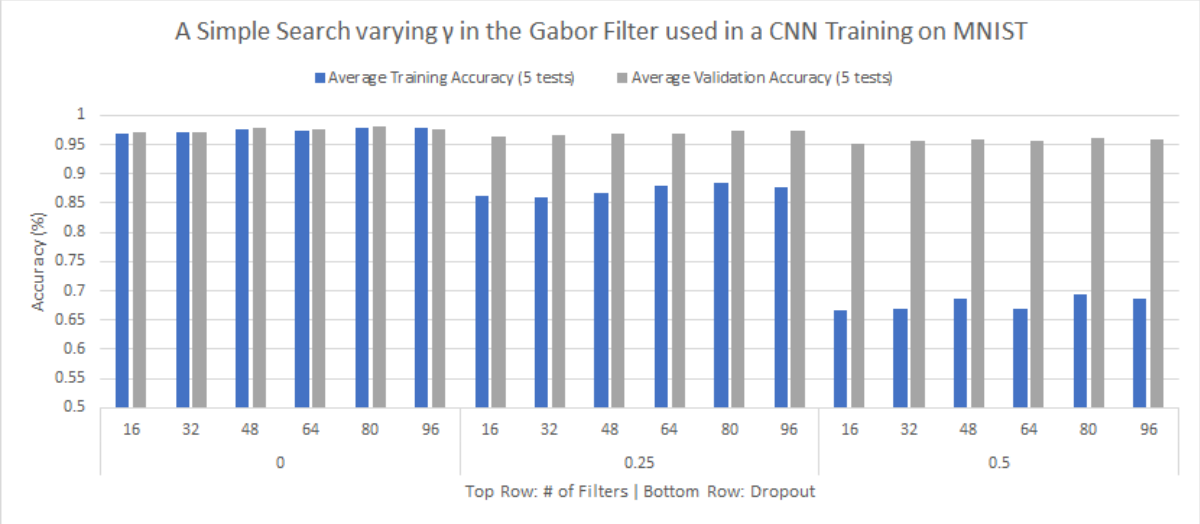
Gamma, γ

Gabor Parameters	Dataset	Dropout	# of Filters	Training		Validation	
				Accuracy	Loss	Accuracy	Loss
$ksize = 5 \times 5$ $sigma = 10$ $lambda = 50$ $theta = 0$ $gamma = vary$ $psi = 0$	MNIST	0	16	0.9695	0.0985	0.9718	0.0877
			32	0.9717	0.0932	0.9723	0.0859
			48	0.9760	0.0794	0.9789	0.0665
			64	0.9738	0.0846	0.9756	0.0777
			80	**0.9786**	**0.0698**	**0.9814**	**0.0613**
			96	0.9778	0.0718	0.9754	0.0763
		0.25	16	0.8632	0.4015	0.9636	0.1397
			32	0.8599	0.4036	0.9660	0.1321
			48	0.8685	0.3760	0.9685	0.1212
			64	0.8809	0.3641	0.9697	0.1193
			80	**0.8857**	**0.3362**	**0.9729**	**0.1082**
			96	0.8782	0.3463	0.9729	0.1010
		0.5	16	0.6664	0.8803	0.9522	0.2483
			32	0.6685	0.8548	0.9559	0.2339
			48	0.6861	0.8343	0.9597	0.2223
			64	0.6701	0.8472	0.9577	0.2283
			80	**0.6948**	**0.8137**	**0.9618**	**0.2217**
			96	0.6864	0.8154	0.9598	0.2008

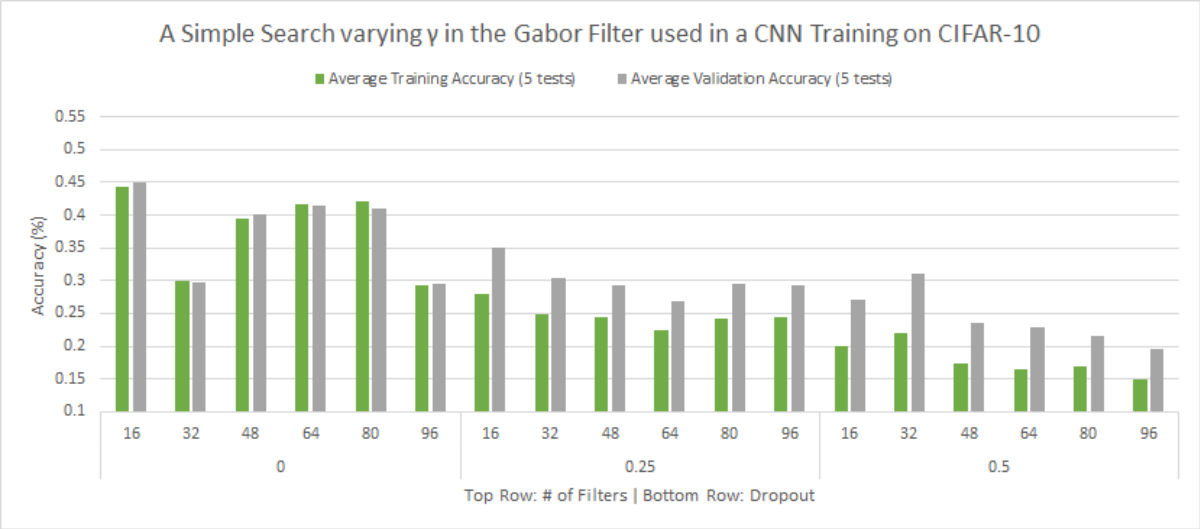
Table 4.22: Running a simple test showing a varying, given first layer using a Gabor filter bank, testing on the MNIST dataset. The layers are able to learn during training. In this test, γ is the only variable that is given a range changing from the intervals given in Table 4.2. The test here is dependent on the number of filters given in the first layer ranging from $[16, 32, 48, 64, 80, 96]$ inclusive. Gamma changes 6 times based on the number of filters for each different dropout given.

Gabor Parameters	Dataset	Dropout	# of Filters	Training		Validation	
				Accuracy	Loss	Accuracy	Loss
$ksize = 5 \times 5$ $sigma = 10$ $lambda = 50$ $theta = 0$ $gamma = vary$ $psi = 0$	CIFAR-10	0	**16**	**0.4428**	**1.5498**	**0.4501**	**1.5290**
			32	0.3000	1.8933	0.2971	1.9111
			48	0.3949	1.6785	0.4018	1.6588
			64	0.4161	1.6319	0.4138	1.6406
			80	0.4219	1.6189	0.4105	1.6404
			96	0.2927	1.8889	0.2948	1.8820
		0.25	**16**	**0.2806**	**1.9239**	**0.3509**	**1.8078**
			32	0.2477	2.0038	0.3049	1.9222
			48	0.2445	2.0153	0.2920	1.9374
			64	0.2255	2.0642	0.2677	1.9960
			80	0.2429	2.0052	0.2954	1.9205
			96	0.2432	2.0149	0.2931	1.9455
		0.5	16	0.1993	2.0989	0.2707	1.9950
			32	**0.2192**	**2.0676**	**0.3098**	**1.9539**
			48	0.1728	2.1830	0.2363	2.1074
			64	0.1656	2.1873	0.2293	2.1132
			80	0.1698	2.1853	0.2147	2.1345
			96	0.1486	2.2226	0.1958	2.1707

Table 4.23: Running a simple test showing a varying, given first layer using a Gabor filter bank, testing on the CIFAR-10 dataset. The layers are able to learn during training. In this test, γ is the only variable that is given a range changing from the intervals given in Table 4.2. The test here is dependent on the number of filters given in the first layer ranging from $[16, 32, 48, 64, 80, 96]$ inclusive. Gamma changes 6 times based on the number of filters for each different dropout given.



(a)



(b)

Figure 4.23: Here we see a different representation of Tables 4.22 and 4.23 showing the training and validation accuracy of the GCNN using MNIST and CIFAR-10 for γ. Each colored bar in the legend shows the training and validation results. The Gabor filter bank has dropout and a fixed number of filters.

Discussion

Gamma, γ, shows a pattern similar to λ for when we use MNIST, the results tend to get better as we increase the number of filters within the first layer. The results in MNIST are very close together in both training and validation. For example, when using 16 filters versus 80 filters,

the performance difference in terms of accuracy is 0.94% which is a very small gain in terms of performance vs complexity. Both datasets reduce performance when dropout is added.

We can also see the same trend where if we use the CIFAR-10 dataset, the results decrease as we add filters. This is likely due to the fact that CIFAR-10 has colour which adds additional complexity in the CNN. We can conclude in saying that γ is more optimized if we use less filters, especially if the dataset is coloured (or complex).

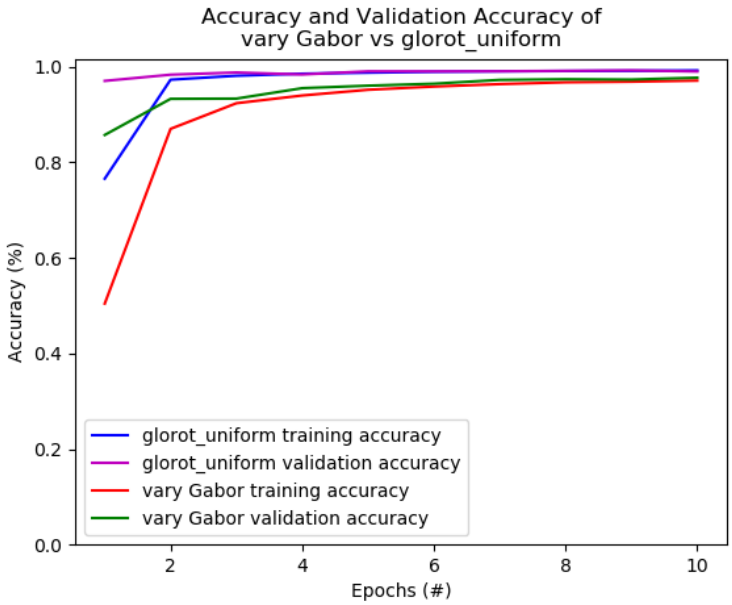


Figure 4.24: A test showing the training results per epoch for γ when the number of filters, $F = 80$ and dropout, $D = 0.25$, when using the Gabor filter as initialization in the first layer. In comparison we can see how a Xavier initialization competes. In this experiment for using a grid search for one parameter, MNIST is the dataset being used to train on. The legend item that says 'vary Gabor' signifies using γ as the parameter being tested on.

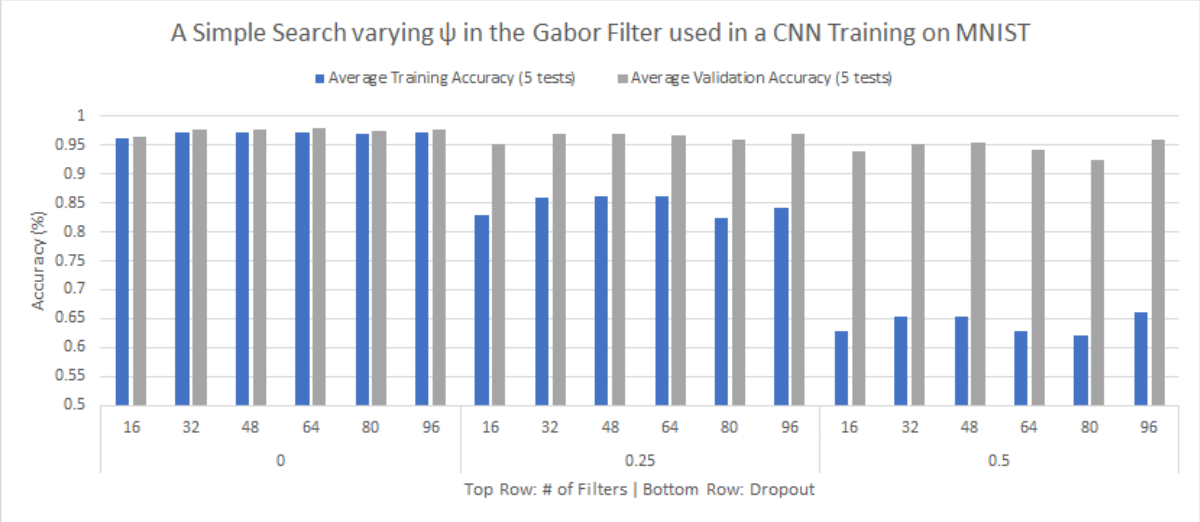
Psi, ψ

Gabor Parameters	Dataset	Dropout	# of Filters	Training		Validation	
				Accuracy	Loss	Accuracy	Loss
$ksize = 5 \times 5$ $sigma = 10$ $lambda = 50$ $theta = 0$ $gamma = 150$ $psi = vary$	MNIST	0	16	0.9607	0.1361	0.9642	0.1194
			32	0.9712	0.0963	0.9763	0.0755
			48	0.9721	0.0919	0.9764	0.0751
			64	**0.9728**	**0.0898**	**0.9782**	**0.0706**
			80	0.9691	0.1026	0.9743	0.0863
			96	0.9721	0.0927	0.9762	0.0770
		0.25	16	0.8297	0.4912	0.9528	0.2050
			32	0.8588	0.4126	0.9700	0.1282
			48	0.8610	0.3993	0.9685	0.1237
			64	**0.8614**	**0.4034**	**0.9677**	**0.1303**
			80	0.8250	0.4986	0.9584	0.1823
			96	0.8415	0.4433	0.9682	0.1314
		0.5	16	0.6287	0.9693	0.9401	0.3704
			32	0.6548	0.9009	0.9513	0.3150
			48	0.6544	0.9238	0.9538	0.3126
			64	0.6292	0.9594	0.9427	0.3331
			80	0.6205	0.9734	0.9244	0.3856
			96	**0.6609**	**0.8682**	**0.9596**	**0.2457**

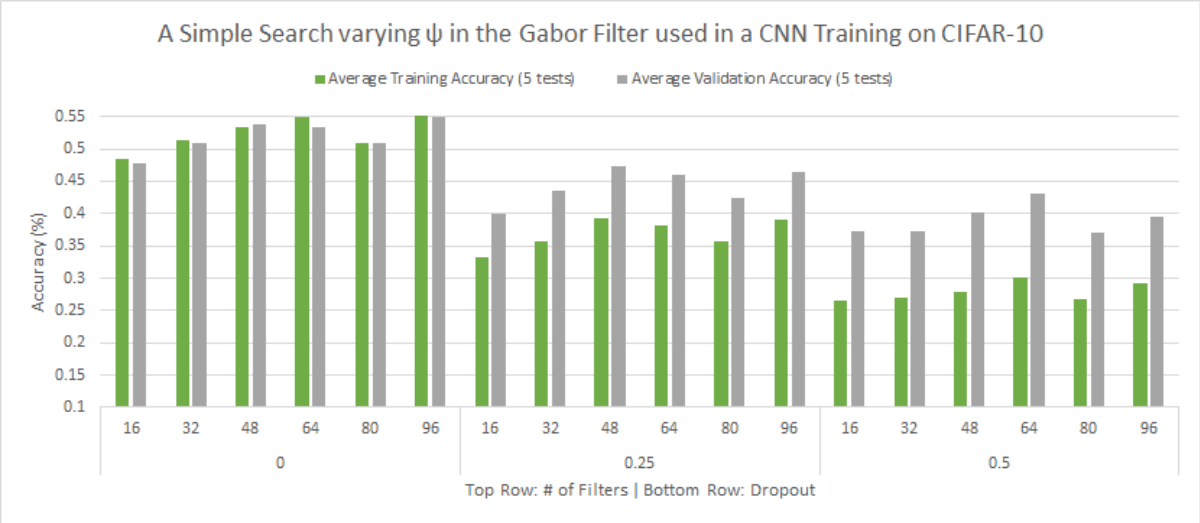
Table 4.24: Running a simple test showing a varying, given first layer using a Gabor filter bank, testing on the MNIST dataset. The layers are able to learn during training. In this test, ψ is the only variable that is given a range changing from the intervals given in Table 4.2. The test here is dependent on the number of filters given in the first layer ranging from $[16, 32, 48, 64, 80, 96]$ inclusive. Psi changes 6 times based on the number of filters for each different dropout given.

Gabor Parameters	Dataset	Dropout	# of Filters	Training		Validation	
				Accuracy	Loss	Accuracy	Loss
$ksize = 5 \times 5$ $sigma = 10$ $lambda = 50$ $theta = 0$ $gamma = 150$ $psi = vary$	CIFAR-10	0	16	0.4841	1.3909	0.4780	1.4067
			32	0.5138	1.3351	0.5094	1.3483
			48	0.5331	1.2814	0.5382	1.2712
			64	0.5483	1.2412	0.5328	1.2770
			80	0.5095	1.3547	0.5095	1.3452
			96	**0.5551**	**1.2249**	**0.5497**	**1.2379**
		0.25	16	0.3322	1.7267	0.4003	1.6046
			32	0.3566	1.6754	0.4348	1.5358
			48	**0.3936**	**1.5976**	**0.4724**	**1.4414**
			64	0.3811	1.6165	0.4605	1.4797
			80	0.3579	1.6848	0.4241	1.5606
			96	0.3896	1.6056	0.4641	1.4702
		0.5	16	0.2652	1.8770	0.3717	1.7143
			32	0.2692	1.8777	0.3726	1.6982
			48	0.2794	1.8521	0.4016	1.6589
			64	**0.3003**	**1.8029**	**0.4297**	**1.6013**
			80	0.2668	1.8981	0.3704	1.7323
			96	0.2911	1.8248	0.3943	1.6565

Table 4.25: Running a simple test showing a varying, given first layer using a Gabor filter bank, testing on the CIFAR-10 dataset. The layers are able to learn during training. In this test, ψ is the only variable that is given a range changing from the intervals given in Table 4.2. The test here is dependent on the number of filters given in the first layer ranging from $[16, 32, 48, 64, 80, 96]$ inclusive. Psi changes 6 times based on the number of filters for each different dropout given.



(a)



(b)

Figure 4.25: Here we see a different representation of Tables 4.24 and 4.25 showing the training and validation accuracy of the GCNN using MNIST and CIFAR-10 for ψ. Each colored bar in the legend shows the training and validation results. The Gabor filter bank has dropout and a fixed number of filters.

Discussion

Following the trend from the previous tests, increasing the number of filters within the first layer appears to increase the performance for the MNIST dataset as well as CIFAR-10. Like θ, we can see that for both datasets, the results show that there is no noticeable pattern. The

performance for both datasets resemble the θ test previously as well where the results obtained on average are higher than the other parameters' results. This could be based on shifting the filter and seeing features that blurring (σ) or width of the band (λ) cannot extract.

Adding dropout to both datasets results in a decreased performance following the trend from previous tests. This can be seen for both training and validation. We can conclude by saying that training with more filters on both datasets, which changes the interval rate of ψ, offers higher results. We can also conclude that when we adjust ψ, the results appear to be higher than average.

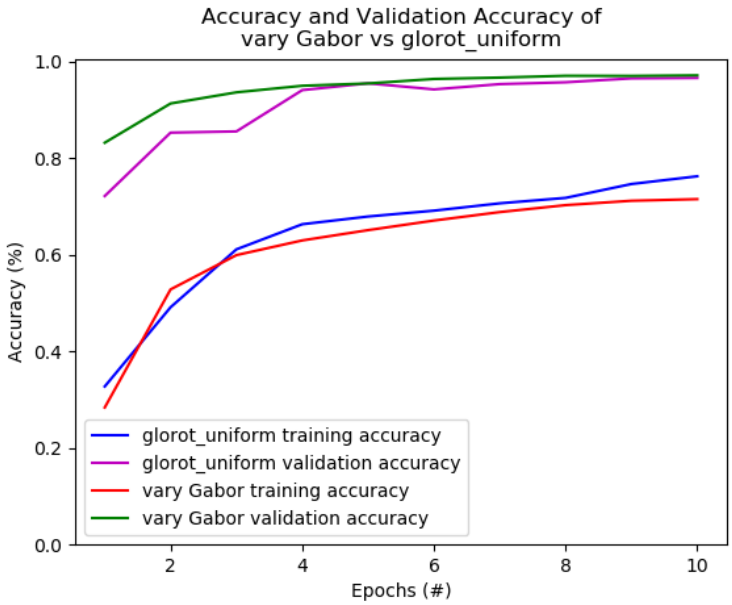


Figure 4.26: A test showing the training results per epoch for ψ when the number of filters, $F = 96$ and dropout, $D = 0.5$, when using the Gabor filter as initialization in the first layer. In comparison we can see how a Xavier initialization competes. In this experiment for using a grid search for one parameter, MNIST is the dataset being used to train on. The legend item that says 'vary Gabor' signifies using ψ as the parameter being tested on.

4.2.3 Summary

In summary for this experiment, by varying one parameter at a time and changing the number of filters (which also changes the intervals for each parameter), we can conclude in saying that σ, λ, and γ offer similar impact to the Gabor filter. We have shown that the results in MNIST

for these parameters are close to each other by a small percentage which tells us that we can use less parameterization here. It especially shows when we compare with using CIFAR-10 as the dataset instead of the MNIST dataset, where results drop linearly with more filters added.

The other two parameters, θ and ψ, offer different results where the results from testing on both datasets yield higher results than σ, λ, and γ. For MNIST, the results are very close, but for CIFAR-10, there is no noticeable pattern. It also appears that increasing the number of filters increases the results for both datasets but this is unknown until there is more data. For now, we can say there is no added benefit for the number of filters used for θ and ψ.

In conclusion, this experiment shows that the number of filters or variations used per parameter affect the results. In particular, for simple datasets, like MNIST, using less filters yields lower results. Variations of θ and ψ offer higher results overall and σ, λ, and γ are offer less impact for overall results. Using less filters for σ, λ, and γ is more beneficial if complexity is a problem. There is no seen benefits of adding or reducing filters for θ and ψ.

Parameter	**Impact Overall**	**Filters**
Sigma	Low	Less
Lambda	Low	Less
Theta	High	No Effect
Gamma	Low	Less
Psi	High	No Effect

Table 4.26: A summary of the impact of each parameter in the Gabor filter when used in the CNN. σ and ψ offer a higher impact overall, and the other 3 variables offer the same impact. When using less filters, σ, λ, and γ offers similar results dropping a small performance percentage of $< 1\%$ when using, for example 16 filters versus 96 filters. σ and ψ offer no benefit of using more or less filters.

4.3 C: Grid Search Vs. Random Search with Shuffled Datasets Training on MNIST, CIFAR-10, and CIFAR-100

4.3.1 Experiment Setup

For the third experiment, we will be using the Gabor filter bank described in [16]. In comparison we will use our randomly initialized Gabor filter bank described in Section 3.3. The values for the parameters are listed in Table 4.2 which will be used on both the randomization test and the grid-search test. Please note that for the remainder of this section, the term uniform is used to describe the grid-search that Ozbulak has created in [16]. Other hyperparameters are used as discussed in Section 3.3. Below, we will see how the 2 different initialization methods work with 3 different datasets that are shuffled for each trial. There are a total of 9 tests, each with 10 trials.

For each test, we change the number of filters. This changes the number of variations of the Gabor filter bank. The Gabor filter bank is created based on the number of filters generated in layer 1 of the CNN. This parameter will be tuned in this experiment and is used in both the uniform and random tests.

Each test will have 10 trials, and 10 or 100 epochs. Due to the simplicity of MNIST, we will use 10 epochs because it converges to a 97% accuracy at about epoch 3. As for the other 2 datasets, CIFAR-10 and CIFAR-100, we will use 100 epochs since they are more complex and require more training which we found in the previous 2 experiments. The averages of these trials will be taken in to compare whether using randomization or uniformity offers better results. The datasets will be shuffled to test the randomization of tests. In the next experiment we will not shuffle the datasets to see whether randomness can outweigh the brute force-like method, grid search. Once the results have been collected for this experiment, we will take the average of the 10 trials. There are 3 different dropouts, $D = [0, 0.25, 0.5]$ and 6 different filter variations, $F = [32, 64, 96]$ for a total of 18 tests (1 test has 10 trials of either 10 or 100

epochs). An example of a complete test set will be shown in the following tables below.

MNIST										
Dropout	Filter Size	Epoch	Random				Uniform			
			Training		Validation		Training		Validation	
			Acc	Loss	Acc	Loss	Acc	Loss	Acc	Loss
0	32	1	0.8367	0.4811	0.9799	0.0693	0.7985	0.5827	0.9719	0.0920
		2	0.9757	0.0800	0.9818	0.0578	0.9721	0.0927	0.9823	0.0581
		3	0.9838	0.0545	0.9881	0.0362	0.9807	0.0621	0.9858	0.0456
		4	0.9871	0.0431	0.9905	0.0294	0.9850	0.0481	0.9900	0.0330
		5	0.9893	0.0362	0.9898	0.0303	0.9878	0.0398	0.9874	0.0361
		6	0.9903	0.0321	0.9896	0.0313	0.9889	0.0351	0.9906	0.0288
		7	0.9916	0.0286	0.9914	0.0252	0.9907	0.0308	0.9908	0.0275
		8	0.9925	0.0254	0.9916	0.0231	0.9919	0.0276	0.9901	0.0283
		9	0.9931	0.0239	0.9923	0.0232	0.9923	0.0259	0.9917	0.0274
		10	0.9933	0.0225	0.9922	0.0226	0.9924	0.0239	0.9904	0.0258

Table 4.27: A table representing one trial run showing the performance gains from a total of 10 epochs using the MNIST dataset. The trial is initialized with 32 Gabor filters in the first layer and shows how random initializations fairs against a grid-search initialization.

MNIST										
Dropout	# of Filters	Trial	Random				Uniform			
			Training		Validation		Training		Validation	
			Acc	Loss	Acc	Loss	Acc	Loss	Acc	Loss
0	32	1	0.9933	0.0225	0.9922	0.0226	0.9924	0.0239	0.9904	0.0258
		2	0.9939	0.0201	0.9922	0.0227	0.9926	0.0244	0.9906	0.0265
		3	0.9928	0.0236	0.9932	0.0230	0.9925	0.0245	0.9909	0.0261
		4	0.9931	0.0222	0.9933	0.0210	0.9916	0.0267	0.9908	0.0297
		5	0.9929	0.0230	0.9921	0.0247	0.9927	0.0235	0.9907	0.0266
		6	0.9933	0.0216	0.9930	0.0220	0.9921	0.0257	0.9913	0.0271
		7	0.9930	0.0229	0.9923	0.0213	0.9918	0.0271	0.9917	0.0268
		8	0.9936	0.0211	0.9919	0.0240	0.9924	0.0251	0.9905	0.0290
		9	0.9928	0.0226	0.9898	0.0297	0.9931	0.0224	0.9916	0.0241
		10	0.9934	0.0207	0.9921	0.0239	0.9937	0.0223	0.9907	0.0269
	Average		0.9932	0.0220	0.9922	0.0235	0.9920	0.0246	0.9910	0.0269
	Standard Deviation		3.55E-04	1.12E-03	9.85E-04	2.47E-03	6.10E-04	1.63E-03	4.57E-04	1.59E-03
	Variance		1.26E-07	1.24E-06	9.70E-07	6.09E-06	3.72E-07	2.65E-06	2.08E-07	2.53E-06

Table 4.28: Results showing a test run with 10 trials for when we use uniform and random initialization of the Gabor filter. The ranges of the Gabor parameter are given in Table 4.2. The dataset, MNIST, dropout, $D = 0$, and the number of filters, $F = 32$, are the parameters we are testing. The other parameters are described in Section 3.3. The dataset is also shuffled for each test. Here we list the accuracy and lossfor both training and validation results. Please note that the term uniform is used in place of the grid-search that Ozbulak had created in [16].

MNIST									
Dropout	# of Filters	Trial	Random				Uniform			
			Training		Validation		Training		Validation	
			Acc	Loss	Acc	Loss	Acc	Loss	Acc	Loss
0.25	64	1	0.9938	0.0200	0.9916	0.0266	0.9934	0.0218	0.9922	0.0254
		2	0.9930	0.0220	0.9919	0.0225	0.9940	0.0205	0.9917	0.0224
		3	0.9936	0.0216	0.9915	0.0226	0.9936	0.0215	0.9924	0.0249
		4	0.9940	0.0203	0.9924	0.0219	0.9934	0.0215	0.9938	0.0211
		5	0.9933	0.0219	0.9922	0.0234	0.9928	0.0235	0.9908	0.0273
		6	0.9932	0.0216	0.9893	0.0317	0.9928	0.0235	0.9870	0.0408
		7	0.9918	0.0254	0.9928	0.0229	0.9934	0.0202	0.9910	0.0279
		8	0.9940	0.0196	0.9922	0.0219	0.9933	0.0222	0.9917	0.0241
		9	0.9944	0.0183	0.9926	0.0224	0.9937	0.0211	0.9934	0.0197
		10	0.9932	0.0217	0.9912	0.0254	0.9931	0.0226	0.9923	0.0236
	Average		0.9930	0.02120	0.9920	0.0241	0.9930	0.0218	0.9920	0.0257
	Standard Deviation		7.27E-04	1.90E-03	1.00E-03	3.06E-03	3.71E-04	1.13E-03	1.88E-03	5.90E-03
	Variance		5.28E-07	3.59E-06	1.01E-06	9.39E-06	1.38E-07	1.28E-06	3.53E-06	3.48E-05

Table 4.29: Results showing a test run with 10 trials for when we use uniform and random initialization of the Gabor filter. The ranges of the Gabor parameter are given in Table 4.2. The dataset, MNIST, dropout, $D = 0.25$, and the number of filters, $F = 64$, are the parameters we are testing. The other parameters are described in Section 3.3. The dataset is also shuffled for each test. Here we list the accuracy and lossfor both training and validation results. Please note that the term uniform is used in place of the grid-search that Ozbulak had created in [16].

MNIST									
Dropout	# of Filters	Trial	Random				Uniform			
			Training		Validation		Training		Validation	
			Acc	Loss	Acc	Loss	Acc	Loss	Acc	Loss
0.5	96	1	0.9939	0.0209	0.9930	0.0222	0.9931	0.0228	0.9915	0.0246
		2	0.9944	0.0186	0.9929	0.0221	0.9938	0.0197	0.9919	0.0227
		3	0.9943	0.0194	0.9931	0.0211	0.9936	0.0210	0.9933	0.0204
		4	0.9935	0.0226	0.9924	0.0238	0.9937	0.0206	0.9911	0.0264
		5	0.9938	0.0198	0.9926	0.0212	0.9938	0.0210	0.9929	0.0215
		6	0.9934	0.0217	0.9912	0.0270	0.9936	0.0213	0.9920	0.0274
		7	0.9933	0.0221	0.9903	0.0270	0.9934	0.0214	0.9911	0.0264
		8	0.9924	0.0249	0.9913	0.0243	0.9938	0.0218	0.9927	0.0253
		9	0.9943	0.0192	0.9925	0.0235	0.9938	0.0208	0.9930	0.0224
		10	0.9938	0.0207	0.9933	0.0225	0.9936	0.0201	0.9928	0.0216
	Average		0.9940	0.0210	0.9920	0.0235	0.9940	0.0210	0.9920	0.0239
	Standard Deviation		6.01E-04	1.90E-03	9.90E-04	2.14E-03	2.38E-04	8.77E-04	8.15E-04	2.46E-03
	Variance		3.61E-07	3.63E-06	9.80E-07	4.58E-06	5.65E-08	7.69E-07	6.65E-07	6.06E-06

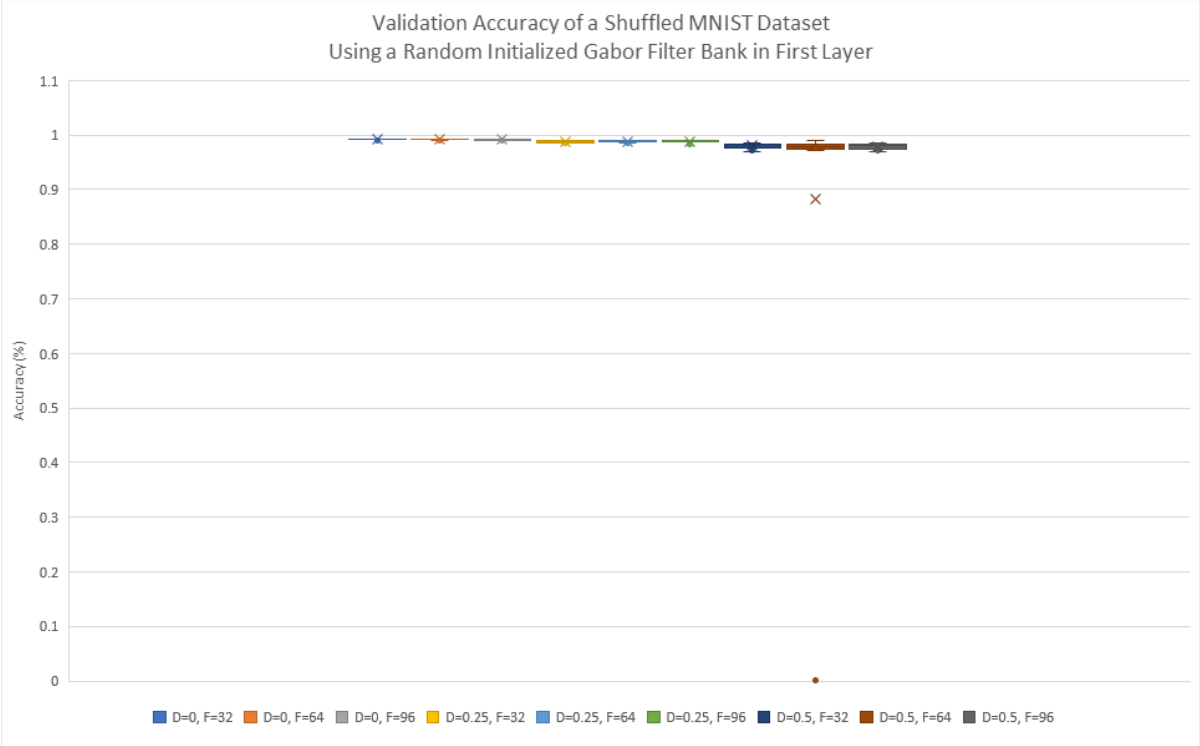
Table 4.30: Results showing a test run with 10 trials for when we use uniform and random initialization of the Gabor filter. The ranges of the Gabor parameter are given in Table 4.2. The dataset, MNIST, dropout, $D = 0.5$, and the number of filters, $F = 96$, are the parameters we are testing. The other parameters are described in Section 3.3. The dataset is also shuffled for each test. Here we list the accuracy and lossfor both training and validation results. Please note that the term uniform is used in place of the grid-search that Ozbulak had created in [16].

4.3.2 Results

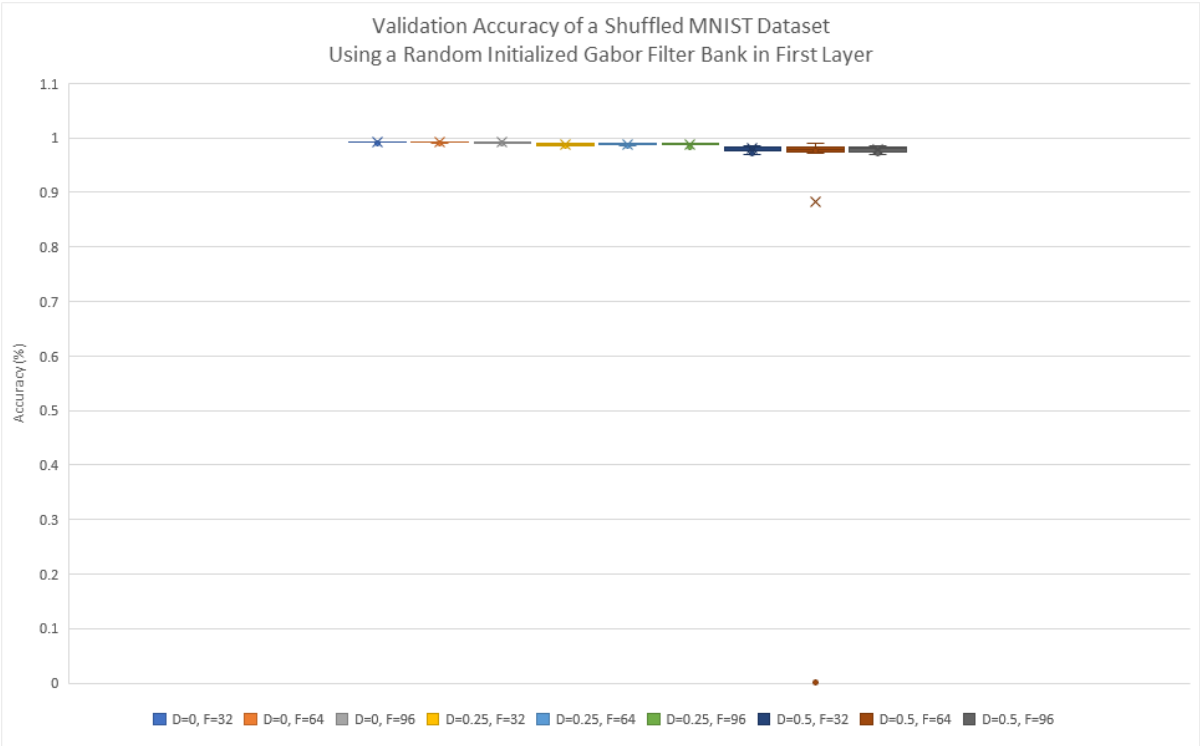
MNIST

MNIST														
Dropout	Test Type	# of Filters	Training						Validation					
			Acc	Loss	Std. Dev Acc	Std. Dev Loss	Var Acc	Var Loss	Acc	Loss	Std. Dev Acc	Std. Dev Loss	Var Acc	Var Loss
0	Random	32	0.9932	0.0220	3.55E-04	1.12E-03	1.26E-07	1.24E-06	0.9922	0.0235	9.85E-04	2.47E-03	9.70E-07	6.09E-06
		64	0.9934	0.0212	7.27E-04	1.90E-03	5.28E-07	3.59E-06	0.9918	0.0241	1.00E-03	3.06E-03	1.01E-06	9.39E-06
		96	0.9937	0.0210	6.01E-04	1.90E-03	3.61E-07	3.63E-06	0.9923	0.0235	9.90E-04	2.14E-03	9.80E-07	4.58E-06
	Uniform	32	0.9925	0.0246	6.10E-04	1.63E-03	3.72E-07	2.65E-06	0.9909	0.0269	4.57E-04	1.59E-03	2.08E-07	2.53E-06
		64	0.9933	0.0218	3.71E-04	1.13E-03	1.38E-07	1.28E-06	0.9916	0.0257	1.88E-03	5.90E-03	3.53E-06	3.48E-05
		96	0.9936	0.0210	2.38E-04	8.77E-04	5.65E-08	7.69E-07	0.9922	0.0239	8.15E-04	2.46E-03	6.65E-07	6.06E-06
0.25	Random	32	0.9342	0.1977	1.08E-02	2.34E-02	1.16E-04	5.47E-04	0.9880	0.0471	1.77E-03	8.01E-03	3.12E-06	6.41E-05
		64	0.9414	0.1838	8.64E-03	2.30E-02	7.47E-05	5.29E-04	0.9881	0.0486	1.64E-03	7.83E-03	2.70E-06	6.13E-05
		96	0.9339	0.1939	6.61E-03	2.17E-02	4.36E-05	4.71E-04	0.9883	0.0461	2.42E-03	1.01E-02	5.83E-06	1.03E-04
	Uniform	32	0.9432	0.1775	6.47E-03	1.85E-02	4.19E-05	3.41E-04	0.9874	0.0472	2.03E-03	8.17E-03	4.12E-06	6.68E-05
		64	0.9406	0.1943	8.52E-03	3.79E-02	7.25E-05	1.43E-03	0.9872	0.0525	1.95E-03	1.40E-02	3.80E-06	1.96E-04
		96	0.9382	0.1898	1.21E-02	3.44E-02	1.48E-04	1.19E-03	0.9887	0.0461	1.83E-03	1.12E-02	3.35E-06	1.25E-04
0.5	Random	32	0.7867	0.6206	1.51E-02	4.37E-02	2.29E-04	1.91E-03	0.9804	0.1283	5.94E-03	4.93E-02	3.53E-05	2.43E-03
		64	0.7966	0.6156	2.65E-02	5.76E-02	7.05E-04	3.32E-03	0.9800	0.1322	5.32E-03	3.98E-02	2.82E-05	1.59E-03
		96	0.7838	0.6436	2.87E-02	9.19E-02	8.25E-04	8.45E-03	0.9798	0.1487	5.86E-03	7.14E-02	3.44E-05	5.10E-03
	Uniform	32	0.7851	0.6355	4.27E-02	1.01E-01	1.83E-03	1.02E-02	0.9801	0.1406	3.94E-03	4.99E-02	1.55E-05	2.49E-03
		64	0.7935	0.5996	1.76E-02	4.90E-02	3.11E-04	2.41E-03	0.9826	0.1079	2.31E-03	2.81E-02	5.33E-06	7.89E-04
		96	0.7544	0.6934	7.31E-02	2.05E-01	5.34E-03	4.20E-02	0.9595	0.2000	4.70E-02	1.84E-01	2.21E-03	3.37E-02

Table 4.31: The following results are the averages of 10 trials for each test using MNIST as the dataset. Each test we run have a different dropout, $D = [0, 0.25, 0.5]$, and the number of filters, $F = [32, 64, 96]$. Each row provides the accuracy, loss, standard deviation, and variance for both training and validation results. Note: Uniform is the term we are using to describe Ozbulak's results in [16]. Here we have a shuffled dataset for each epoch for all trials.



(a)



(b)

Figure 4.27: Here we see a different representation of Table 4.31 showing the mean, maximum and minimum values as well as the mean of the validation accuracy. Within the legends are each permutation of the experiment using the shuffled MNIST dataset in both training and validation sets.

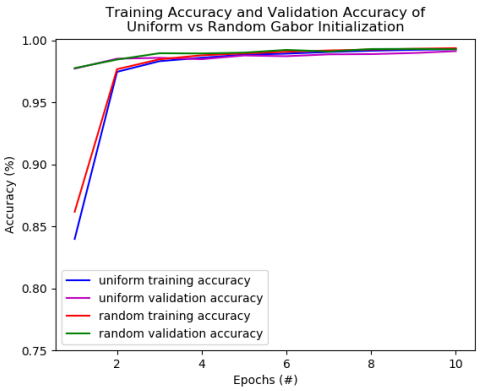
Discussion

When we add dropout to the MNIST dataset, we can see a drop in training and validation results. Although it is not a significant drop, there is still a noticeable pattern that when we add dropout, results tend to drop. This was seen and discussed in the 2 previous experiments as well. We can also see that the loss increases on average as we increase the dropout.

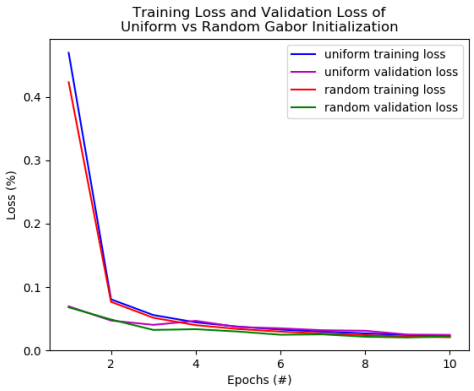
When we use different numbers of filters in layer 1, we can see a pattern where if we increase the filters, we can see an improvement for results. This pattern is shown when dropout, $D = [0, 0.25]$, but when $D = 0.5$, we see the opposite trend where the performance drops. This can be explained with using the standard deviation and the variance though. The test where $D = 0.5$ and the number of filters, $F = 96$ shows an increase for the standard deviation and variance which signifies that there is more variance for the results. Assuming this value is unreliable compared to it's neighboring tests, we can assume the value is not as accurate as it could be. The trend would continue where if we increase the filter numbers, we would increase the performance. This happens in both uniform and random tests. That said, the difference between using 32 and 96 filters is small. The benefits of using a smaller filter size here would benefit the network more than using a larger number of filters.

Out of the 9 different tests comparing a uniform initialization method versus a random initialization method, MNIST appears to favor random initialization more than a uniform distribution. To add to this point, using a smaller filter size favours random initialization. The 2 tests that beat the random distribution had 96 filters as their first layer. The other 7 tests were all favoured for random initialization.

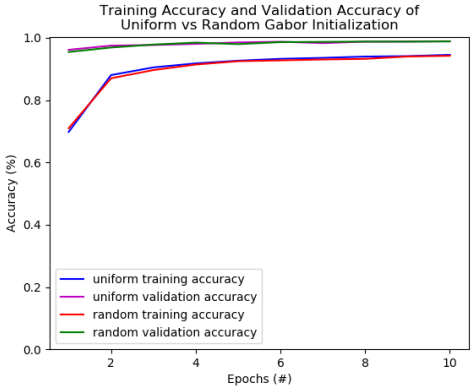
In summary, adding dropout decreases the performance. MNIST favours random initialization more than a grid-search like initialization. Using more filters provides a more accurate result overall, but with a small gain over using a smaller number of filters. One can accomplish the same goals if they were to use a smaller filter size all while reducing the complexity and the number of parameters within the CNN.



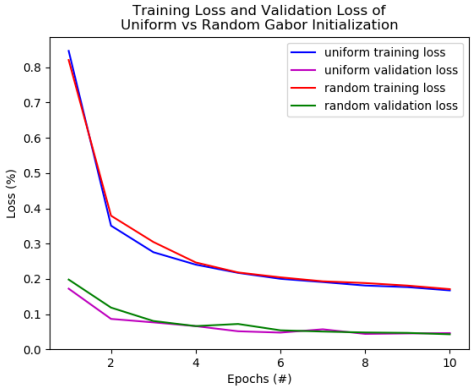
(a) Accuracy: $D = 0$, $F = 96$



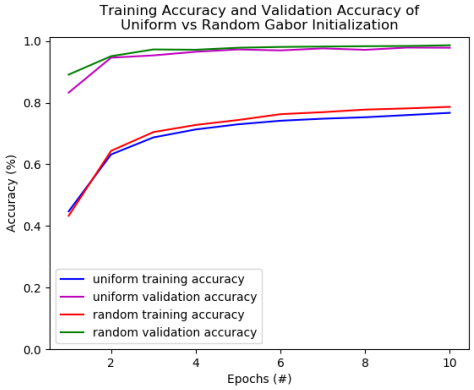
(b) Loss: $D = 0$, $F = 96$



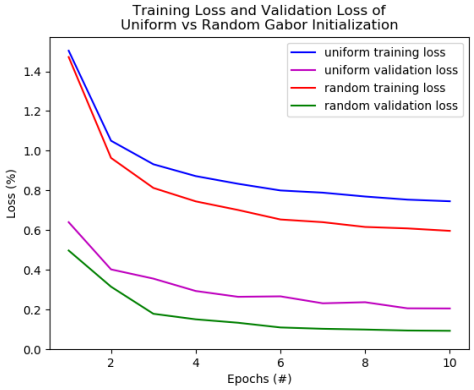
(c) Accuracy: $D = 0.25$, $F = 96$



(d) Loss: $D = 0.25$, $F = 96$



(e) Accuracy: $D = 0.50$, $F = 96$



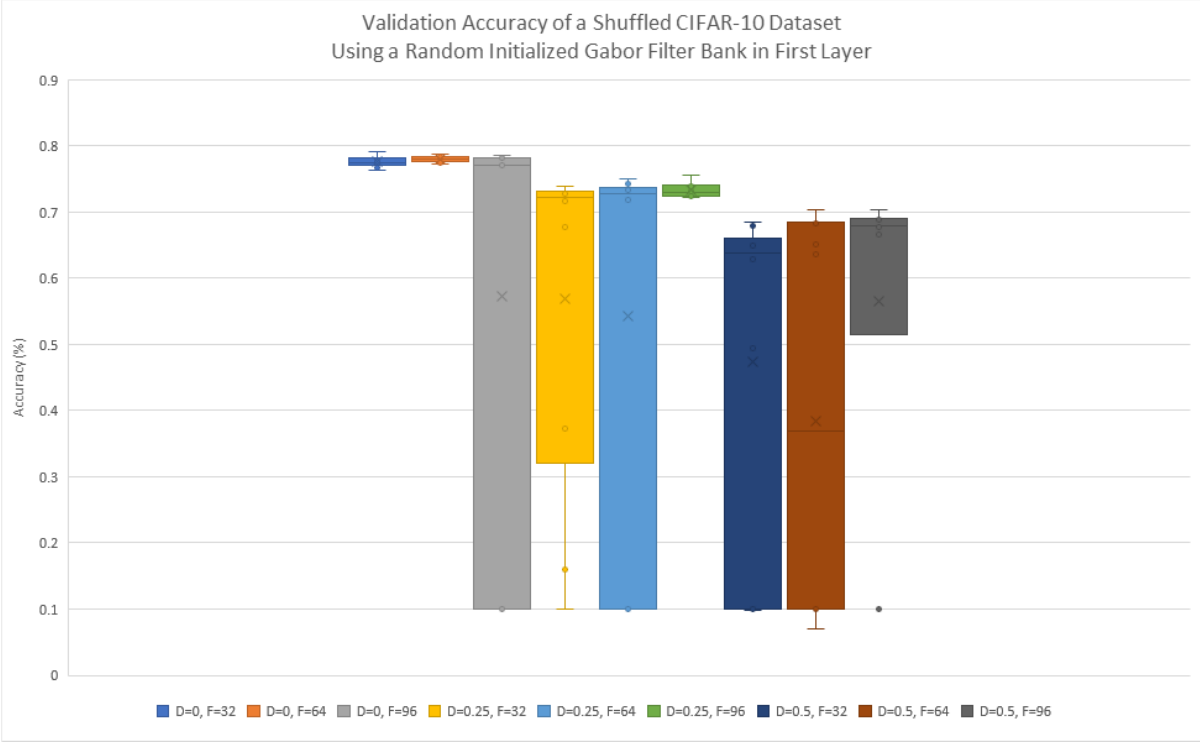
(f) Loss: $D = 0.50$, $F = 96$

Figure 4.28: Training on MNIST shows a drop in performance when we add dropout to the network. The performance finds its peak very fast, likely due to the simplicity of the dataset. We can see a performance drop when we switch to a dataset with colour such as CIFAR-10. This is likely do to the extra overhead of dealing with colour in the network. To see an example, refer to 4.30. Using random or grid-search initialization for the Gabor filters shows similar training results and testing results.

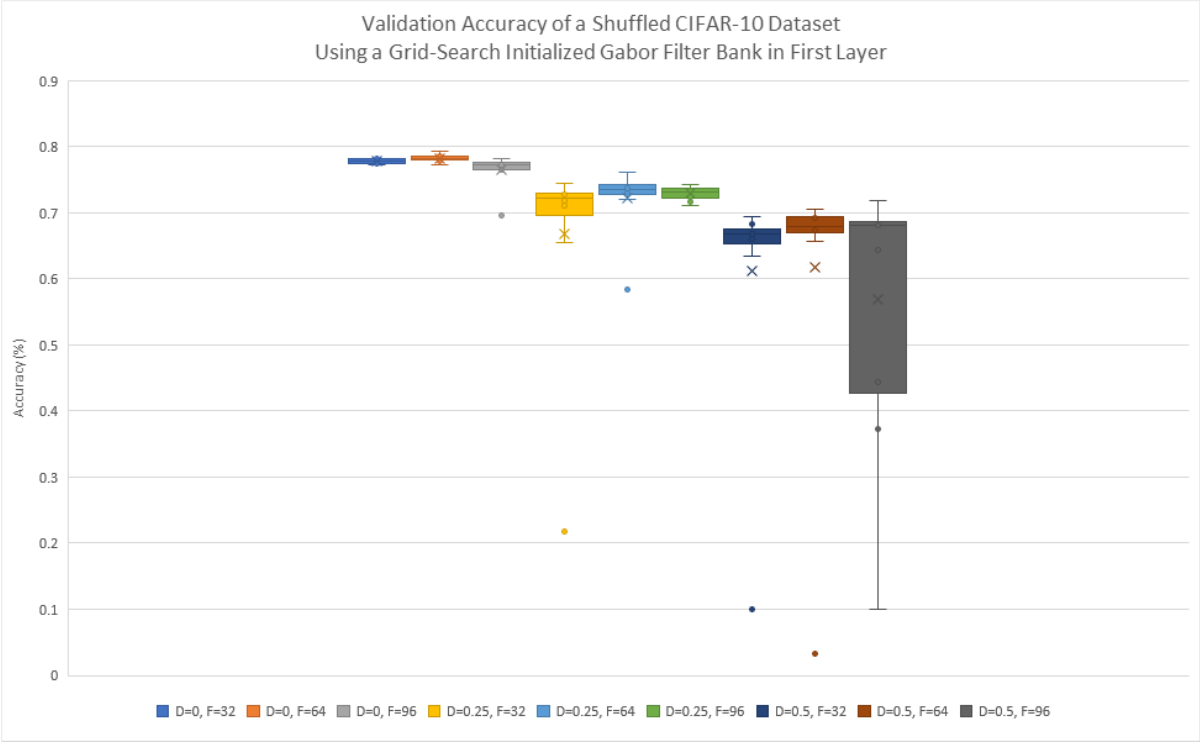
CIFAR-10

CIFAR-10															
Dropout	**Test Type**	**# of Filters**	**Training**						**Validation**						
			Acc	**Loss**	**Std. Dev Acc**	**Std. Dev Loss**	**Var Acc**	**Var Loss**	**Acc**	**Loss**	**Std. Dev Acc**	**Std. Dev Loss**	**Var Acc**	**Var Loss**
0	Random	32	1.0000	0.0021	9.66E-06	3.52E-04	9.33E-11	1.24E-07	0.7758	1.6800	8.21E-03	1.11E-01	6.74E-05	1.24E-02
		64	1.0000	0.0020	1.03E-05	7.75E-04	1.07E-10	6.01E-07	0.7805	1.6166	5.05E-03	7.14E-02	2.55E-05	5.10E-03
		96	0.7292	0.6920	4.36E-01	1.11E+00	1.90E-01	1.24E+00	0.5736	1.8701	3.27E-01	3.02E-01	1.07E-01	9.14E-02
	Uniform	32	1.0000	0.0021	1.05E-05	2.33E-04	1.11E-10	5.44E-08	0.7782	1.6554	3.67E-03	4.42E-02	1.35E-05	1.95E-03
		64	1.0000	0.0020	8.43E-06	3.15E-04	7.11E-11	9.91E-08	0.7829	1.6095	5.65E-03	5.31E-02	3.20E-05	2.82E-03
		96	0.9852	0.0425	4.68E-02	1.28E-01	2.19E-03	1.64E-02	0.7644	1.6372	2.49E-02	2.30E-01	6.20E-04	5.27E-02
0.25	Random	32	0.6458	0.9419	3.28E-01	8.29E-01	1.07E-01	6.87E-01	0.5686	1.3385	2.56E-01	5.39E-01	6.57E-02	2.91E-01
		64	0.6321	0.9654	3.70E-01	9.24E-01	1.37E-01	8.54E-01	0.5436	1.4020	3.06E-01	6.26E-01	9.38E-02	3.91E-01
		96	0.8774	0.3639	2.81E-02	7.38E-02	7.91E-04	5.45E-03	0.7334	1.0769	1.11E-02	9.48E-02	1.23E-04	8.99E-03
	Uniform	32	0.7696	0.6280	2.21E-01	5.68E-01	4.88E-02	3.23E-01	0.6683	1.1372	1.60E-01	3.38E-01	2.56E-02	1.15E-01
		64	0.8309	0.4752	1.23E-01	3.22E-01	1.52E-02	1.04E-01	0.7221	1.0346	4.98E-02	9.94E-02	2.48E-03	9.89E-03
		96	0.8596	0.4005	3.70E-02	9.25E-02	1.37E-03	8.56E-03	0.7295	1.0285	9.80E-03	6.67E-02	9.60E-05	4.45E-03
0.5	Random	32	0.4254	1.4979	2.37E-01	5.82E-01	5.63E-02	3.39E-01	0.4738	1.4871	2.64E-01	5.77E-01	6.96E-02	3.33E-01
		64	0.4067	1.5416	2.76E-01	6.77E-01	7.60E-02	4.59E-01	0.4365	1.5611	2.91E-01	6.41E-01	8.47E-02	4.11E-01
		96	0.5654	1.1586	2.49E-01	6.05E-01	6.20E-02	3.66E-01	0.5648	1.3284	2.45E-01	5.19E-01	6.02E-02	2.69E-01
	Uniform	32	0.5588	1.1567	1.73E-01	4.22E-01	2.98E-02	1.78E-01	0.6112	1.1801	1.80E-01	3.96E-01	3.25E-02	1.57E-01
		64	0.6507	0.9526	3.44E-02	7.11E-02	1.18E-03	5.06E-03	0.6837	1.0256	1.36E-02	6.02E-02	1.86E-04	3.63E-03
		96	0.4956	1.3215	2.00E-01	4.85E-01	4.00E-02	2.35E-01	0.5700	1.2466	2.02E-01	4.58E-01	4.09E-02	2.09E-01

Table 4.32: The following results are the averages of 10 trials for each test using CIFAR-10 as the dataset. Each test we run have a different dropout, $D = [0, 0.25, 0.5]$, and the number of filters, $F = [32, 64, 96]$. Each row provides the accuracy, loss, standard deviation, and variance for both training and validation results. Note: Uniform is the term we are using to describe Ozbulak's results in [16]. Here we have a shuffled dataset for each epoch for all trials.



(a)



(b)

Figure 4.29: Here we see a different representation of Table 4.32 showing the mean, maximum and minimum values as well as the mean of the validation accuracy. Within the legends are each permutation of the experiment using the shuffled CIFAR-10 dataset in both training and validation sets. Longer bars just shows the range in values are more spread apart. In some cases of shuffling the dataset, training does not even start up and is never able to converge.

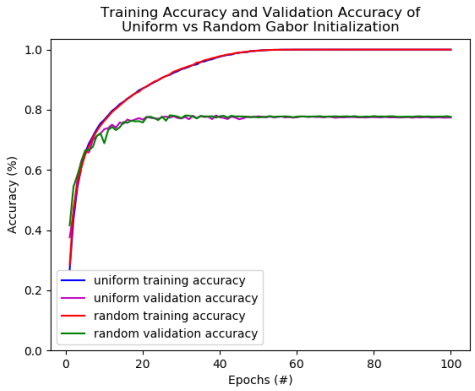
Discussion

From out results, we can see that the network is overfitting on the CIFAR-10 dataset. The overfitting starts to happen around the 20th epoch and as we add dropout, the point at which the CNN overfits increases as described in the figure below. There is no benefit of adding dropout here in this case because we find that by adding dropout, our performance drops in both training and validation.

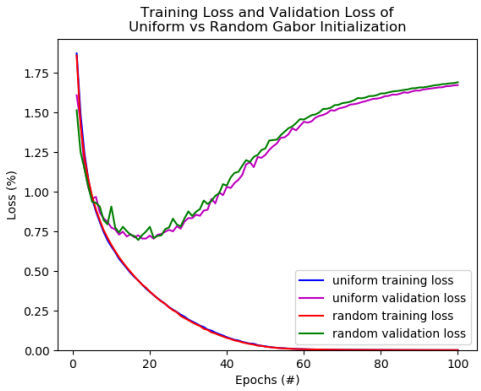
Increasing the number of filters appears to improve the performance for the network for both uniform and random initialization methods. The results shown in Table 4.32 shows that as we increase the filters from 32 to 96, there is a small improvement for accuracy and loss. The pattern appears to break on certain tests but the standard deviation and variance are larger than normal. This could mean that the result is not as accurate as the pattern suggests.

When training on CIFAR-10, the dataset leans towards using a grid-search initialization over a random initialization. Out of 9 tests comparing the different initialization methods, 8 of the 9 tests favours grid-search. Also, the overall performance of the network is increased and produces a higher accuracy when using grid-search compared to randomness.

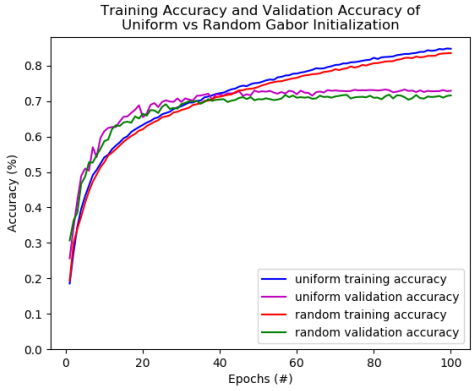
In summary, when testing with CIFAR-10, the dataset favours using a grid-search for initialization instead of random initialization. Not only does it favour grid-search as the initialization method, it produces higher results and performs better. Dropout is not needed when training on this network because the dataset overfits for all tested scenarios. If dropout is included, the results drop as well. Using more filters improves the performance of the network with a small gain.



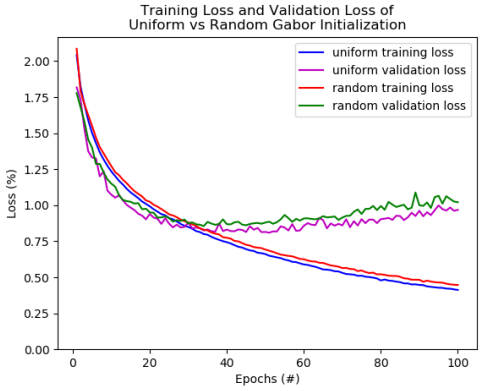
(a) Accuracy: $D = 0$, $F = 32$



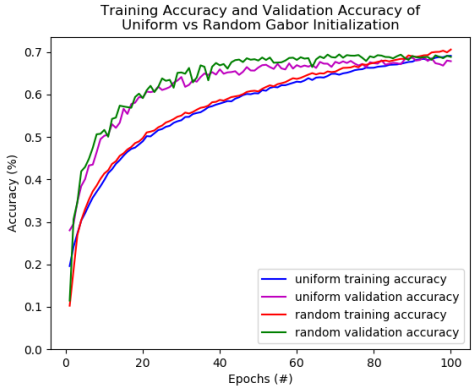
(b) Loss: $D = 0$, $F = 32$



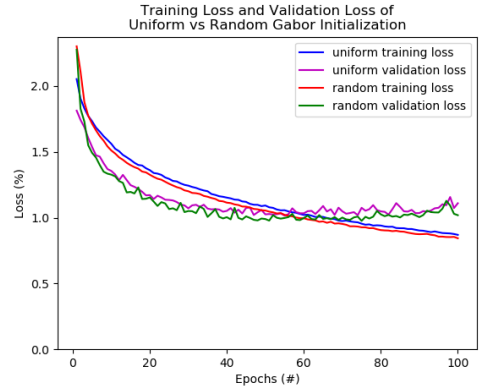
(c) Accuracy: $D = 0.25$, $F = 32$



(d) Loss: $D = 0.25$, $F = 32$



(e) Accuracy: $D = 0.50$, $F = 64$



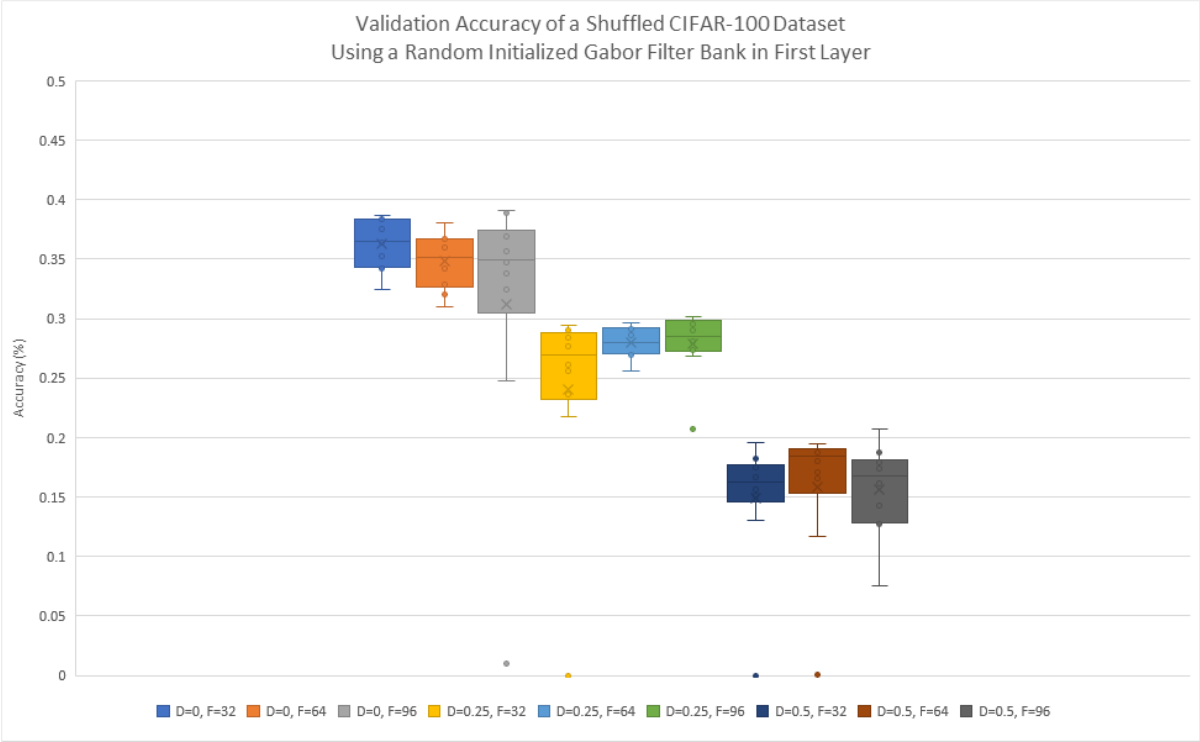
(f) Loss: $D = 0.50$, $F = 64$

Figure 4.30: Training on the CIFAR-10 dataset results in overfitting even with dropout using our network described in Section 3.3. As we add dropout to both random and uniform initialization methods, we increase the epoch at which we reach the point where the CNN overfits. When dropout, $D = 0$, we can see that around epoch, $E = 18$ the network starts to overfit. When $D = 0.25$, we see that the network overfits around $E = 30$, and finally when $D = 0.50$, the epoch at which it overfits is about $E = 35$. The performance of the network surprisingly drops with an increased dropout. Using random or grid-search initialization for the Gabor filters shows similar training results and testing results.

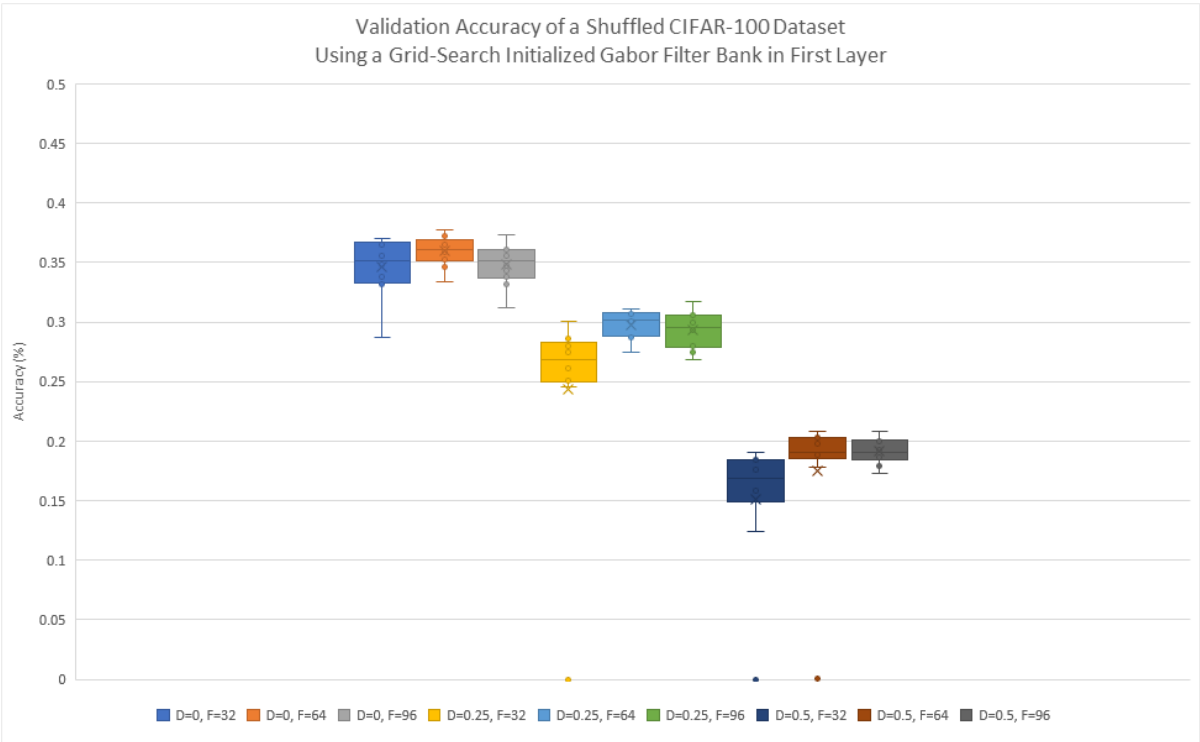
CIFAR-100

CIFAR-100														
Dropout	**Test Type**	**# of Filters**	**Training**						**Validation**					
			Acc	**Loss**	**Std. Dev Acc**	**Std. Dev Loss**	**Var Acc**	**Var Loss**	**Acc**	**Loss**	**Std. Dev Acc**	**Std. Dev Loss**	**Var Acc**	**Var Loss**
0	Random	32	0.6315	1.2221	1.03E-01	4.05E-01	1.06E-02	1.64E-01	0.3625	3.0255	2.13E-02	4.99E-01	4.53E-04	2.49E-01
		64	0.7852	0.6660	5.06E-02	1.57E-01	2.56E-03	2.45E-02	0.3481	3.9916	2.31E-02	5.94E-01	5.33E-04	3.53E-01
		96	0.6407	1.3513	2.91E-01	1.37E+00	8.49E-02	1.88E+00	0.3123	3.8403	1.14E-01	8.34E-01	1.29E-02	6.96E-01
	Uniform	32	0.7485	0.7814	4.25E-02	1.33E-01	1.80E-03	1.78E-02	0.3464	3.7686	2.57E-02	3.49E-01	6.63E-04	1.22E-01
		64	0.8104	0.5940	2.19E-02	6.74E-02	4.82E-04	4.55E-03	0.3596	3.9668	1.28E-02	2.06E-01	1.64E-04	4.26E-02
		96	0.8028	0.6128	3.23E-02	9.88E-02	1.04E-03	9.76E-03	0.3485	4.0598	1.79E-02	3.97E-01	3.22E-04	1.57E-01
0.25	Random	32	0.1921	3.1185	2.36E-02	1.62E-01	5.58E-04	2.62E-02	0.2669	2.9259	2.50E-02	1.40E-01	6.26E-04	1.96E-02
		64	0.2089	2.9971	1.54E-02	1.09E-01	2.36E-04	1.20E-02	0.2799	2.8390	1.34E-02	6.91E-02	1.80E-04	4.78E-03
		96	0.2059	3.0114	3.09E-02	2.11E-01	9.54E-04	4.45E-02	0.2789	2.8465	2.79E-02	1.57E-01	7.81E-04	2.46E-02
	Uniform	32	0.1953	3.0779	1.95E-02	9.21E-02	3.81E-04	8.48E-03	0.2725	2.8888	1.91E-02	9.55E-02	3.64E-04	9.12E-03
		64	0.2275	2.8772	1.21E-02	6.62E-02	1.46E-04	4.38E-03	0.2979	2.7567	1.22E-02	4.37E-02	1.50E-04	1.91E-03
		96	0.2251	2.8946	1.50E-02	9.44E-02	2.26E-04	8.91E-03	0.2935	2.7709	1.53E-02	5.73E-02	2.33E-04	3.28E-03
0.5	Random	32	0.0841	3.7936	9.91E-03	1.00E-01	9.82E-05	1.01E-02	0.1654	3.4995	1.79E-02	1.24E-01	3.20E-04	1.54E-02
		64	0.0908	3.7511	1.47E-02	1.24E-01	2.15E-04	1.53E-02	0.1738	3.4591	2.43E-02	1.32E-01	5.89E-04	1.73E-02
		96	0.0800	3.8273	1.70E-02	1.56E-01	2.89E-04	2.43E-02	0.1558	3.5403	3.83E-02	2.13E-01	1.46E-03	4.52E-02
	Uniform	32	0.0891	3.7426	1.15E-02	9.54E-02	1.31E-04	9.10E-03	0.1714	3.4429	2.18E-02	1.15E-01	4.77E-04	1.32E-02
		64	0.0978	3.6493	4.92E-03	4.38E-02	2.43E-05	1.92E-03	0.1948	3.3219	9.29E-03	5.60E-02	8.63E-05	3.13E-03
		96	0.0986	3.6632	9.11E-03	7.36E-02	8.31E-05	5.42E-03	0.1914	3.3450	1.08E-02	7.88E-02	1.17E-04	6.21E-03

Table 4.33: The following results are the averages of 10 trials for each test using CIFAR-10 as the dataset. Each test we run have a different dropout, $D = [0, 0.25, 0.5]$, and the number of filters, $F = [32, 64, 96]$. Each row provides the accuracy, loss, standard deviation, and variance for both training and validation results. Note: Uniform is the term we are using to describe Ozbulak's results in [16]. Here we have a shuffled dataset for each epoch for all trials.



(a)



(b)

Figure 4.31: Here we see a different representation of Table 4.33 showing the mean, maximum and minimum values as well as the mean of the validation accuracy. Within the legends are each permutation of the experiment using the shuffled CIFAR-100 dataset in both training and validation sets.

c

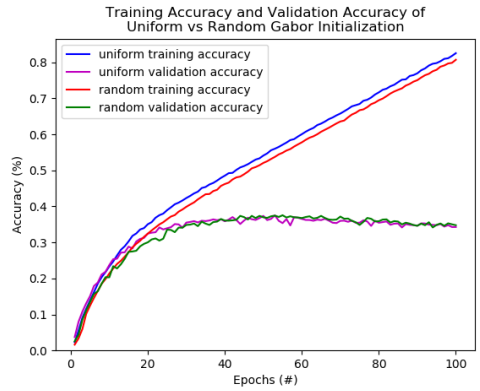
Discussion

When we are training with CIFAR-100, we know that it is more complex adding 90 classes more than CIFAR-10. With 100 classes, the results are decreased substantially. Adding dropout with these tests reduces the results even further. Adding a dropout, $D = 0.50$, is about half as accurate than having no dropout at all. This happens for both initialization methods and continues the trend from the other datasets, MNIST and CIFAR-10.

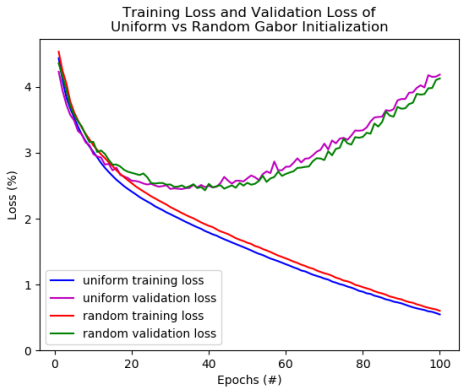
Following the same trend from using the other 2 datasets, for CIFAR-100, increasing the number of filters increases the performance of the CNN. Although it appears that the results on average are less accurate, the loss is increased and the standard deviation and variation is much greater. We can say, that because of this it does follow the trend. Similar to the other datasets as well, the performance for adding more filters offers a small gain. For example, for 32 filters and 25% dropout initialized randomly, we see that there is a 1.5% increase in validation accuracy when using 96 filters with the same dropout.

When using our architecture (described in Section 3.3), we see the same trend when training on the CIFAR-10 dataset where the CNN favours CIFAR-100 if it is initialized using grid-search. In all of the 9 tests run comparing the initialization methods, 8 tests scored higher using a grid search initialization. On average, using grid-search also improves the results compared to using random initialization. For example, the test with dropout, $D = 0.50$, and filters, $F = 64$, the difference between random and grid-search is 2.1% improved validation accuracy.

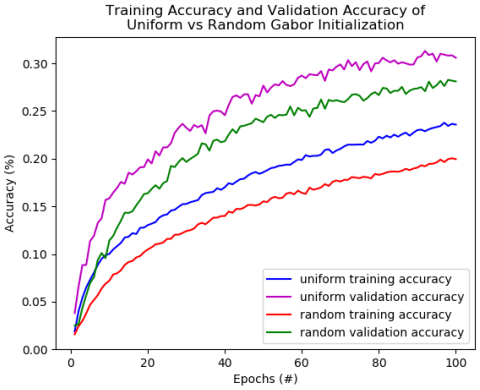
In summary, using dropout reduces the performance of the network when training on CIFAR-100. Using a 50% dropout compared to having no dropout reduces the (validation) accuracy by almost half. Increasing the number of filters in either initialization method provides a higher validation accuracy and lower loss. CIFAR-100 favours a grid search initialization method over randomization. On average, using grid-search initialization also boosts the validation accuracy.



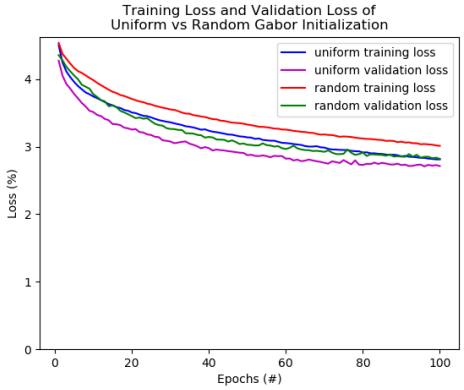
(a) Accuracy: $D = 0$, $F = 96$



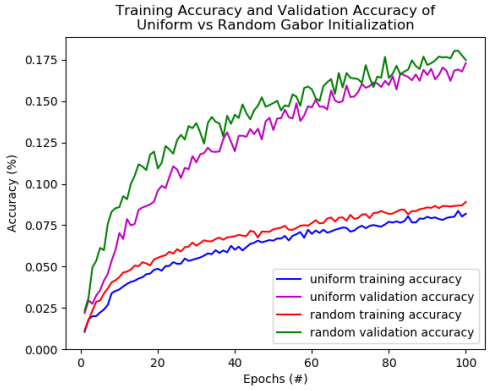
(b) Loss: $D = 0$, $F = 96$



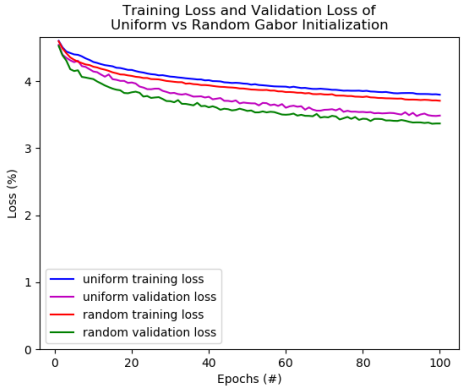
(c) Accuracy: $D = 0.25$, $F = 96$



(d) Loss: $D = 0.25$, $F = 96$



(e) Accuracy: $D = 0.50$, $F = 96$



(f) Loss: $D = 0.50$, $F = 96$

Figure 4.32: Training on the CIFAR-100 dataset shows a drop in performance when we add dropout which shows no benefit in adding dropout. This follows our findings from training with other datasets. We can also see that there is overfitting happening where dropout, $D = 0$ at around epoch, $E = 20$. The performance begins to level off around epoch, $E = 100$ which shows the maximum performance we can obtain based on our given architecture (Section 3.3). Using random or grid-search initialization for the Gabor filters shows similar training results and testing results.

Comparing with Ozbulak's Results

MNIST								
Dropout	**Filter Size**	**Epoch**	**Training**			**Validation**		
			Random	**Uniform**	**Ozbulak**	**Random**	**Uniform**	**Ozbulak**
0	96	1	85.27	84.67	87.36	97.67	97.49	96.60
		2	97.73	97.56	97.56	98.32	98.55	98.26
		3	98.47	98.45	98.35	98.63	98.96	98.69
		4	98.84	98.74	98.70	99.08	98.97	98.74
		5	98.99	98.95	98.89	99.26	99.01	98.70
		6	99.14	99.10	99.03	99.19	99.05	99.23
		7	99.28	99.21	99.15	98.98	99.23	99.22
		8	99.33	99.26	99.21	99.33	99.20	98.97
		9	99.40	99.36	99.31	99.29	99.23	99.25
		10	99.44	99.38	99.33	99.29	99.30	99.25

Table 4.34: A comparison table of the first 10 epochs of training for MNIST. Here we compare the 2 different initialization methods, random vs grid-search (labelled uniform on the following table). Ozbulak's results are on the third column to compare what he has found and whether the results are consistent or not. Note the dropout that is used and the filter size. For a list of the hyperparameters, and parameter ranges in the Gabor filter, please refer to Table 4.2. In our tests, we use multiple dropouts and different filter sizes.

CIFAR-10								
Dropout	Filter Size	Epoch	Training			Validation		
			Random	Uniform	Ozbulak	Random	Uniform	Ozbulak
0	96	1	27.66	27.26	32.44	39.55	39.35	45.00
		2	46.54	43.83	52.50	54.73	52.00	59.21
		3	56.91	54.20	61.77	62.56	56.76	62.24
		4	63.43	60.64	66.72	61.70	60.93	66.64
		5	67.39	64.94	70.60	67.28	66.08	69.80
		6	70.93	68.43	73.36	70.11	65.87	70.90
		7	73.77	71.18	75.62	67.71	70.71	73.54
		8	75.54	73.74	77.33	73.41	70.69	74.40
		9	77.97	75.61	79.26	71.63	73.80	74.85
		10	79.36	77.21	80.50	75.06	73.77	75.73

Table 4.35: A comparison table of the first 10 epochs of training for CIFAR-10. Here we compare the 2 different initialization methods, random vs grid-search (labelled uniform on the following table). Ozbulak's results are on the third column to compare what he has found and whether the results are consistent or not. Note the dropout that is used and the filter size. For a list of the hyperparameters, and parameter ranges in the Gabor filter, please refer to Table 4.2. In our tests, we use multiple dropouts and different filter sizes.

CIFAR-100								
Dropout	Filter Size	Epoch	Training			Validation		
			Random	Uniform	Ozbulak	Random	Uniform	Ozbulak
0	96	1	2.20	1.86	2.51	3.98	3.37	5.06
		2	4.99	5.16	7.66	6.67	7.02	10.16
		3	9.09	8.70	11.68	9.48	10.60	12.72
		4	12.04	12.35	14.32	13.82	14.08	16.03
		5	14.60	14.68	17.34	16.04	15.90	18.49
		6	17.12	16.76	20.12	17.41	18.29	20.71
		7	19.20	19.01	22.75	20.76	19.69	23.45
		8	21.45	21.27	24.75	20.94	22.63	24.33
		9	23.10	23.38	26.71	23.70	23.48	26.64
		10	24.50	25.01	27.89	25.05	23.97	28.55

Table 4.36: A comparison table of the first 10 epochs of training for CIFAR-100. Here we compare the 2 different initialization methods, random vs grid-search (labelled uniform on the following table). Ozbulak's results are on the third column to compare what he has found and whether the results are consistent or not. Note the dropout that is used and the filter size. For a list of the hyperparameters, and parameter ranges in the Gabor filter, please refer to Table 4.2. In our tests, we use multiple dropouts and different filter sizes.

Discussion

Looking at Tables 4.34, 4.35, and 4.36, we can see that using either initialization method gives similar results. In our tests, we do 10 trials and we show the first 10 epochs of training for the CNN. Please note that Ozbulak uses a grid-search like method to initialize his CNN using the Gabor filter in the first layer. We can see that in most cases, Ozbulak's results are the highest when we look at the CIFAR-10 and CIFAR-100 datasets. This follows the trend of having a uniform (grid-search) distribution provides a higher performance for the network when working with these more complex datasets. For MNIST, the results are very close to each other with no noticeable difference. Ozbulak's results also show a higher training and testing accuracy per epoch on average which cannot be explained other than the results are not reproducible. All 3 datasets show this pattern, all of which cannot be reproduced. It is more noticeable in the CIFAR-10 and CIFAR-100 datasets.

4.3.3 Summary

[New: Dropout adds no benefits] Adding dropout is not necessary when training on MNIST, CIFAR-10, and CIFAR-100. By adding dropout, it reduces the performance of the CNN and in some cases, it will cut the performance by half. Training with CIFAR-10 was the only dataset where we saw overfitting even with dropout.

[New: Increasing filters adds performance] On average, increasing the number of filters improves the performance of the network with a small gain in accuracy and a reduction in loss. For MNIST, using a smaller filter size is more optimal as the performance gain is very small for the amount of extra filters (and hence more parameters) added. For CIFAR-10 and CIFAR-100, if a small performance boost is required, using more filters is advised since there is more features to extract with the Gabor filter and more variations can offer more extracted features.

[New: MNIST →random GCNN, CIFAR-10 and CIFAR-100 →grid-search GCNN] MNIST favours a randomized initialization for the Gabor filter, while both CIFAR-10 and CIFAR-100 favour a grid search initialization. In the CIFAR-10 and CIFAR-100 tests, using grid-search is

far superior because on average the validation accuracy was much higher than using a random initialization. Depending on the dataset, it appears that random initialization is superior for simple datasets like MNIST, and grid-search is superior for datasets that are more complex, like CIFAR-10 and CIFAR-100.

4.4 D: Grid Search Vs. Random Search with Non-Shuffled Datasets Training on MNIST, CIFAR-10, and CIFAR-100

4.4.1 Experiment Setup

In this experiment, the Gabor bank is the same as Section 4.3. The hyperparameters used are the same from Section 4.3 and as described in Section 3.3. This experiment has a total of 9 tests, each with 10 trials. The averages of the trials are taken and are shown in the following tables below.

The Gabor filter bank will change depending on the number of filter supplied to the first layer. There are 3 different number of filters to change for 3 different dropouts. The randomization and grid-search parameter ranges are listed in Table 4.2.

This experiment's main focus is to see how well the Gabor filter bank (either random or grid-search initialized) performs on a non-shuffled dataset. What this means is that for each trial, the dataset used is in the same order, no shuffling occurs. This also happens within each epoch so that there is absolutely no shuffling of the dataset. The validation data is not shuffled as well. Here we can compare the results with Section 4.3, where the dataset is shuffled for all epochs, trials, and tests.

CIFAR-10										
Dropout	Filter Size	Epoch	Random				Uniform			
			Training		Validation		Training		Validation	
			Acc	Loss	Acc	Loss	Acc	Loss	Acc	Loss
0.5	96	1	0.2843	1.8616	0.4272	1.5425	0.2920	1.8398	0.4407	1.4968
		10	0.7849	0.6065	0.7272	0.7824	0.7912	0.5958	0.7304	0.7994
		20	0.8936	0.3125	0.7394	0.8878	0.8984	0.3066	0.7594	0.8038
		30	0.9498	0.1607	0.7762	0.9372	0.9483	0.1642	0.7666	0.9772
		40	0.9779	0.0793	0.7732	1.1691	0.9768	0.0829	0.7766	1.0662
		50	0.9969	0.0240	0.7851	1.3590	0.9941	0.0345	0.7778	1.2876
		60	0.9996	0.0090	0.7833	1.5250	0.9990	0.0127	0.7826	1.4072
		70	1.0000	0.0034	0.7842	1.6205	1.0000	0.0047	0.7758	1.5899
		80	1.0000	0.0022	0.7843	1.6795	1.0000	0.0028	0.7848	1.5924
		90	1.0000	0.0016	0.7849	1.7193	1.0000	0.0020	0.7873	1.6336
		100	1.0000	0.0013	0.7838	1.7478	1.0000	0.0016	0.7877	1.6624

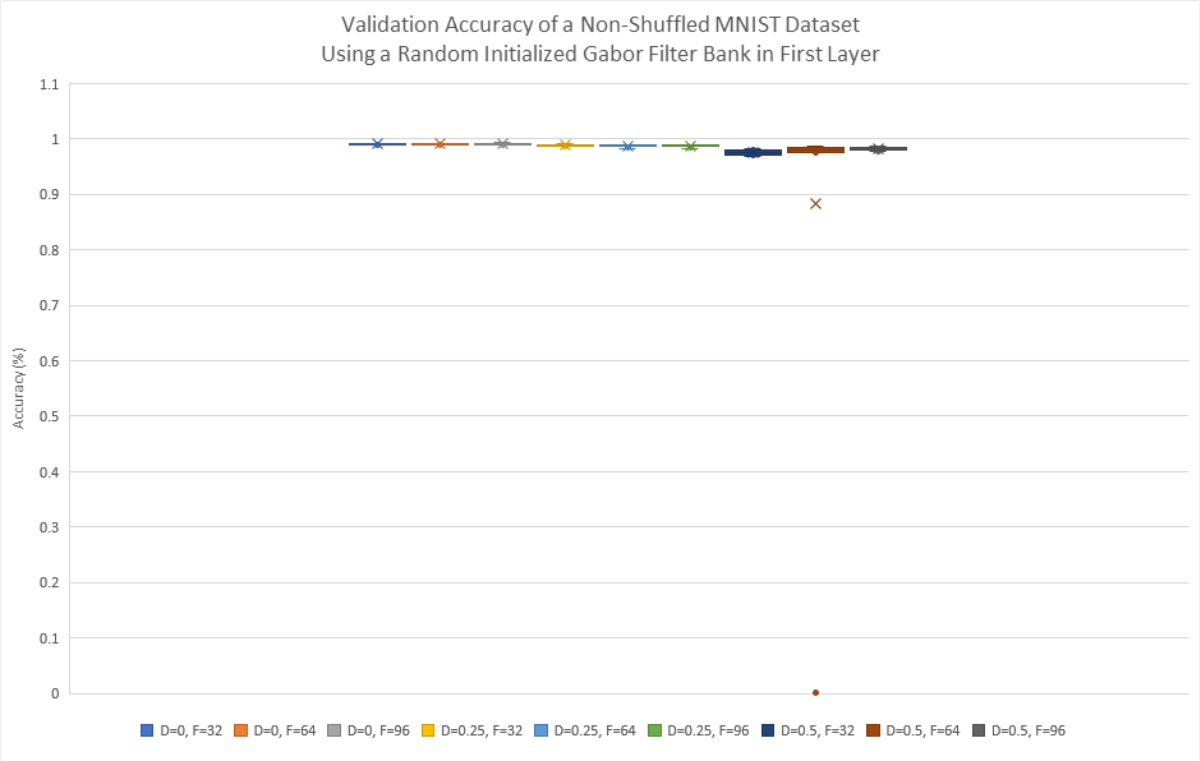
Table 4.37: A trial run showing how our Gabor initialized CNN trains on the CIFAR-10 dataset. There are a total of 100 epochs, where each epoch runs the same non-shuffled dataset. Aside from the first epoch, every 10th epoch is recorded. The dropout, $D = 0$, and the number of filters, $F = 96$, show that there is overfitting occurring around epoch 20.

4.4.2 Results

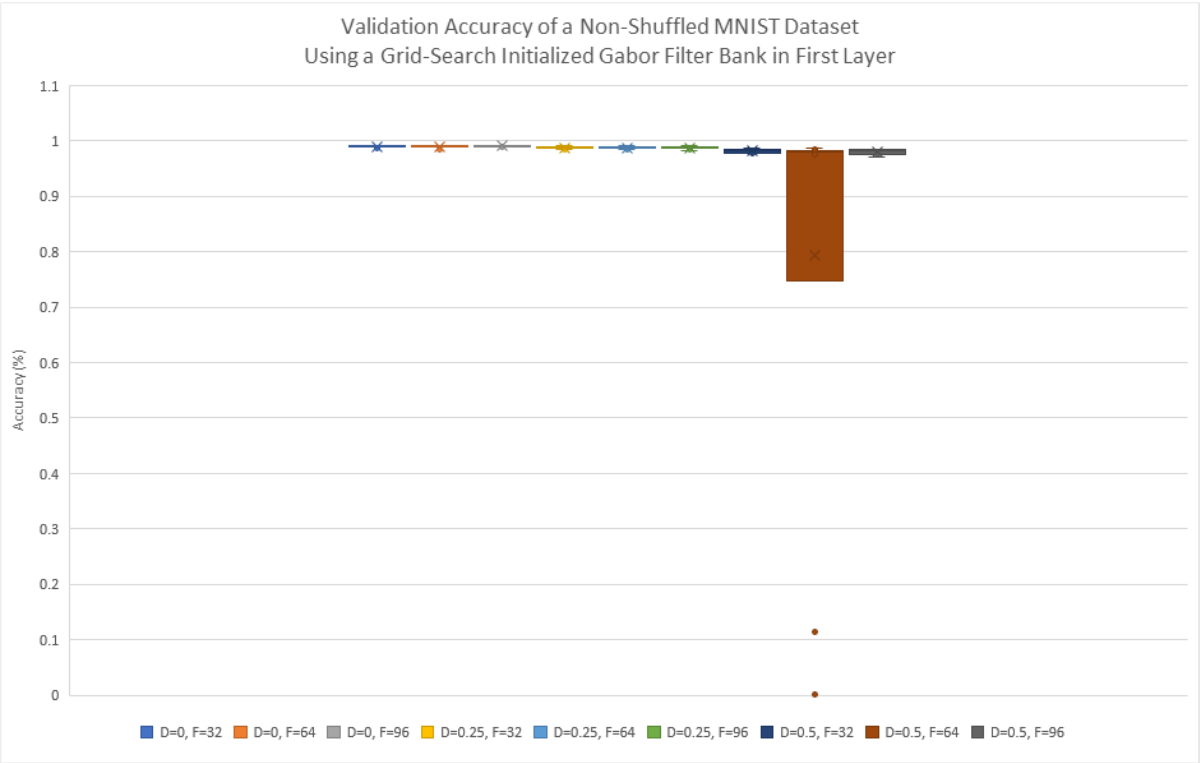
MNIST

MNIST														
Dropout	Test Type	# of Filters	Training						Validation					
			Acc	Loss	Std. Dev Acc	Std. Dev Loss	Var Acc	Var Loss	Acc	Loss	Std. Dev Acc	Std. Dev Loss	Var Acc	Var Loss
0	Random	32	0.9931	0.0229	4.58E-04	1.44E-03	2.10E-07	2.08E-06	0.9905	0.0276	8.92E-04	2.45E-03	7.96E-07	5.99E-06
		64	0.9936	0.0214	6.76E-04	2.09E-03	4.57E-07	4.37E-06	0.9907	0.0280	1.26E-03	3.50E-03	1.58E-06	1.23E-05
		96	0.9938	0.0212	7.50E-04	2.37E-03	5.62E-07	5.63E-06	0.9913	0.0256	9.39E-04	2.68E-03	8.82E-07	7.16E-06
	Uniform	32	0.9926	0.0249	4.12E-04	1.39E-03	1.70E-07	1.93E-06	0.9899	0.0306	9.65E-04	3.21E-03	9.32E-07	1.03E-05
		64	0.9930	0.0241	7.81E-04	2.84E-03	6.10E-07	8.05E-06	0.9899	0.0311	1.60E-03	5.46E-03	2.57E-06	2.98E-05
		96	0.9933	0.0229	4.55E-04	1.32E-03	2.07E-07	1.74E-06	0.9906	0.0282	6.81E-04	1.94E-03	4.63E-07	3.77E-06
0.25	Random	32	0.9363	0.1878	6.15E-03	1.49E-02	3.78E-05	2.23E-04	0.9885	0.0423	1.20E-03	3.64E-03	1.44E-06	1.33E-05
		64	0.9379	0.1883	1.18E-02	4.59E-02	1.39E-04	2.11E-03	0.9873	0.0501	2.14E-03	1.54E-02	4.58E-06	2.36E-04
		96	0.9350	0.1984	1.75E-02	4.32E-02	3.05E-04	1.87E-03	0.9875	0.0505	2.34E-03	1.53E-02	5.46E-06	2.35E-04
	Uniform	32	0.9399	0.1830	6.72E-03	1.89E-02	4.52E-05	3.57E-04	0.9874	0.0496	1.59E-03	7.83E-03	2.52E-06	6.14E-05
		64	0.9411	0.1772	4.22E-03	1.62E-02	1.78E-05	2.64E-04	0.9877	0.0453	1.76E-03	9.11E-03	3.09E-06	8.30E-05
		96	0.9434	0.1717	5.31E-03	1.32E-02	2.82E-05	1.76E-04	0.9878	0.0432	2.42E-03	6.83E-03	5.86E-06	4.66E-05
0.5	Random	32	0.7730	0.6519	1.67E-02	8.28E-02	2.78E-04	6.86E-03	0.9756	0.1516	5.18E-03	5.34E-02	2.69E-05	2.86E-03
		64	0.7915	0.6023	3.15E-02	6.97E-02	9.95E-04	4.86E-03	0.9817	0.1149	3.70E-03	4.22E-02	1.37E-05	1.78E-03
		96	0.8007	0.5884	2.55E-02	3.57E-02	6.49E-04	1.27E-03	0.9824	0.1098	3.02E-03	1.91E-02	9.10E-06	3.63E-04
	Uniform	32	0.7891	0.6095	1.99E-02	4.46E-02	3.96E-04	1.99E-03	0.9815	0.1257	3.04E-03	3.60E-02	9.25E-06	1.29E-03
		64	0.7297	0.7834	2.21E-01	5.48E-01	4.87E-02	3.00E-01	0.8925	0.3644	2.74E-01	6.88E-01	7.50E-02	4.74E-01
		96	0.7966	0.5916	2.87E-02	5.78E-02	8.23E-04	3.34E-03	0.9803	0.1306	4.40E-03	4.44E-02	1.94E-05	1.97E-03

Table 4.38: The following results are the averages of 10 trials for each test using MNIST as the dataset. Each test we run have a different dropout, $D = [0, 0.25, 0.5]$, and the number of filters, $F = [32, 64, 96]$. Each row provides the accuracy, loss, standard deviation, and variance for both training and validation results. Note: Uniform is the term we are using to describe Ozbulak's results in [16]. The results are from using a non-shuffled dataset instead of being shuffled like experiment C (Section 4.3).



(a)



(b)

Figure 4.33: Here we see a different representation of Table 4.38 showing the mean, maximum and minimum values as well as the mean of the validation accuracy. Within the legends are each permutation of the experiment using the non-shuffled MNIST dataset in both training and validation sets.

Discussion

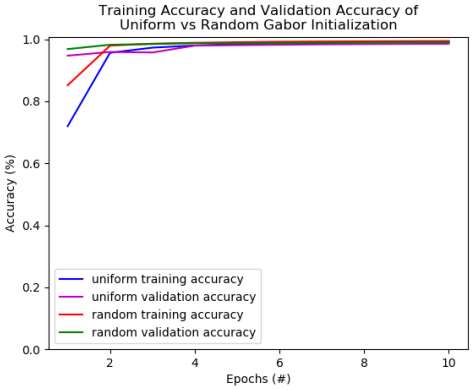
When training on the MNIST dataset using a non-shuffled dataset, we see that there is a drop when we introduce dropout to the network. As we increase the dropout, we see a linear drop in terms of accuracy, and an increase in loss as well. This is a continuous trend we see from the previous experiments A, B, and C (Sections 4.1, 4.2, and 4.3. There is no benefit for adding dropout.

In all of the test trials we run for this experiment, we see that as we increase the number of filters, the validation accuracy increases. The only test that does not show an improvement is the final test where dropout, $D = 0.50$, using a uniform (grid-search) distribution. This result appears to be unreliable as the error is higher than normal, as well as the standard deviation and variances are much higher than other tests. The trend of increasing the number of filters appears in all of the experiments discussed so far and continues in this experiment. One thing to note however is that because the results are so close together, there is no real benefit in adding more filters to the network.

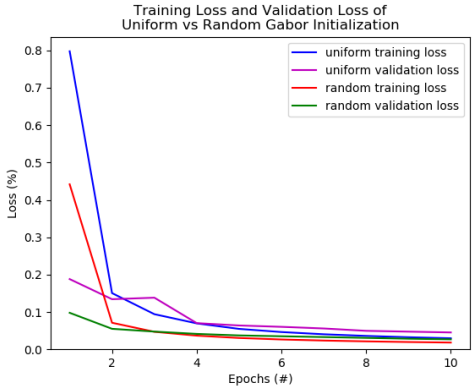
In both initialization methods, using random and uniform (grid-search) distributions for the first layer show an increase in performance by adding more filters. For MNIST, the trend was to see a random initialization method outperforming the uniform distribution, which is still noticeable here. 5 of the 9 tests provided when comparing 3 different dropouts at 3 different number of filters, shows that random performs better than using a grid-search initialization for the Gabor filter in the first layer. For example, when there is 50% dropout with 96 filters, there is a 0.21% increase for using a random initialization. Although it is an improvement for using a random initialization, it is not significant.

In summary for working with MNIST, adding dropout reduces the performance of the network. Increasing the number of filters improves the performance of the network, but not by much. If speed is the goal, using less filters will be more beneficial since there will consequently be less parameters to tune. Random initialization for the Gabor filter in the first layer is more successful than using a uniform (grid-search) distribution, but the differences are very

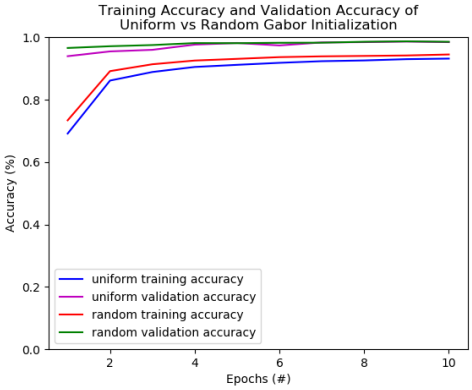
marginal.



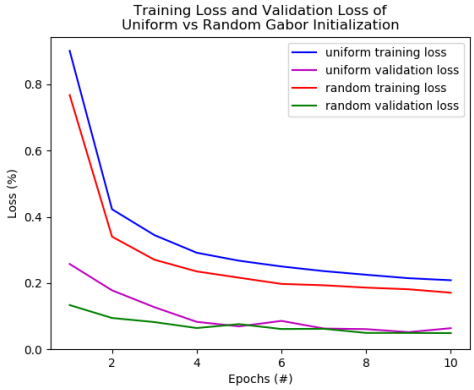
(a) Accuracy: $D = 0$, $F = 64$



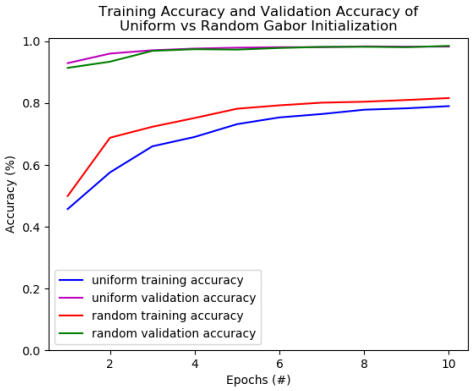
(b) Loss: $D = 0$, $F = 64$



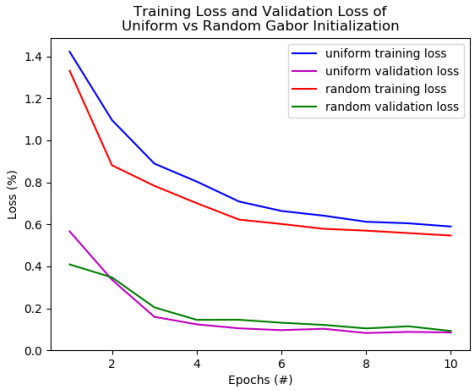
(c) Accuracy: $D = 0.25$, $F = 64$



(d) Loss: $D = 0.25$, $F = 64$



(e) Accuracy: $D = 0.50$, $F = 64$



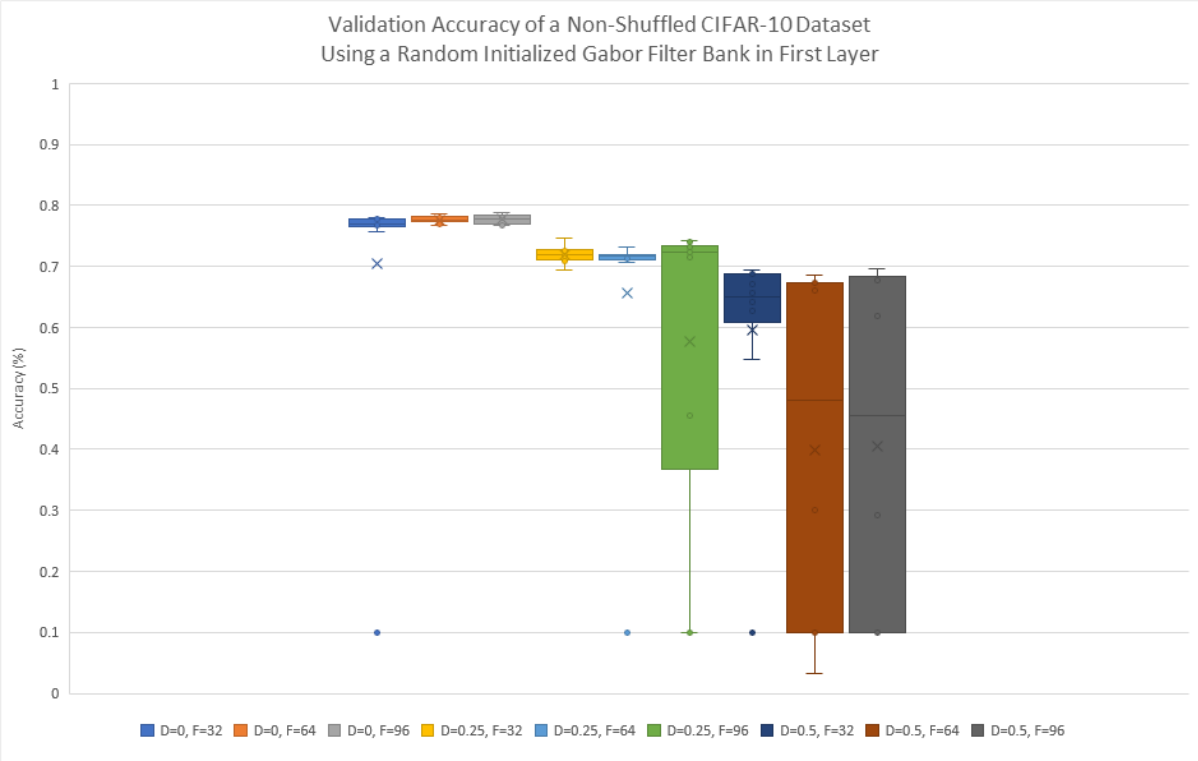
(f) Loss: $D = 0.50$, $F = 64$

Figure 4.34: Training on the MNIST dataset shows with no shuffling for the dataset shows a fast training with high results. We continue the trend of seeing a dip in the performance when we add dropout, albeit, an insignificant amount. Using either initialization method offers similar performance. Comparing with Figure 4.28, we can see that shuffling the dataset almost offers no difference. The Gabor CNN plateaus its performance very early.

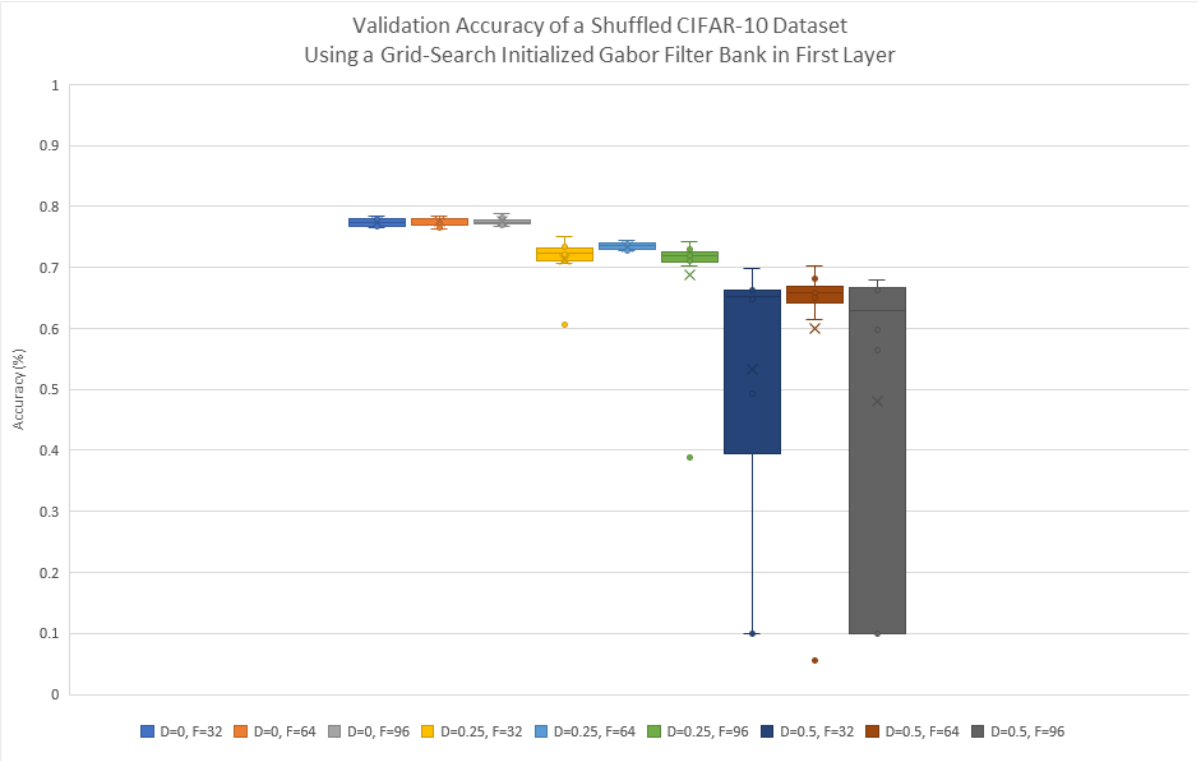
CIFAR-10

CIFAR-10															
Dropout	**Test Type**	**# of Filters**	**Training**						**Validation**						
			Acc	**Loss**	**Std. Dev Acc**	**Std. Dev Loss**	**Var Acc**	**Var Loss**	**Acc**	**Loss**	**Std. Dev Acc**	**Std. Dev Loss**	**Var Acc**	**Var Loss**
0	Random	32	0.9096	0.2324	2.86E-01	7.27E-01	8.17E-02	5.29E-01	0.7046	1.7995	2.13E-01	1.87E-01	4.52E-02	3.51E-02
		64	1.0000	0.0020	2.15E-05	5.56E-04	4.62E-10	3.09E-07	0.7767	1.7504	5.51E-03	5.69E-02	3.04E-05	3.24E-03
		96	1.0000	0.0016	1.03E-05	4.90E-04	1.07E-10	2.40E-07	0.7771	1.7585	7.89E-03	7.80E-02	6.23E-05	6.09E-03
	Uniform	32	1.0000	0.0024	1.26E-05	2.32E-04	1.60E-10	5.37E-08	0.7740	1.7617	6.45E-03	6.45E-02	4.16E-05	4.16E-03
		64	1.0000	0.0020	1.03E-05	3.11E-04	1.07E-10	9.68E-08	0.7758	1.7632	7.26E-03	8.60E-02	5.28E-05	7.39E-03
		96	1.0000	0.0019	1.40E-05	3.08E-04	1.96E-10	9.47E-08	0.7753	1.7642	5.97E-03	5.76E-02	3.56E-05	3.32E-03
0.25	Random	32	0.8269	0.4735	4.97E-02	1.36E-01	2.47E-03	1.86E-02	0.7199	1.0167	1.40E-02	8.27E-02	1.97E-04	6.83E-03
		64	0.7504	0.6768	2.41E-01	5.97E-01	5.80E-02	3.56E-01	0.6556	1.1764	1.95E-01	4.12E-01	3.82E-02	1.70E-01
		96	0.6753	0.8621	3.45E-01	8.57E-01	1.19E-01	7.34E-01	0.5764	1.3589	2.65E-01	5.14E-01	7.04E-02	2.64E-01
	Uniform	32	0.8297	0.4809	1.03E-01	2.79E-01	1.07E-02	7.78E-02	0.7127	1.0833	3.94E-02	1.04E-01	1.55E-03	1.08E-02
		64	0.8734	0.3638	1.02E-02	2.87E-02	1.04E-04	8.26E-04	0.7348	1.0497	5.93E-03	4.77E-02	3.52E-05	2.28E-03
		96	0.8082	0.5297	1.68E-01	4.18E-01	2.81E-02	1.75E-01	0.6869	1.1551	1.05E-01	1.73E-01	1.11E-02	3.01E-02
0.5	Random	32	0.5554	1.1667	1.83E-01	4.44E-01	3.33E-02	1.97E-01	0.5956	1.2421	1.80E-01	3.88E-01	3.22E-02	1.51E-01
		64	0.4113	1.5269	2.64E-01	6.51E-01	6.99E-02	4.24E-01	0.4425	1.5664	2.66E-01	5.76E-01	7.06E-02	3.32E-01
		96	0.3937	1.5701	2.90E-01	7.10E-01	8.43E-02	5.04E-01	0.4048	1.6590	2.87E-01	5.98E-01	8.26E-02	3.58E-01
	Uniform	32	0.4917	1.3337	2.25E-01	5.50E-01	5.07E-02	3.03E-01	0.5324	1.3551	2.34E-01	5.14E-01	5.49E-02	2.64E-01
		64	0.6177	1.0186	7.27E-02	1.62E-01	5.29E-03	2.62E-02	0.6549	1.1139	2.90E-02	8.10E-02	8.39E-04	6.56E-03
		96	0.4466	1.4363	2.51E-01	6.19E-01	6.28E-02	3.83E-01	0.4804	1.4922	2.65E-01	5.67E-01	7.02E-02	3.21E-01

Table 4.39: The following results are the averages of 10 trials for each test using CIFAR-10 as the dataset. Each test we run have a different dropout, $D = [0, 0.25, 0.5]$, and the number of filters, $F = [32, 64, 96]$. Each row provides the accuracy, loss, standard deviation, and variance for both training and validation results. Note: Uniform is the term we are using to describe Ozbulak's results in [16]. The results are from using a non-shuffled dataset instead of being shuffled like experiment C (Section 4.3).



(a)



(b)

Figure 4.35: Here we see a different representation of Table 4.39 showing the mean, maximum and minimum values as well as the mean of the validation accuracy. Within the legends are each permutation of the experiment using the non-shuffled CIFAR-10 dataset in both training and validation sets.

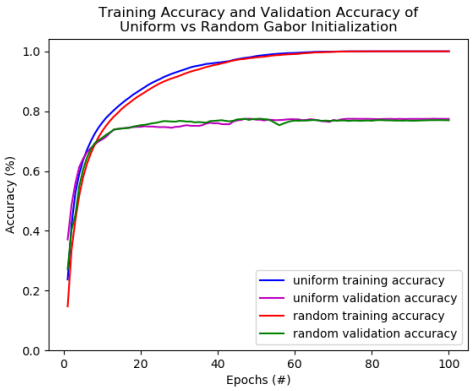
Discussion

When training on the CIFAR-10 dataset, we see the same trend we have seem in all previous experiments, where by adding dropout, the performance of the network is reduced. This occurs in all filters given and in both initialization methods of random versus grid-search. There is no benefit for adding dropout to the network. We can see from the training results that the network is overfitting as well. This is the same result from experiment C (Section 4.3) where increasing the dropout only gives the network a couple more epoch rounds and finally overfitting regardless of what dropout percentage is used (Figure 4.36).

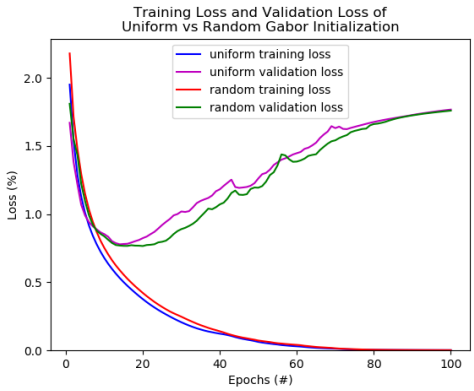
From the results that we have obtained after averaging the 10 trials for each permutation, no dropout results with a linear trend that increasing the filters improves the validation accuracy of the network. Having dropout shows that there is no noticeable pattern. This is inconsistent in what we have seen from experiment C where adding more filters should improve the performance of the network. Looking at the standard deviation and the variance for these inconsistencies shows that the results may be inconsistent and unreliable because the values are higher than the tests where we check the number of filters, $F = 32$. This could mean that the trend can still exist, but requires more tests.

Using random initialization versus grid-search initialization show when training with CIFAR-10, a grid-search distribution is favoured. 5 of the 9 tests comparing both methods shows that uniform initialization provides a higher validation accuracy. Not only does the network perform better with grid-search, the average results are higher than using a random initialization.

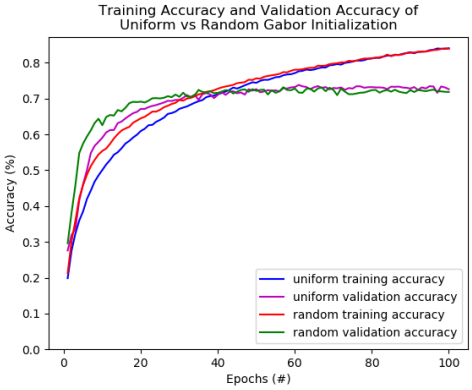
Adding dropout to the network when training on CIFAR-10 shows that the performance drops significantly. The network will overfit on this simple AlexNet-like architecture regardless of how much dropout is given (seen in Figure 4.36) which means that there is no benefit of introducing dropout to this dataset. This is consistent with experiment C. Changing the number of filters in the first layer shows that the performance increases in terms of validation accuracy and a reduction in loss. This continues the trend seen in experiment C. Finally, grid-search is more preferred when training with CIFAR-10, just like experiment C.



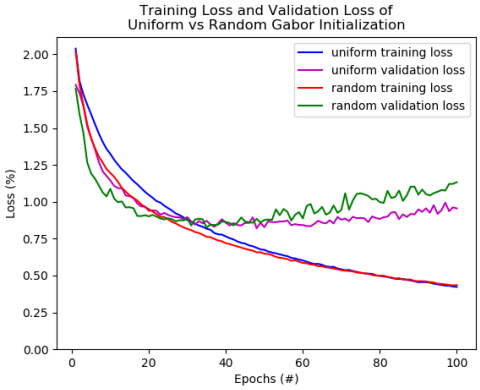
(a) Accuracy: $D = 0$, $F = 32$



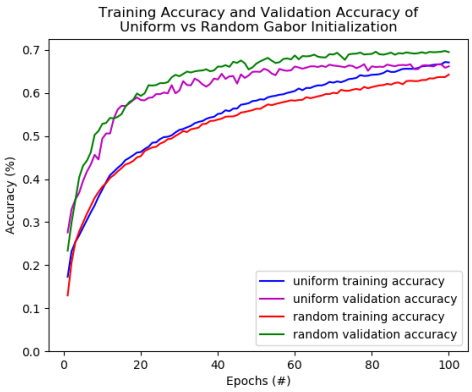
(b) Loss: $D = 0$, $F = 32$



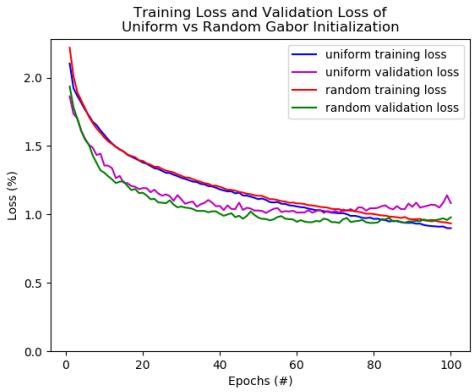
(c) Accuracy: $D = 0.25$, $F = 32$



(d) Loss: $D = 0.25$, $F = 32$



(e) Accuracy: $D = 0.50$, $F = 64$



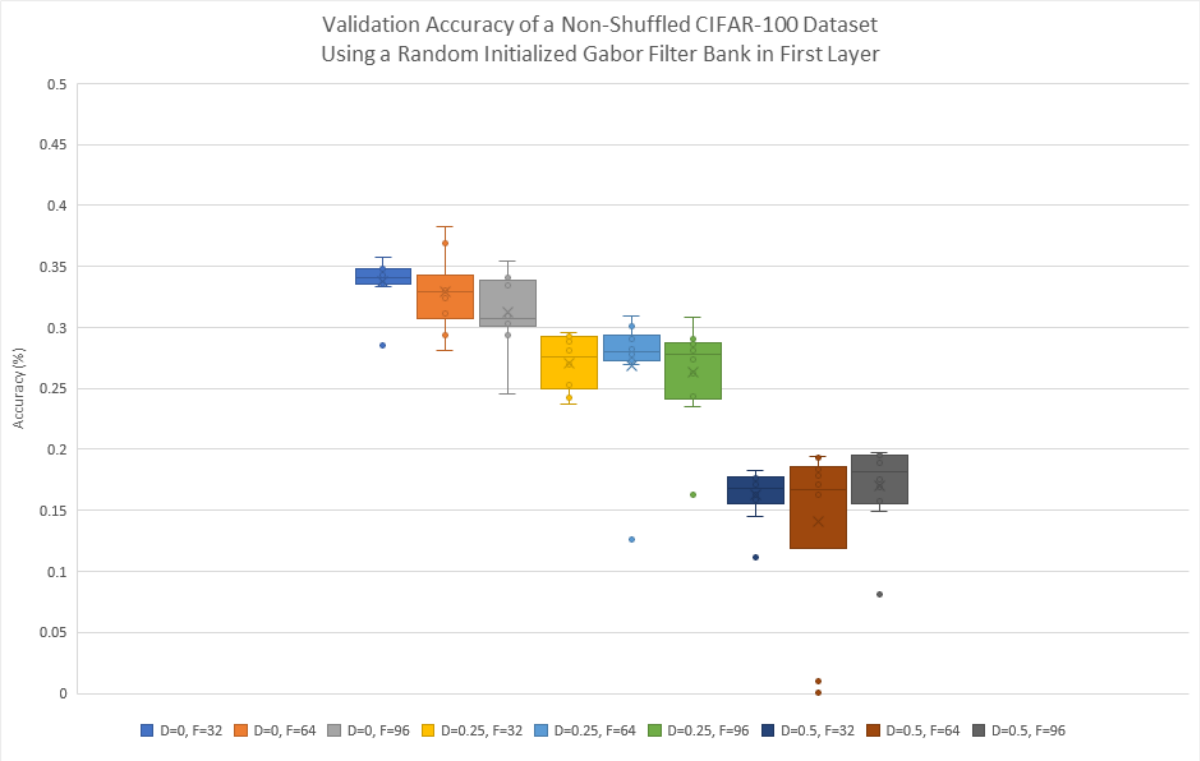
(f) Loss: $D = 0.50$, $F = 64$

Figure 4.36: Training on the CIFAR-10 dataset results in overfitting even with dropout using our network described in Section 3.3. As we add dropout to both random and uniform initialization methods, we increase the epoch at which we reach the point where the CNN overfits. When dropout, $D = 0$, we can see that around epoch, $E = 18$ the network starts to overfit. When $D = 0.25$, we see that the network overfits around $E = 30$, and finally when $D = 0.50$, the epoch at which it overfits is about $E = 35$. The performance of the network surprisingly drops with an increased dropout. This is similar to the result found in Table 4.30.

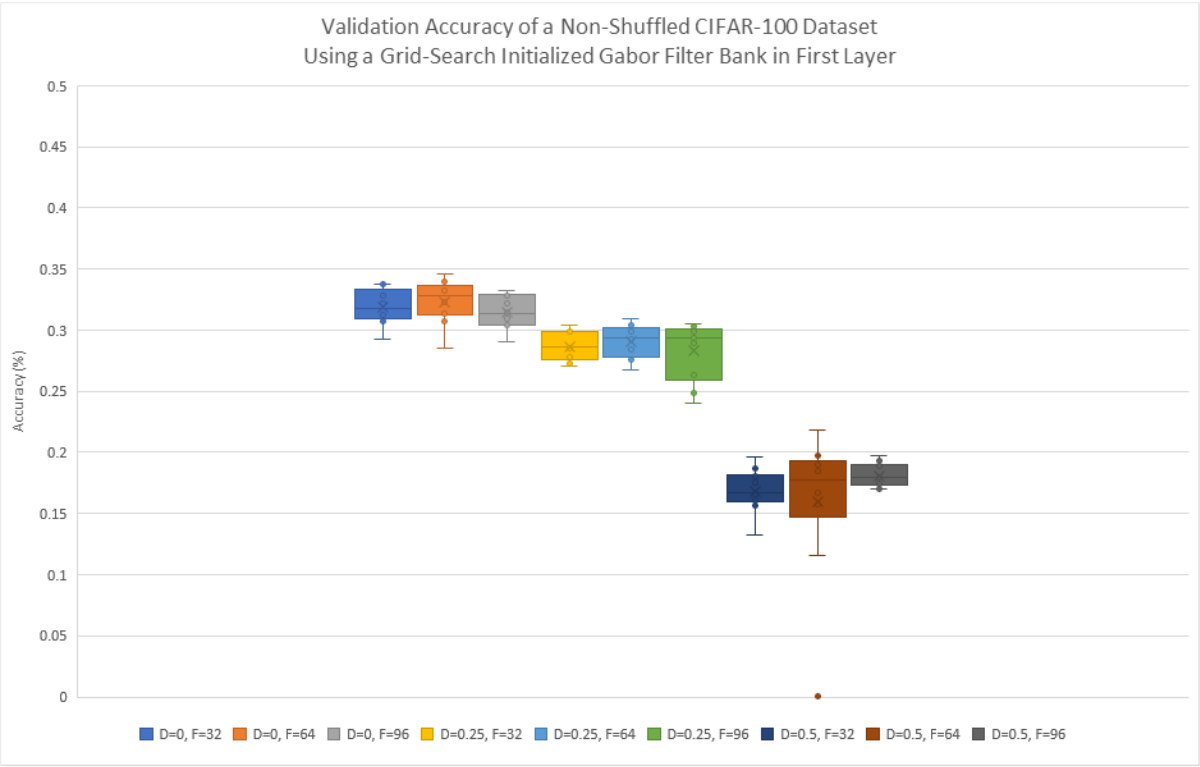
CIFAR-100

CIFAR-100															
Dropout	**Test Type**	**# of Filters**	**Training**						**Validation**						
			Acc	**Loss**	**Std. Dev Acc**	**Std. Dev Loss**	**Var Acc**	**Var Loss**	**Acc**	**Loss**	**Std. Dev Acc**	**Std. Dev Loss**	**Var Acc**	**Var Loss**
0	Random	32	0.6722	1.0659	3.55E-02	1.32E-01	1.26E-03	1.75E-02	0.3375	3.5132	1.98E-02	4.09E-01	3.91E-04	1.67E-01
		64	0.7062	0.9318	5.67E-02	1.90E-01	3.22E-03	3.61E-02	0.3289	3.9445	3.05E-02	5.77E-01	9.28E-04	3.33E-01
		96	0.7641	0.7462	3.49E-02	1.13E-01	1.22E-03	1.27E-02	0.3128	4.5071	3.11E-02	6.74E-01	9.68E-04	4.54E-01
	Uniform	32	0.7064	0.9331	2.25E-02	6.69E-02	5.08E-04	4.48E-03	0.3192	3.9506	1.48E-02	2.91E-01	2.20E-04	8.48E-02
		64	0.7662	0.7433	2.41E-02	7.20E-02	5.79E-04	5.19E-03	0.3232	4.3605	1.84E-02	4.15E-01	3.40E-04	1.72E-01
		96	0.7649	0.7368	3.38E-02	1.03E-01	1.14E-03	1.06E-02	0.3150	4.5593	1.34E-02	3.62E-01	1.80E-04	1.31E-01
0.25	Random	32	0.1940	3.1275	2.37E-02	1.58E-01	5.59E-04	2.50E-02	0.2710	2.9229	2.27E-02	1.27E-01	5.13E-04	1.62E-02
		64	0.2033	3.0405	4.33E-02	3.21E-01	1.88E-03	1.03E-01	0.2687	2.9031	5.17E-02	3.06E-01	2.67E-03	9.33E-02
		96	0.1955	3.0788	4.09E-02	2.88E-01	1.67E-03	8.31E-02	0.2630	2.9373	4.18E-02	2.30E-01	1.74E-03	5.31E-02
	Uniform	32	0.2107	2.9898	1.23E-02	7.17E-02	1.52E-04	5.14E-03	0.2864	2.8185	1.20E-02	5.82E-02	1.43E-04	3.39E-03
		64	0.2309	2.8630	1.47E-02	8.04E-02	2.17E-04	6.47E-03	0.2906	2.7904	1.37E-02	6.48E-02	1.87E-04	4.20E-03
		96	0.2137	2.9684	2.95E-02	1.90E-01	8.69E-04	3.60E-02	0.2835	2.8331	2.39E-02	1.26E-01	5.72E-04	1.58E-02
0.5	Random	32	0.0834	3.8118	1.18E-02	1.02E-01	1.40E-04	1.04E-02	0.1629	3.5217	2.13E-02	1.27E-01	4.55E-04	1.61E-02
		64	0.0796	3.8442	2.69E-02	2.94E-01	7.24E-04	8.66E-02	0.1529	3.5695	5.47E-02	3.91E-01	2.99E-03	1.53E-01
		96	0.0908	3.7211	1.66E-02	1.93E-01	2.76E-04	3.73E-02	0.1704	3.4356	3.57E-02	2.43E-01	1.27E-03	5.89E-02
	Uniform	32	0.0859	3.7851	1.16E-02	1.05E-01	1.34E-04	1.10E-02	0.1685	3.4851	1.77E-02	1.16E-01	3.12E-04	1.35E-02
		64	0.0924	3.7101	1.48E-02	1.34E-01	2.19E-04	1.79E-02	0.1784	3.4198	2.81E-02	1.70E-01	7.89E-04	2.88E-02
		96	0.0911	3.7255	6.56E-03	5.91E-02	4.31E-05	3.49E-03	0.1811	3.4135	9.24E-03	6.01E-02	8.54E-05	3.62E-03

Table 4.40: The following results are the averages of 10 trials for each test using CIFAR-100 as the dataset. Each test we run have a different dropout, $D = [0, 0.25, 0.5]$, and the number of filters, $F = [32, 64, 96]$. Each row provides the accuracy, loss, standard deviation, and variance for both training and validation results. Note: Uniform is the term we are using to describe Ozbulak's results in [16]. The results are from using a non-shuffled dataset instead of being shuffled like experiment C (Section 4.3).



(a)



(b)

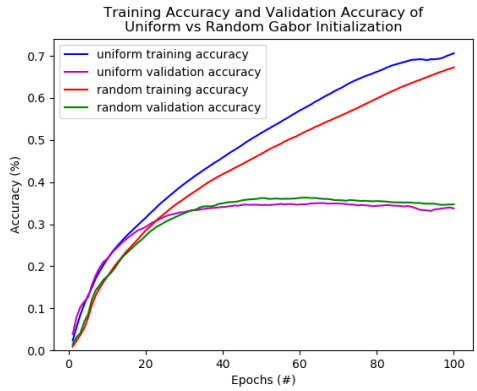
Figure 4.37: Here we see a different representation of Table 4.40 showing the mean, maximum and minimum values as well as the mean of the validation accuracy. Within the legends are each permutation of the experiment using the non-shuffled CIFAR-100 dataset in both training and validation sets.

Discussion

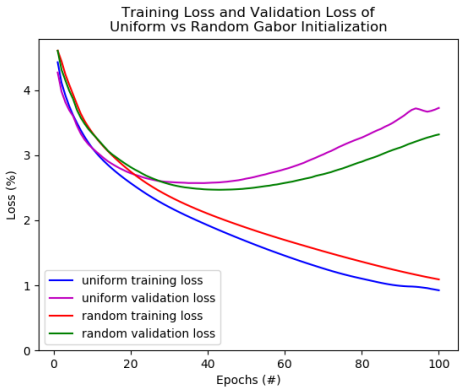
Similar to experiment C, when dropout is introduced to the network, the performance drops dramatically. Here in this case, we see that by adding 50% dropout, there is almost half the performance as if we added no dropout to the network. This means that there is no added benefit of adding dropout to the network. There is no visible overfitting as well within the 100 epochs, but the accuracy and loss curves start to plateau towards 100 epochs.

As with previous experiments and datasets, when we increase the number of filters, we see that the network improves its performance when training on CIFAR-100 dataset. The results that fail to show this trend have a higher loss, standard deviation, and variance, which shows that the result is unreliable. For example, when the network has a dropout, $D = 0.50$, and the number of filters, $F = 32$, the network's validation accuracy increases by 2% when comparing with its neighbor of 96 filters ($F = 96$). This can be seen in Table 4.40.

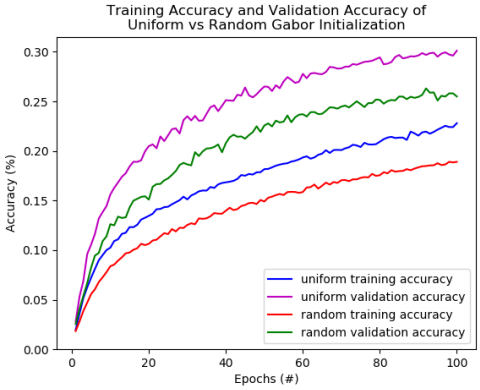
The difference between using random search versus grid-search shows when training with CIFAR-100. The dataset prefers a grid-search distribution as the initialization method for the first layer when using the Gabor filter. In all 9 tests comparing both the initialization methods, 7 tests favoured using grid-search. Using grid-search as the initialization method also allows the network to perform better. The average performance in terms of both training and accuracy is much higher than using a random initialization. This can be seen for all permutations of dropout and the number of filters for both initialization methods.



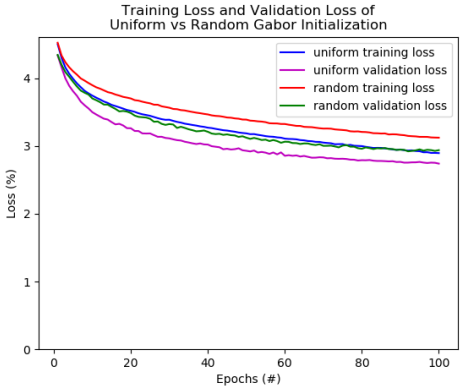
(a) Accuracy: $D = 0$, $F = 32$



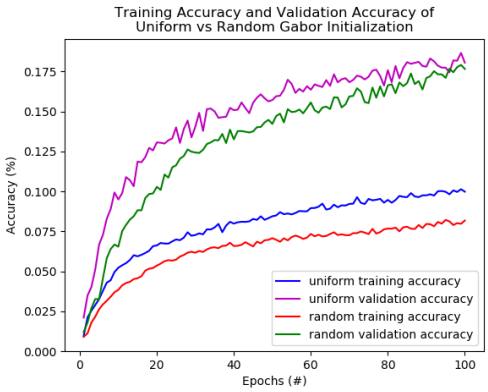
(b) Loss: $D = 0$, $F = 32$



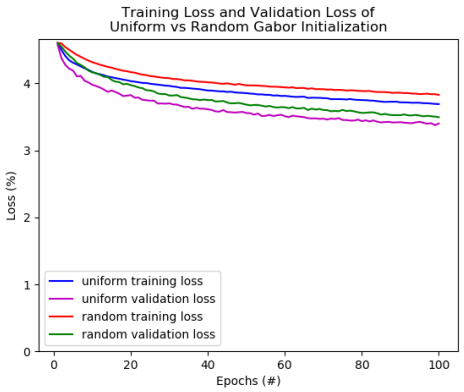
(c) Accuracy: $D = 0.25$, $F = 32$



(d) Loss: $D = 0.25$, $F = 32$



(e) Accuracy: $D = 0.50$, $F = 32$



(f) Loss: $D = 0.50$, $F = 32$

Figure 4.38: Training on the CIFAR-100 dataset shows a drop in performance following the trend when we add dropout to the Gabor CNN. We have seen this from previous tests, and also in experiment C. When dropout, $D = 0$, we see that the network is overfitting at around epoch, $E = 20$. The performance begins to converge towards the end of training (epoch, $E = 100$). Training with a random initialization for the Gabor filter bank shows a decrease in performance in comparison to being initialized randomly.

In summary, when training with CIFAR-100, there is no added benefit of using dropout. The performance drops dramatically when dropout is used, and in some cases, is almost half as bad. Adding more filters will generally produce higher results for all cases shown in Table 4.40. CIFAR-100 also favours grid-search instead of random-search for the first layer Gabor filter initialization.

4.4.3 Summary

[Confirm that dropout adds no benefits] Just like experiment C (Section 4.3), we see that by adding dropout, the network performs worse which means that there is no added benefit for using dropout. Even in the case of CIFAR-10, the graphs in Figure 4.30 and Figure 4.36 show that overfitting is not solved with adding dropout. This applies for all datasets that this network trains on (MNIST, CIFAR-10, and CIFAR-100).

[Confirm that increasing filters adds performance] Adding more filters show that there is an increase in performance for all datasets tested on. Although not very noticeable according to the averages taken, the errors, standard deviations, and variances of the results that seem out of place are higher than results that fit the trend. Adding more filters for MNIST is not as beneficial because there is a marginal loss in terms of both training and validation accuracy of the network. In contrast, it is more beneficial to use more than less filters when training on CIFAR-10 and CIFAR-100.

[Confirm MNIST →random GCNN, CIFAR-10 and CIFAR-100 →grid-search GCNN] In this experiment, using a random initialization method for the Gabor filter in the first layer is only useful for the MNIST dataset. This trend continues from experiment C as well. For the other two datasets, CIFAR-10 and CIFAR-100, the grid-search initialization method is more preferred in both datasets. As a bonus to using the grid-search for these 2 datasets, the performance is boosted on average when comparing to the random initialization.

[New: Non-shuffled datasets slightly drop performance] When we see the results from using either initialization methods, there is no noticeable difference in terms of both training and

validation accuracy (this can be seen in Tables 4.41, 4.42, 4.43, and 4.44). We can conclude in saying that when using a shuffled dataset versus a non-shuffled dataset that the Gabor filter bank used as the first layer trains regardless of initialization method learns its weights normally like any other network. Testing the randomization of sets, we see that the Gabor filter bank has no impact on having a shuffled dataset or a non-shuffled dataset during training.

[Summary of Experiment D] In summary, there is no benefit in adding dropout. Increasing the number of filters results in higher performance. In the case of training on MNIST, using less filters is more beneficial because there is no large gain in performance. When training on CIFAR-10 and CIFAR-100, adding more filters provides a larger boost in performance. Using a random initialization method for the Gabor filter bank in the first layer appears to be better when used on MNIST. Using grid-search initialization is preferred when training on both CIFAR-10 and CIFAR-100 datasets. When using a non-shuffled dataset, it compares similarly to having a shuffled dataset for MNIST, but for CIFAR-10 and CIFAR-100, the performances of using a non-shuffled dataset seem to be lower than using a shuffled dataset.

Random Search							
Dropout	**# of Filters**	**Training Accuracy**					
		Shuffled			**Non-Shuffled**		
		MNIST	**CIFAR-10**	**CIFAR-100**	**MNIST**	**CIFAR-10**	**CIFAR-100**
0	32	0.9932	1.0000	0.6315	0.9931	0.9096	0.6722
	64	0.9934	1.0000	0.7852	0.9936	1.0000	0.7062
	96	0.9937	0.7292	0.6407	0.9938	1.0000	0.7641
0.25	32	0.9342	0.6458	0.1921	0.9363	0.8269	0.1940
	64	0.9414	0.6321	0.2089	0.9379	0.7504	0.2033
	96	0.9339	0.8774	0.2059	0.9350	0.6753	0.1955
0.50	32	0.7867	0.4254	0.0841	0.7730	0.5554	0.0834
	64	0.7966	0.4067	0.0908	0.7915	0.4113	0.0796
	96	0.7838	0.5654	0.0800	0.8007	0.3937	0.0908

Table 4.41: Here are the results of the random search initialization method used for the Gabor filter that initialized the first layer in the network. The training accuracy of each dataset in terms of the dropout and the number of filters given in each test are listed.

Grid Search							
Dropout	# of Filters	Training Accuracy					
		Shuffled			Non-Shuffled		
		MNIST	CIFAR-10	CIFAR-100	MNIST	CIFAR-10	CIFAR-100
0	32	0.9925	1.0000	0.7485	0.9926	1.0000	0.7064
	64	0.9933	1.0000	0.8104	0.9930	1.0000	0.7662
	96	0.9936	0.9852	0.8028	0.9933	1.0000	0.7649
0.25	32	0.9432	0.7696	0.1953	0.9399	0.8297	0.2107
	64	0.9406	0.8309	0.2275	0.9411	0.8734	0.2309
	96	0.9382	0.8596	0.2251	0.9434	0.8082	0.2137
0.50	32	0.7851	0.5588	0.0891	0.7891	0.4917	0.0859
	64	0.7935	0.6507	0.0978	0.7297	0.6177	0.0924
	96	0.7544	0.4956	0.0986	0.7966	0.4466	0.0911

Table 4.42: Here are the results of the grid-search initialization method used for the Gabor filter that initialized the first layer in the network. The training accuracy of each dataset in terms of the dropout and the number of filters given in each test are listed.

Random Search							
Dropout	# of Filters	Validation Accuracy					
		Shuffled			Non-Shuffled		
		MNIST	CIFAR-10	CIFAR-100	MNIST	CIFAR-10	CIFAR-100
0	32	0.9922	0.7758	0.3625	0.9905	0.7046	0.3375
	64	0.9918	0.7805	0.3481	0.9907	0.7767	0.3289
	96	0.9923	0.5736	0.3123	0.9913	0.7771	0.3128
0.25	32	0.9880	0.5686	0.2669	0.9885	0.7199	0.2710
	64	0.9881	0.5436	0.2799	0.9873	0.6556	0.2687
	96	0.9883	0.7334	0.2789	0.9875	0.5764	0.2630
0.50	32	0.9804	0.4738	0.1654	0.9756	0.5956	0.1629
	64	0.9800	0.4365	0.1738	0.9817	0.4425	0.1529
	96	0.9798	0.5648	0.1558	0.9824	0.4048	0.1704

Table 4.43: Here are the results of the random search initialization method used for the Gabor filter that initialized the first layer in the network. The validation accuracy of each dataset in terms of the dropout and the number of filters given in each test are listed.

Grid Search							
Dropout	# of Filters	Validation Accuracy					
		Shuffled			Non-Shuffled		
		MNIST	CIFAR-10	CIFAR-100	MNIST	CIFAR-10	CIFAR-100
0	32	0.9909	0.7782	0.3464	0.9899	0.7740	0.3192
	64	0.9916	0.7829	0.3596	0.9899	0.7758	0.3232
	96	0.9922	0.7644	0.3485	0.9906	0.7753	0.3150
0.25	32	0.9874	0.6683	0.2725	0.9874	0.7127	0.2864
	64	0.9872	0.7221	0.2979	0.9877	0.7348	0.2906
	96	0.9887	0.7295	0.2935	0.9878	0.6869	0.2835
0.50	32	0.9801	0.6112	0.1714	0.9815	0.5324	0.1685
	64	0.9826	0.6837	0.1948	0.8925	0.6549	0.1784
	96	0.9595	0.5700	0.1914	0.9803	0.4804	0.1811

Table 4.44: Here are the results of the grid-search initialization method used for the Gabor filter that initialized the first layer in the network. The validation accuracy of each dataset in terms of the dropout and the number of filters given in each test are listed.

4.5 E: Grid Search Vs. Random Search with a Non-Trainable First Layer and Non-Shuffled Datasets Training on MNIST, CIFAR-10, and CIFAR-100

4.5.1 Experiment Setup

In our fifth experiment, we extend what we have learned from using the non-shuffled and shuffled variants of the datasets and add another limiting factor for the Gabor filter. Instead of allowing the Gabor filter bank in the first layer to learn during training, we disallow the layer to learn new weights at all. We can see that from using a shuffled or a non-shuffled datasets, the Gabor CNN is unaffected (shown in Tables 4.41, 4.43, 4.42, and 4.44). Because we are freezing the first layer's weights as the Gabor filter bank, we can see the effects of the Gabor filter bank on each of the datasets.

During the tests in this experiment, we will explore changing the filters and dropout once

again. We will see 9 different permutations of 3 different dropouts, $D = [0, 0.25, 0.50]$, and 3 different number of filters, $F = [32, 64, 96]$. Other than this, there are no other changes to the network except as described above by freezing the weights learned in the Gabor filter.

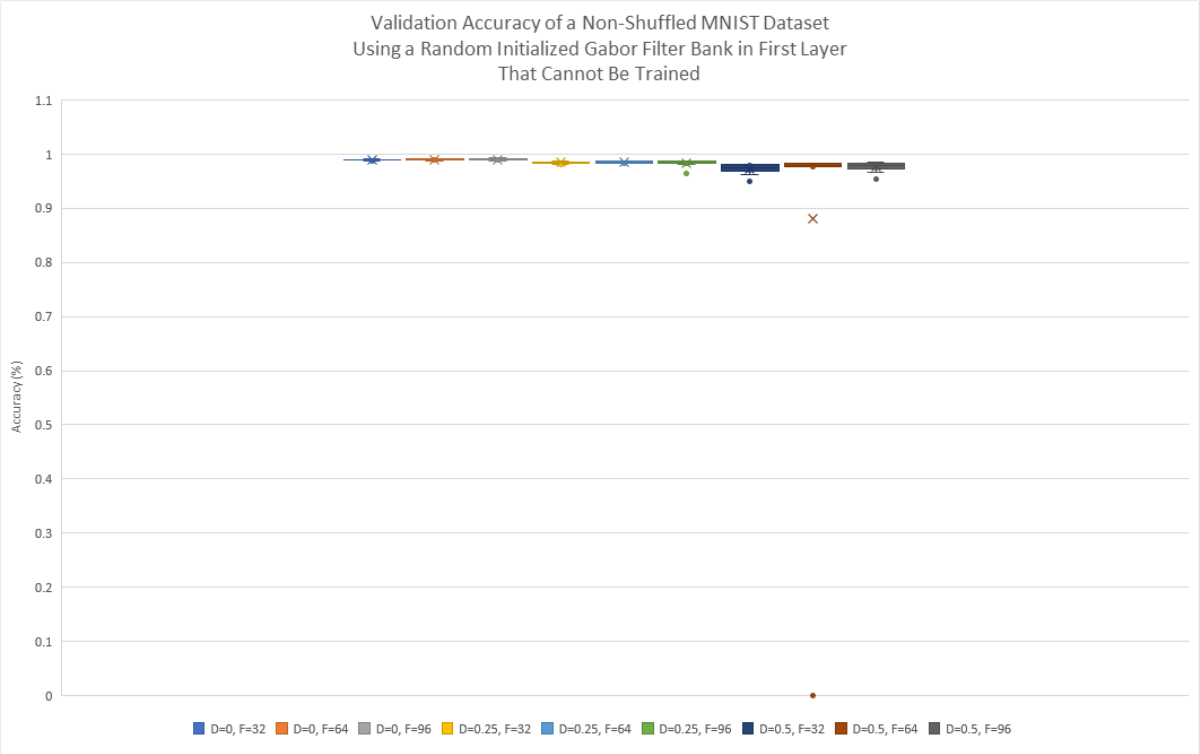
The datasets used are MNIST, CIFAR-10, and CIFAR-100. This experiment is similar to experiment C (Section 4.1) and experiment D (Section 4.4). These datasets will not be shuffled as we have found that there is no change in performance if we shuffle the dataset.

4.5.2 Results

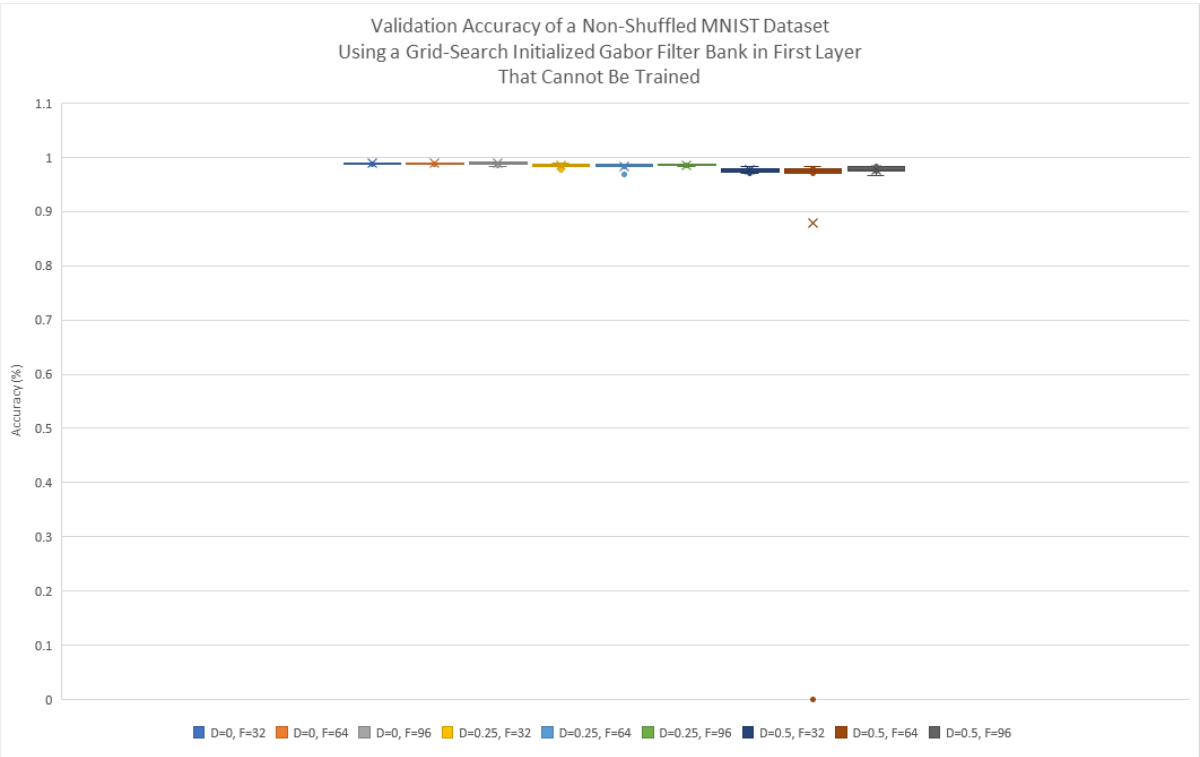
MNIST

MNIST														
Dropout	Test Type	# of Filters	Training						Validation					
			Acc	Loss	Std. Dev Acc	Std. Dev Loss	Var Acc	Var Loss	Acc	Loss	Std. Dev Acc	Std. Dev Loss	Var Acc	Var Loss
0	Random	32	0.9914	0.0285	6.05E-04	1.89E-03	3.66E-07	3.58E-06	0.9890	0.0330	8.29E-04	2.46E-03	6.87E-07	6.05E-06
		64	0.9922	0.0257	6.12E-04	1.76E-03	3.75E-07	3.09E-06	0.9900	0.0302	1.03E-03	3.25E-03	1.07E-06	1.05E-05
		96	0.9931	0.0229	7.08E-04	2.14E-03	5.01E-07	4.57E-06	0.9900	0.0293	1.56E-03	3.94E-03	2.43E-06	1.55E-05
	Uniform	32	0.9909	0.0302	6.13E-04	1.85E-03	3.76E-07	3.41E-06	0.9887	0.0334	9.17E-04	2.83E-03	8.40E-07	8.02E-06
		64	0.9911	0.0293	5.73E-04	1.75E-03	3.28E-07	3.06E-06	0.9885	0.0341	6.52E-04	1.86E-03	4.25E-07	3.47E-06
		96	0.9919	0.0271	6.73E-04	1.92E-03	4.53E-07	3.69E-06	0.9885	0.0343	2.58E-03	6.11E-03	6.67E-06	3.74E-05
0.25	Random	32	0.9283	0.2152	5.51E-03	1.75E-02	3.04E-05	3.06E-04	0.9842	0.0572	2.07E-03	9.27E-03	4.27E-06	8.59E-05
		64	0.9353	0.2007	6.82E-03	1.93E-02	4.65E-05	3.71E-04	0.9850	0.0560	1.98E-03	7.03E-03	3.90E-06	4.94E-05
		96	0.9256	0.2262	3.31E-02	9.73E-02	1.10E-03	9.46E-03	0.9830	0.0661	6.58E-03	4.28E-02	4.32E-05	1.83E-03
	Uniform	32	0.9349	0.1987	5.26E-03	1.33E-02	2.77E-05	1.76E-04	0.9848	0.0528	3.11E-03	1.00E-02	9.70E-06	9.99E-05
		64	0.9257	0.2183	2.14E-02	5.62E-02	4.57E-04	3.16E-03	0.9836	0.0640	5.63E-03	2.49E-02	3.17E-05	6.21E-04
		96	0.9332	0.1996	8.31E-03	2.06E-02	6.90E-05	4.25E-04	0.9858	0.0526	1.14E-03	5.23E-03	1.30E-06	2.73E-05
0.5	Random	32	0.7707	0.6737	4.48E-02	1.20E-01	2.01E-03	1.45E-02	0.9723	0.1797	1.01E-02	9.99E-02	1.02E-04	9.97E-03
		64	0.7755	0.6253	3.14E-02	7.01E-02	9.89E-04	4.91E-03	0.9796	0.1269	2.57E-03	3.32E-02	6.63E-06	1.10E-03
		96	0.7937	0.6282	3.03E-02	8.03E-02	9.20E-04	6.44E-03	0.9766	0.1492	9.27E-03	8.14E-02	8.59E-05	6.62E-03
	Uniform	32	0.7722	0.6498	2.72E-02	7.68E-02	7.40E-04	5.90E-03	0.9767	0.1492	4.41E-03	6.44E-02	1.94E-05	4.15E-03
		64	0.7897	0.6224	2.60E-02	5.21E-02	6.74E-04	2.71E-03	0.9759	0.1434	4.25E-03	3.70E-02	1.81E-05	1.37E-03
		96	0.7839	0.6191	1.59E-02	3.16E-02	2.52E-04	9.95E-04	0.9775	0.1434	5.50E-03	4.58E-02	3.03E-05	2.10E-03

Table 4.45: The following results are the averages of 10 trials for each test using MNIST as the dataset. Each test we run have a different dropout, $D = [0, 0.25, 0.5]$, and the number of filters, $F = [32, 64, 96]$. Each row provides the accuracy, loss, standard deviation, and variance for both training and validation results. Note: Uniform is the term we are using to describe Ozbulak's results in [16]. The results are from using a non-shuffled dataset instead of being shuffled like experiment C (Section 4.3). The first layer in the CNN is also untrainable to see how well the Gabor filter bank can extract features without learning.



(a)



(b)

Figure 4.39: Here we see a different representation of Table 4.45 showing the mean, maximum and minimum values as well as the mean of the validation accuracy. Within the legends are each permutation of the experiment using the non-shuffled MNIST dataset in both training and validation sets. The first layer within the network is frozen as well so it cannot learn during training.

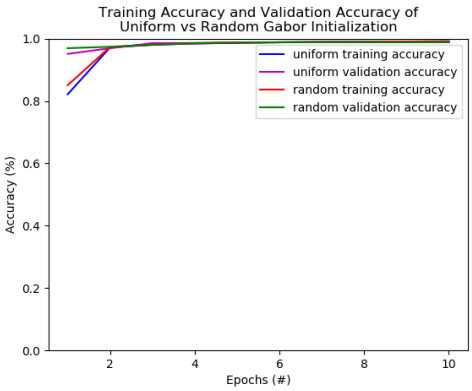
Discussion

We can see that when we add dropout to the network, the performance drops (Figure 4.40). This follows what we have found in previous experiments. Another pattern that we see is that although dropout drops the performance of the network, when we train on MNIST, the performance drop is very minimal (< 1% difference). We see this pattern for all permutations in this experiment for MNIST. This could be that MNIST is a very simple dataset, that being grayscale, 1 channel of data, and a set of 10 classes.

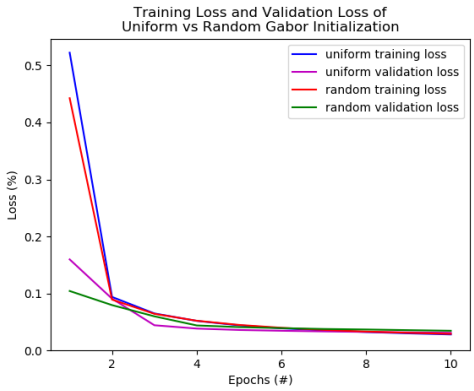
Although not noticeable at first, in Figure 4.39, we can see that there is a pattern where we increase the number of filters in the first layer for the Gabor filter, we increase the number if different randomly initialized Gabor filters which increases the performance. Albeit, a minimal gain when training on MNIST. We can see this for all initialization methods (random versus grid-search) and for all levels of dropout.

By comparing the validation accuracy from each trial using either random or grid-search initialization, we see that 6 times random initialization provides a higher result than grid-search. This follows the pattern we have seen in previous experiments.

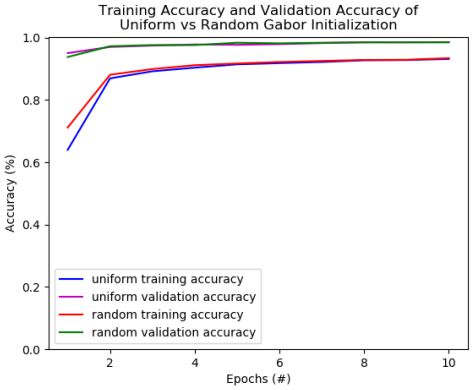
In summary, adding dropout offers no benefit to training the dataset. We see that the training converges very quickly and no overfitting is found in all cases of dropout (Figure 4.40). Adding more filters in the first layer for the Gabor filter gives a small performance boost, but also adds more complexity/parameters to the network. Using random initialization for the Gabor filter is likely faster than using a grid-search as in our case, out of 9 tests, we see that random initialization for the Gabor filter in the first layer performs better.



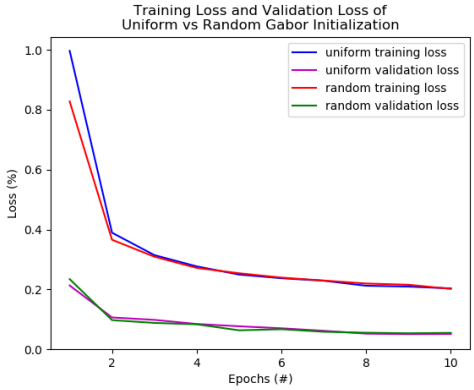
(a) Accuracy: $D = 0$, $F = 32$



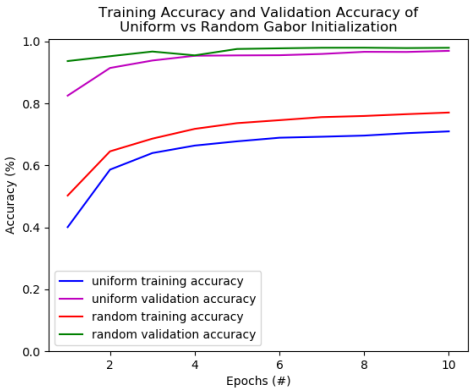
(b) Loss: $D = 0$, $F = 32$



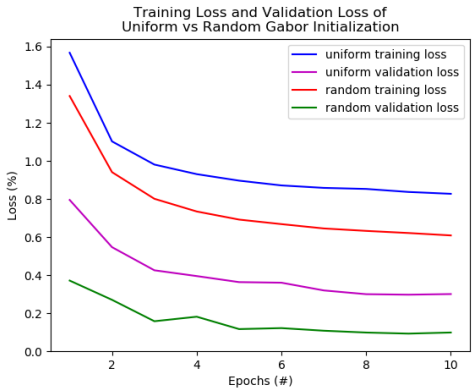
(c) Accuracy: $D = 0.25$, $F = 32$



(d) Loss: $D = 0.25$, $F = 32$



(e) Accuracy: $D = 0.50$, $F = 32$



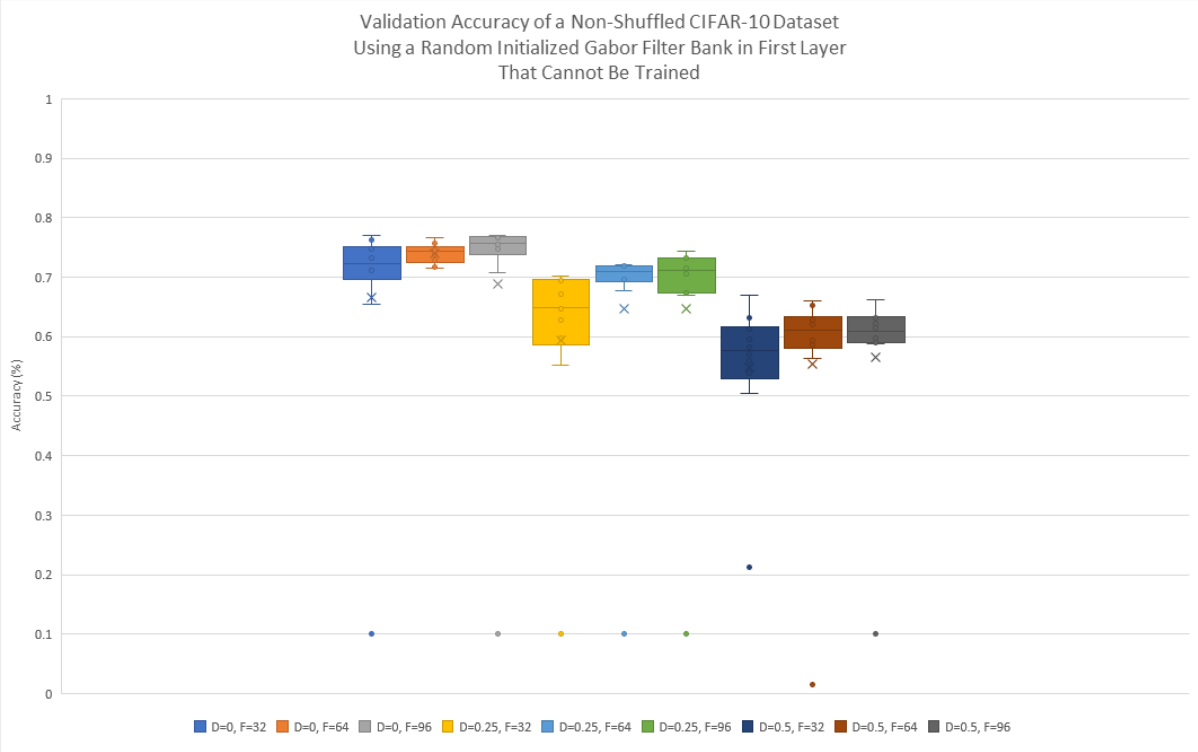
(f) Loss: $D = 0.50$, $F = 32$

Figure 4.40: Training on the MNIST dataset shows that the training converges very fast resulting a high accuracy and low loss. Adding dropout to the network shows a drop in performance in all cases which continues what we have found in previous experiments. Using a grid-search initialization for the Gabor filter bank compared to being initialized with a random distribution in the Gabor filter bank shows no significant boost in performance. In most cases, randomization outperforms grid-search, but the difference is insignificant.

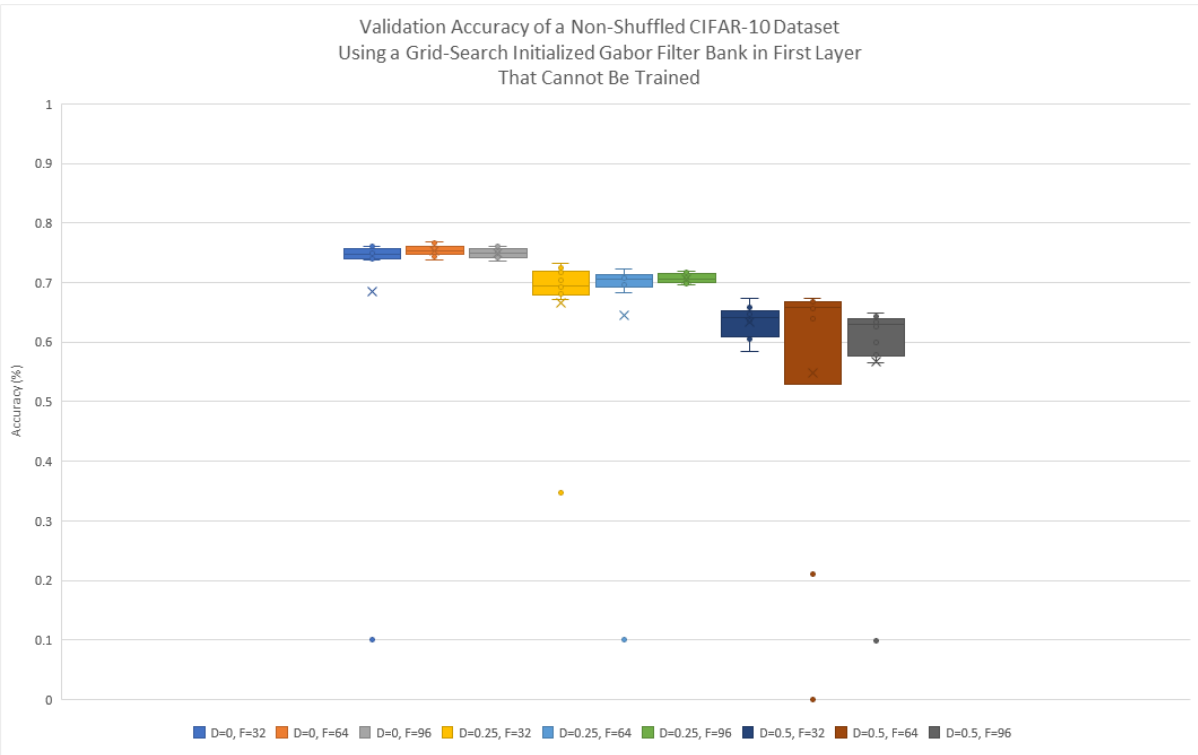
CIFAR-10

MNIST														
Dropout	**Test Type**	**# of Filters**	**Training**						**Validation**					
			Acc	**Loss**	**Std. Dev Acc**	**Std. Dev Loss**	**Var Acc**	**Var Loss**	**Acc**	**Loss**	**Std. Dev Acc**	**Std. Dev Loss**	**Var Acc**	**Var Loss**
0	Random	32	0.9005	0.2768	2.83E-01	7.13E-01	8.00E-02	5.08E-01	0.6653	1.7172	2.01E-01	2.90E-01	4.06E-02	8.43E-02
		64	0.9965	0.0239	5.87E-03	2.69E-02	3.44E-05	7.26E-04	0.7402	1.7238	1.66E-02	1.18E-01	2.76E-04	1.40E-02
		96	0.9015	0.2594	2.84E-01	7.22E-01	8.07E-02	5.21E-01	0.6886	1.7602	2.08E-01	2.69E-01	4.31E-02	7.26E-02
	Uniform	32	0.9093	0.2398	2.86E-01	7.25E-01	8.16E-02	5.25E-01	0.6851	1.8035	2.06E-01	1.93E-01	4.23E-02	3.72E-02
		64	1.0000	0.0061	1.84E-05	1.14E-03	3.38E-10	1.29E-06	0.7540	1.7919	9.41E-03	8.47E-02	8.86E-05	7.18E-03
		96	1.0000	0.0064	4.99E-05	1.41E-03	2.49E-09	1.98E-06	0.7497	1.8189	8.09E-03	7.47E-02	6.55E-05	5.58E-03
0.25	Random	32	0.6524	0.9284	2.14E-01	5.36E-01	4.59E-02	2.88E-01	0.5945	1.1841	1.80E-01	4.04E-01	3.25E-02	1.63E-01
		64	0.7451	0.6859	2.30E-01	5.72E-01	5.29E-02	3.27E-01	0.6474	1.1339	1.93E-01	4.17E-01	3.72E-02	1.74E-01
		96	0.7522	0.6687	2.35E-01	5.86E-01	5.50E-02	3.43E-01	0.6474	1.1589	1.94E-01	4.14E-01	3.77E-02	1.72E-01
	Uniform	32	0.7625	0.6470	1.69E-01	4.25E-01	2.86E-02	1.81E-01	0.6661	1.0967	1.14E-01	2.44E-01	1.30E-02	5.97E-02
		64	0.7455	0.6845	2.33E-01	5.80E-01	5.41E-02	3.36E-01	0.6456	1.1575	1.92E-01	4.05E-01	3.69E-02	1.64E-01
		96	0.8323	0.4668	1.33E-02	3.18E-02	1.78E-04	1.01E-03	0.7067	1.0657	7.94E-03	5.08E-02	6.30E-05	2.58E-03
0.5	Random	32	0.4795	1.3364	1.21E-01	2.88E-01	1.47E-02	8.27E-02	0.5477	1.2954	1.27E-01	2.49E-01	1.61E-02	6.20E-02
		64	0.5591	1.1468	4.86E-02	1.17E-01	2.36E-03	1.38E-02	0.6195	1.1563	3.34E-02	8.43E-02	1.12E-03	7.11E-03
		96	0.4839	1.3418	1.38E-01	3.44E-01	1.92E-02	1.18E-01	0.5647	1.2416	1.65E-01	3.75E-01	2.72E-02	1.41E-01
	Uniform	32	0.5783	1.0977	4.09E-02	9.78E-02	1.67E-03	9.56E-03	0.6345	1.1340	2.67E-02	7.78E-02	7.15E-04	6.05E-03
		64	0.5488	1.1850	1.48E-01	3.66E-01	2.18E-02	1.34E-01	0.6116	1.1824	1.42E-01	3.09E-01	2.01E-02	9.54E-02
		96	0.4835	1.3238	1.42E-01	3.57E-01	2.03E-02	1.28E-01	0.5674	1.2428	1.67E-01	3.78E-01	2.78E-02	1.43E-01

Table 4.46: The following results are the averages of 10 trials for each test using CIFAR-10 as the dataset. Each test we run have a different dropout, $D = [0, 0.25, 0.5]$, and the number of filters, $F = [32, 64, 96]$. Each row provides the accuracy, loss, standard deviation, and variance for both training and validation results. Note: Uniform is the term we are using to describe Ozbulak's results in [16]. The results are from using a non-shuffled dataset instead of being shuffled like experiment C (Section 4.3). The first layer in the CNN is also untrainable to see how well the Gabor filter bank can extract features without learning.



(a)



(b)

Figure 4.41: Here we see a different representation of Table 4.46 showing the mean, maximum and minimum values as well as the mean of the validation accuracy. Within the legends are each permutation of the experiment using the non-shuffled CIFAR-10 dataset in both training and validation sets. The first layer within the network is frozen as well so it cannot learn during training.

Discussion

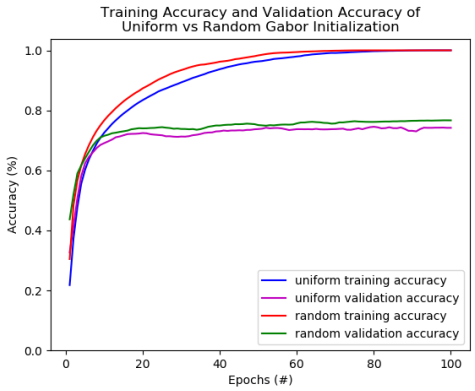
The results in Table 4.46 show a similar pattern found from previous experiments (C and D) which show that as we increase the dropout (in all permutations of this experiment) we see a drop in performance. For example, the dropout found in the test using a grid-search initialization for the Gabor filter with dropout and the number of filters, $D = [0, 0.5]$, and $F = 96$, we see a difference of 12.4% validation accuracy. This shows that there is no benefit in adding dropout to the network when training on CIFAR-10. We also see that when using our architecture (Section 3.3), we see that training on CIFAR-10 results in overfitting in with dropout layers of 25% and 0%. (shown in Figure 4.42).

As we increase the number of filters, we see an increase in performance when we train on CIFAR-10. This happens in all levels of dropout and for both initialization methods for the Gabor filter used in layer 1. This continues the pattern found from previous experiments and tests (Sections 4.3 and 4.4). The difference in performance varies more than the MNIST results in this experiment. This could be the result of training a more complex dataset in comparison to MNIST.

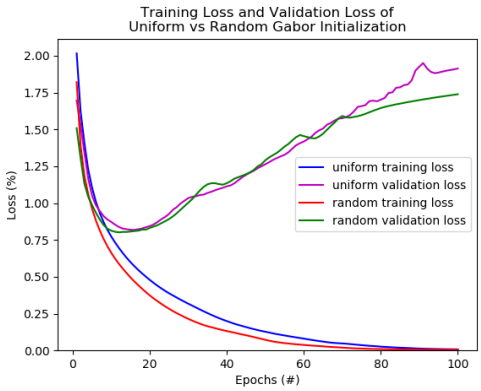
Training on the CIFAR-10 dataset, we see a similar pattern once again found from previous experiments where using a grid-search initialization for the Gabor filter is more preferred over random initialization. In this test, we see that 7 out of 9 tests, the validation accuracy favours a grid-search Gabor filter bank. This is consistent in our findings and shows here in this test once again. This is likely due to the complexity of the dataset, that of which having 2 additional channels over the MNIST dataset. Not only does CIFAR-10 favour a grid-search Gabor filter bank, it also performs (on average) better than using the random initialization. The average results appear to be around 5% higher in terms of validation accuracy. The loss is much lower when using a grid-search as well.

In summary, adding dropout adds no benefit to the system because it decreases the performance of the network. CIFAR-10 overfits unless we add a dropout layer of at least 50% when training with 100 epochs. Increasing the number of filters when training on CIFAR-10

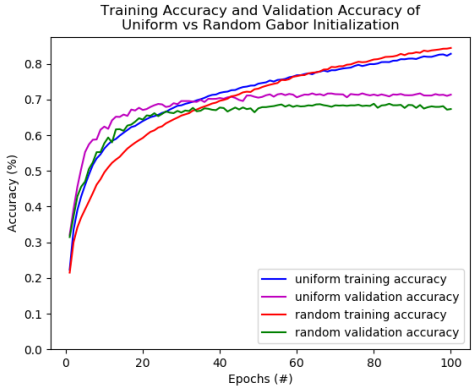
performs better than having less number of filters. Using the grid-search initialization for the Gabor filter bank performs, on average, higher than using a random initialization method for the Gabor filter bank. CIFAR-10 also favours using the grid-search method.



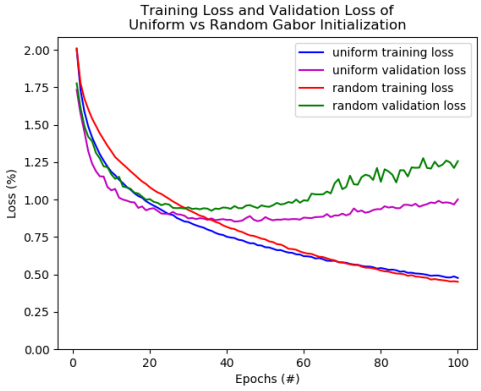
(a) Accuracy: $D = 0$, $F = 96$



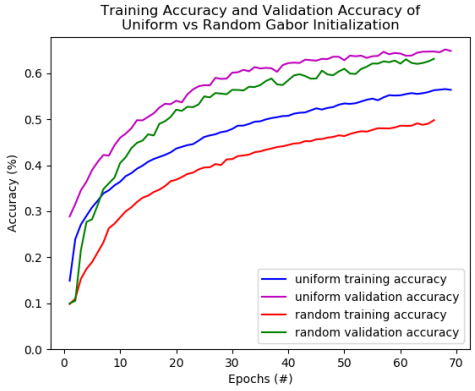
(b) Loss: $D = 0$, $F = 96$



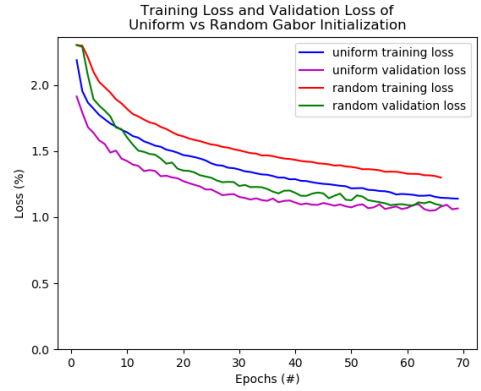
(c) Accuracy: $D = 0.25$, $F = 96$



(d) Loss: $D = 0.25$, $F = 96$



(e) Accuracy: $D = 0.50$, $F = 96$



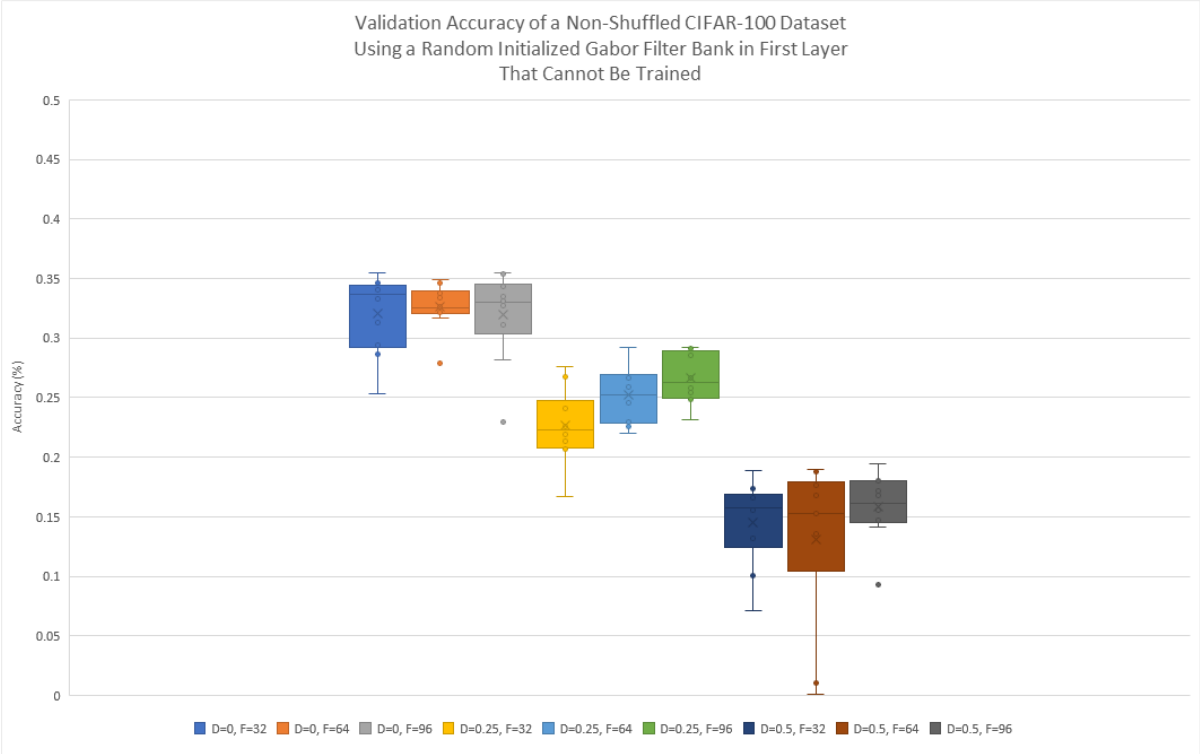
(f) Loss: $D = 0.50$, $F = 96$

Figure 4.42: Training on the CIFAR-10 dataset results in overfitting if we do not add a dropout, where $D = 0.50$. The overfitting occurs around epoch, $E = 10$, which is much lower than previous experiments. The performance of the network drops with an increased dropout which is similar to what we have found from previous experiments. Using a grid-search initialization for the Gabor filter bank improves the performance of training and testing for CIFAR-10 when compared to being initialized with a random distribution in the Gabor filter bank.

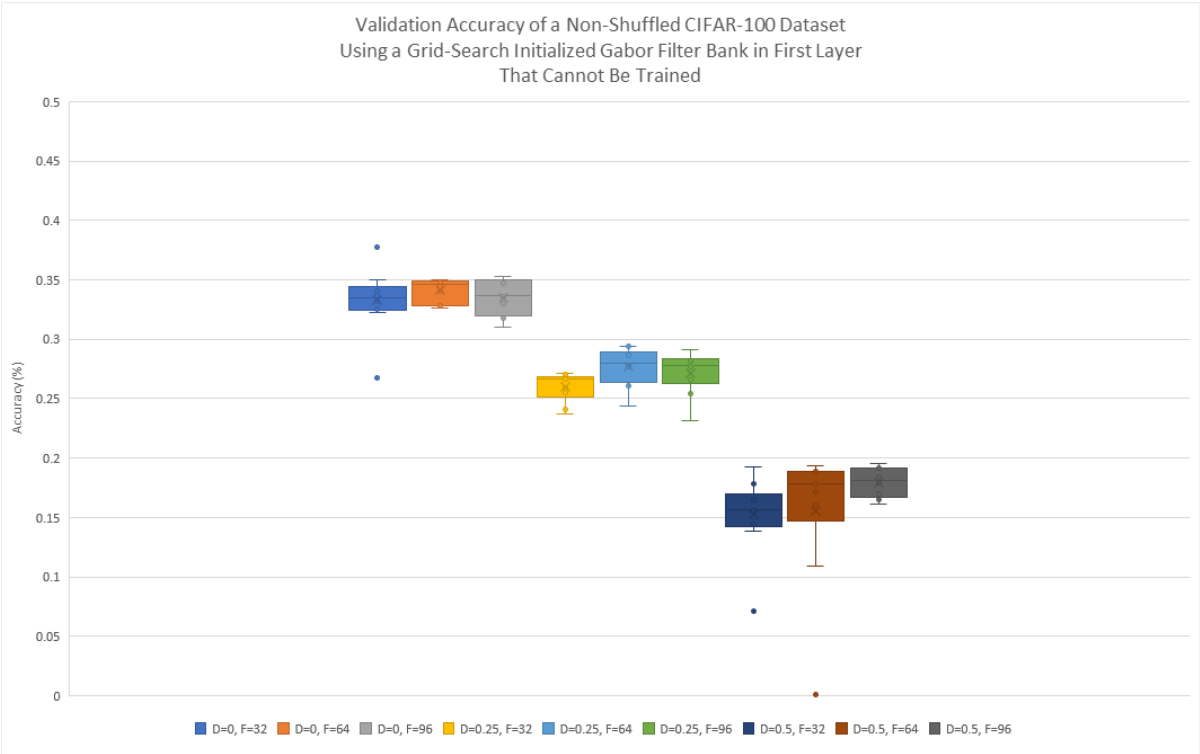
CIFAR-100

MNIST														
Dropout	**Test Type**	**# of Filters**	**Training**						**Validation**					
			Acc	**Loss**	**Std. Dev Acc**	**Std. Dev Loss**	**Var Acc**	**Var Loss**	**Acc**	**Loss**	**Std. Dev Acc**	**Std. Dev Loss**	**Var Acc**	**Var Loss**
0	Random	32	0.4929	1.7824	8.75E-02	3.91E-01	7.65E-03	1.53E-01	0.3209	2.8907	3.33E-02	1.88E-01	1.11E-03	3.54E-02
		64	0.6111	1.2945	4.11E-02	1.41E-01	1.69E-03	1.98E-02	0.3260	3.2319	1.98E-02	1.83E-01	3.93E-04	3.35E-02
		96	0.6483	1.1443	3.22E-02	9.40E-02	1.04E-03	8.84E-03	0.3195	3.5552	3.82E-02	5.52E-01	1.46E-03	3.05E-01
	Uniform	32	0.6010	1.3345	2.82E-02	9.72E-02	7.94E-04	9.45E-03	0.3328	3.1162	2.75E-02	2.96E-01	7.57E-04	8.74E-02
		64	0.6636	1.1208	1.59E-02	5.17E-02	2.54E-04	2.67E-03	0.3411	3.2862	9.67E-03	1.65E-01	9.35E-05	2.72E-02
		96	0.6689	1.0950	1.35E-02	4.15E-02	1.81E-04	1.72E-03	0.3353	3.4237	1.51E-02	1.90E-01	2.28E-04	3.62E-02
0.25	Random	32	0.1545	3.3734	2.32E-02	1.55E-01	5.40E-04	2.41E-02	0.2266	3.1499	3.17E-02	1.60E-01	1.00E-03	2.55E-02
		64	0.1773	3.2184	1.92E-02	1.09E-01	3.69E-04	1.18E-02	0.2523	3.0114	2.30E-02	1.13E-01	5.29E-04	1.28E-02
		96	0.1906	3.1179	1.94E-02	1.24E-01	3.78E-04	1.54E-02	0.2667	2.9358	2.16E-02	1.04E-01	4.66E-04	1.09E-02
	Uniform	32	0.1853	3.1597	1.25E-02	6.57E-02	1.57E-04	4.31E-03	0.2603	2.9585	1.24E-02	5.68E-02	1.54E-04	3.23E-03
		64	0.2069	3.0178	1.72E-02	1.02E-01	2.96E-04	1.04E-02	0.2768	2.8644	1.62E-02	7.60E-02	2.61E-04	5.78E-03
		96	0.1973	3.0629	1.56E-02	9.17E-02	2.45E-04	8.40E-03	0.2716	2.9059	1.78E-02	8.54E-02	3.16E-04	7.29E-03
0.5	Random	32	0.0727	3.9313	1.61E-02	1.43E-01	2.61E-04	2.03E-02	0.1450	3.6636	3.61E-02	1.90E-01	1.30E-03	3.59E-02
		64	0.0757	3.8917	2.50E-02	2.67E-01	6.25E-04	7.12E-02	0.1496	3.6332	5.28E-02	3.59E-01	2.79E-03	1.29E-01
		96	0.0812	3.8213	1.58E-02	1.47E-01	2.50E-04	2.15E-02	0.1582	3.5549	2.88E-02	1.83E-01	8.28E-04	3.35E-02
	Uniform	32	0.0758	3.8776	1.33E-02	1.22E-01	1.76E-04	1.49E-02	0.1527	3.5891	3.25E-02	1.73E-01	1.06E-03	2.99E-02
		64	0.0867	3.7625	1.47E-02	1.30E-01	2.16E-04	1.68E-02	0.1687	3.4769	2.71E-02	1.48E-01	7.37E-04	2.19E-02
		96	0.0902	3.7356	8.04E-03	6.35E-02	6.46E-05	4.03E-03	0.1789	3.4365	1.22E-02	6.22E-02	1.49E-04	3.87E-03

Table 4.47: The following results are the averages of 10 trials for each test using CIFAR-100 as the dataset. Each test we run have a different dropout, $D = [0, 0.25, 0.5]$, and the number of filters, $F = [32, 64, 96]$. Each row provides the accuracy, loss, standard deviation, and variance for both training and validation results. Note: Uniform is the term we are using to describe Ozbulak's results in [16]. The results are from using a non-shuffled dataset instead of being shuffled like experiment C (Section 4.3). The first layer in the CNN is also untrainable to see how well the Gabor filter bank can extract features without learning.



(a)



(b)

Figure 4.43: Here we see a different representation of Table 4.47 showing the mean, maximum and minimum values as well as the mean of the validation accuracy. Within the legends are each permutation of the experiment using the non-shuffled CIFAR-10 dataset in both training and validation sets. The first layer within the network is frozen as well so it cannot learn during training.

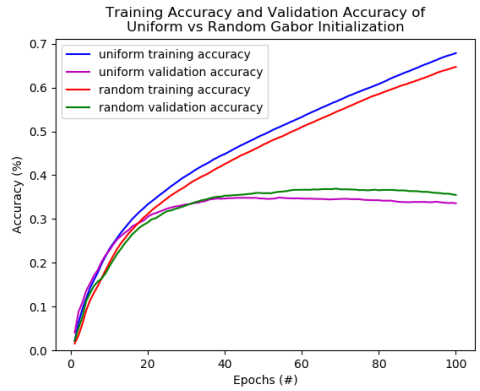
Discussion

Like the previous tests in this experiment, we see a drop in performance when we add a dropout layer when training on CIFAR-100. This is a continuous trend that we see from previous experiments and training on CIFAR-100 with a frozen first layer follows suit. We also see that by adding a 50% dropout layer to the network, the performance is almost halved in terms of validation accuracy. There is no added benefit for having dropout included in our architecture. Both initialization methods for the Gabor filter show this pattern as well. With no dropout layer added, we see that the network overfits (Figure 4.44).

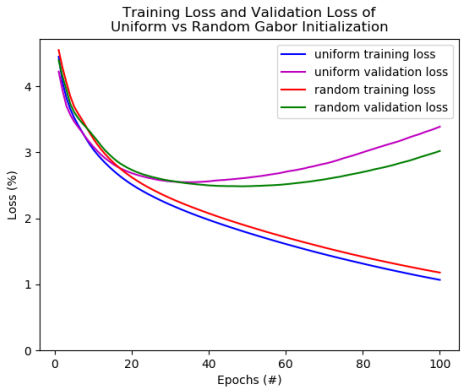
Increasing the number of filters helps improve the performance for all tests in this experiment. The performance boost seems to be very little (for example, when looking at a grid-search initialization method for the Gabor filter using a dropout, $D = 0.25$, we see an increase in performance by 1.13% in terms of validation accuracy when comparing 2 different number of filters, $F = [32, 96]$. This follows the trend seen in previous tests and experiments.

When training on CIFAR-100, we see that with our network, it favours using the grid-search initialization method for the Gabor filter bank rather than the random initialization method. In all of the 9 different comparisons for the initialization methods, we see that all 9 cases favour the grid-search in terms of validation accuracy. The performance of using a grid-search also boosts the performance over using a random initialization by roughly 1%-2% in terms of validation accuracy.

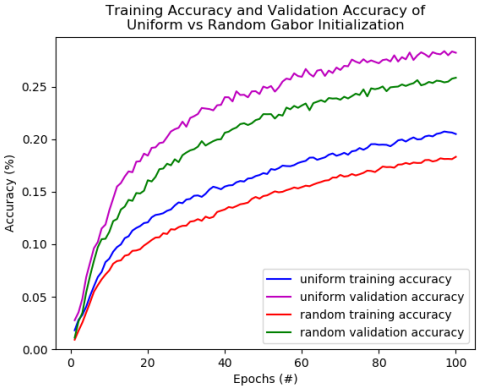
In summary, adding dropout adds no benefit to the network. Although the network requires dropout to overcome overfitting, with dropout the performance of the network drops. Adding more Gabor filters in the first layer boosts performance but the increase is not very high. When we initialize the Gabor filter bank with a random distribution versus a grid-search distribution, CIFAR-100 does better and outperforms random initialization when using a grid-search for the Gabor filter bank.



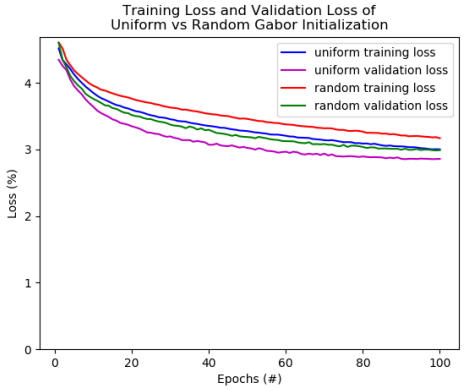
(a) Accuracy: $D = 0$, $F = 96$



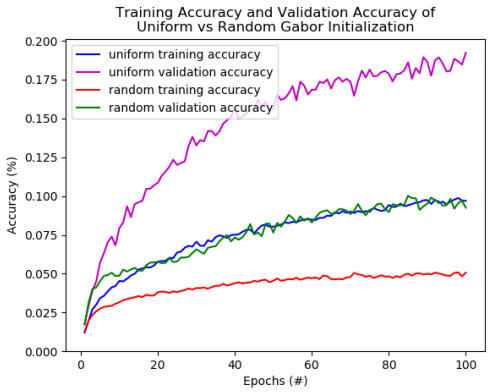
(b) Loss: $D = 0$, $F = 96$



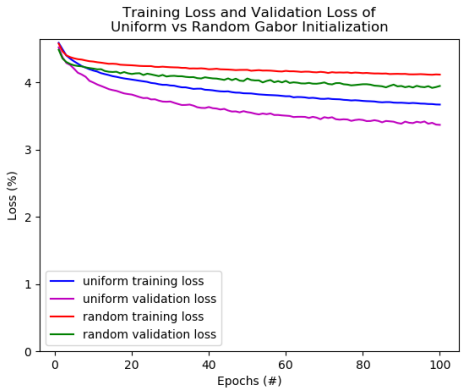
(c) Accuracy: $D = 0.25$, $F = 96$



(d) Loss: $D = 0.25$, $F = 96$



(e) Accuracy: $D = 0.50$, $F = 96$



(f) Loss: $D = 0.50$, $F = 96$

Figure 4.44: Training on the CIFAR-100 dataset results in overfitting if we do not add a dropout. The overfitting occurs around epoch, $E = 10$, when dropout, $D = 0$. The performance of the network drops with an increased dropout which is similar to what we have found from previous experiments. Using a grid-search initialization for the Gabor filter bank improves the performance of training and testing for CIFAR-100 when compared to being initialized with a random distribution in the Gabor filter bank.

4.5.3 Summary

[Confirm that dropout adds no benefits] In terms of dropout for this experiment, we see the same trend we have seen in previous experiments where adding dropout (although in some cases reduces and minimizes overfitting) reduces the performance of the network in terms of validation accuracy. Adding dropout to overcome overfitting (in CIFAR-10 and in some cases CIFAR-100) isn't necessary because of the reduction in performance. The network eventually converges to a lower accuracy than a network without dropout. This is because when overfitting occurs, it occurs at an earlier epoch with a higher accuracy using less dropout. As we add more dropout, the point of overfitting is extended, but the network achieves a lower accuracy overall.

[Confirm that increasing filters adds performance] Just like experiment C and D, we see the same result for filters: adding more filters improves the performance of the network. For the MNIST dataset and the CIFAR-100 dataset, the performance boost is very small. For the CIFAR-10 dataset, the boost can be large or small (requires more testing).

[Confirm MNIST →random GCNN, CIFAR-10 and CIFAR-100 →grid-search GCNN] The same pattern shows in this experiment where if we train on MNIST, we see that it is more likely to achieve a better performing network when initialized with a random distribution for the Gabor filter bank in the first layer. When training with CIFAR-10 and CIFAR-100, we see the opposite happening where it prefers using a grid-search initialization instead for the Gabor filter bank. Using either initialization method for MNIST shows similar performances. Using grid-search initialization for CIFAR-10 and CIFAR-100 however, results in a boost on average in terms of validation accuracy over using random initialization.

[Confirm that non-shuffled datasets slightly drop performance]

[New: Freezing the Gabor filter bank drops performance but still converges] Freezing the first layer (the Gabor filter bank) shows only a small drop in performance when compared to giving the network the ability to learn in the first layer. The Gabor filter when initialized via random or a grid-search distribution is just as effective Gabor filters that are able to learn through training. We can see these small differences in Tables 4.48 to 4.51. The performance drop in terms of

validation accuracy and training accuracy are insignificant for MNIST, but for CIFAR-10 and CIFAR-100, the difference is more noticeable.

Random Search										
Dropout	# of Filters	Training Accuracy								
		Shuffled			Non-Shuffled			Frozen First Layer, Non-Shuffled		
		MNIST	CIFAR-10	CIFAR-100	MNIST	CIFAR-10	CIFAR-100	MNIST	CIFAR-10	CIFAR-100
0	32	0.9932	1.0000	0.6315	0.9931	0.9096	0.6722	0.9914	0.9005	0.4929
	64	0.9934	1.0000	0.7852	0.9936	1.0000	0.7062	0.9922	0.9965	0.6111
	96	0.9937	0.7292	0.6407	0.9938	1.0000	0.7641	0.9931	0.9015	0.6483
0.25	32	0.9342	0.6458	0.1921	0.9363	0.8269	0.1940	0.9283	0.9093	0.6010
	64	0.9414	0.6321	0.2089	0.9379	0.7504	0.2033	0.9353	1.0000	0.6636
	96	0.9339	0.8774	0.2059	0.9350	0.6753	0.1955	0.9256	1.0000	0.6689
0.50	32	0.7867	0.4254	0.0841	0.7730	0.5554	0.0834	0.7707	0.6524	0.1545
	64	0.7966	0.4067	0.0908	0.7915	0.4113	0.0796	0.7755	0.7451	0.1773
	96	0.7838	0.5654	0.0800	0.8007	0.3937	0.0908	0.7937	0.7522	0.1906

Table 4.48: Using a random initialized Gabor filter bank in the first layer, we compare the results from using a shuffled network, a non-shuffled network, and a non-shuffled untrainable, first layer network in terms of training accuracy.

Grid Search										
Dropout	# of Filters	Training Accuracy								
		Shuffled			Non-Shuffled			Frozen First Layer, Non-Shuffled		
		MNIST	CIFAR-10	CIFAR-100	MNIST	CIFAR-10	CIFAR-100	MNIST	CIFAR-10	CIFAR-100
0	32	0.9925	1.0000	0.7485	0.9926	1.0000	0.7064	0.9909	0.9093	0.6010
	64	0.9933	1.0000	0.8104	0.9930	1.0000	0.7662	0.9911	1.0000	0.6636
	96	0.9936	0.9852	0.8028	0.9933	1.0000	0.7649	0.9919	1.0000	0.6689
0.25	32	0.9432	0.7696	0.1953	0.9399	0.8297	0.2107	0.9349	0.7625	0.1853
	64	0.9406	0.8309	0.2275	0.9411	0.8734	0.2309	0.9257	0.7455	0.2069
	96	0.9382	0.8596	0.2251	0.9434	0.8082	0.2137	0.9332	0.8323	0.1973
0.50	32	0.7851	0.5588	0.0891	0.7891	0.4917	0.0859	0.7722	0.5783	0.0758
	64	0.7935	0.6507	0.0978	0.7297	0.6177	0.0924	0.7897	0.5488	0.0867
	96	0.7544	0.4956	0.0986	0.7966	0.4466	0.0911	0.7839	0.4835	0.0902

Table 4.49: Using a grid-search initialized Gabor filter bank in the first layer, we compare the results from using a shuffled network, a non-shuffled network, and a non-shuffled untrainable, first layer network in terms of training accuracy.

Random Search										
Dropout	# of Filters	Validation Accuracy								
		Shuffled			Non-Shuffled			Frozen First Layer, Non-Shuffled		
		MNIST	CIFAR-10	CIFAR-100	MNIST	CIFAR-10	CIFAR-100	MNIST	CIFAR-10	CIFAR-100
0	32	0.9922	0.7758	0.3625	0.9905	0.7046	0.3375	0.9890	0.6653	0.3209
	64	0.9918	0.7805	0.3481	0.9907	0.7767	0.3289	0.9900	0.7402	0.3260
	96	0.9923	0.5736	0.3123	0.9913	0.7771	0.3128	0.9900	0.6886	0.3195
0.25	32	0.9880	0.5686	0.2669	0.9885	0.7199	0.2710	0.9842	0.5945	0.2266
	64	0.9881	0.5436	0.2799	0.9873	0.6556	0.2687	0.9850	0.6474	0.2523
	96	0.9883	0.7334	0.2789	0.9875	0.5764	0.2630	0.9830	0.6474	0.2667
0.50	32	0.9804	0.4738	0.1654	0.9756	0.5956	0.1629	0.9723	0.5477	0.1450
	64	0.9800	0.4365	0.1738	0.9817	0.4425	0.1529	0.9796	0.6195	0.1496
	96	0.9798	0.5648	0.1558	0.9824	0.4048	0.1704	0.9766	0.5647	0.1582

Table 4.50: Using a random initialized Gabor filter bank in the first layer, we compare the results from using a shuffled network, a non-shuffled network, and a non-shuffled untrainable, first layer network in terms of validation accuracy.

Grid Search										
Dropout	# of Filters	Validation Accuracy								
		Shuffled			Non-Shuffled			Frozen First Layer, Non-Shuffled		
		MNIST	CIFAR-10	CIFAR-100	MNIST	CIFAR-10	CIFAR-100	MNIST	CIFAR-10	CIFAR-100
0	32	0.9909	0.7782	0.3464	0.9899	0.7740	0.3192	0.9887	0.6851	0.3328
	64	0.9916	0.7829	0.3596	0.9899	0.7758	0.3232	0.9885	0.7540	0.3411
	96	0.9922	0.7644	0.3485	0.9906	0.7753	0.3150	0.9885	0.7497	0.3353
0.25	32	0.9874	0.6683	0.2725	0.9874	0.7127	0.2864	0.9848	0.6661	0.2603
	64	0.9872	0.7221	0.2979	0.9877	0.7348	0.2906	0.9836	0.6456	0.2768
	96	0.9887	0.7295	0.2935	0.9878	0.6869	0.2835	0.9858	0.7067	0.2716
0.50	32	0.9801	0.6112	0.1714	0.9815	0.5324	0.1685	0.9767	0.6345	0.1527
	64	0.9826	0.6837	0.1948	0.8925	0.6549	0.1784	0.9759	0.6116	0.1687
	96	0.9595	0.5700	0.1914	0.9803	0.4804	0.1811	0.9775	0.5674	0.1789

Table 4.51: Using a grid-search initialized Gabor filter bank in the first layer, we compare the results from using a shuffled network, a non-shuffled network, and a non-shuffled untrainable, first layer network in terms of validation accuracy.

4.6 F: Grid Search Vs. Random Search with a Simple Rock Database

4.6.1 Experiment Setup

In this experiment, we take our Gabor filter bank initialization methods and input them within another network created by Pascual [17]. We will be supplying the Gabor filter bank within the first layer following our own architecture. His architecture is described in Section 3.3.2 and a visual representation of the architecture is in Figure 3.6. The main purpose of this experiment is to see how our Gabor filter bank created in the first layer reacts to a custom dataset. This dataset was also described in Section 3.1.4. It is a simple dataset consisting of 9 rock classes, and the images collected are in colour. This will add complexity just like the CIFAR-10 dataset, except all of the classes are a type of rocks rather than different objects (such as cars, food, etc.).

Just like the other experiments from C through E (Sections 4.3, 4.4, 4.5), we will use a permutation of dropouts and a change in the number of filters, $D = [0, 0.25, 0.5]$, and $F = [32, 64, 96, 128]$ respectively. We will use 128 filters in the first layer only for a more direct comparison with Pascual's results obtained in his simple CNN. The only difference when comparing our results to his is that the first layer will be generated based on a Gabor filter random initialization or a grid-search initialization. We can include our findings of dropout and tuning the number of filters within the first layer and compare these results as well.

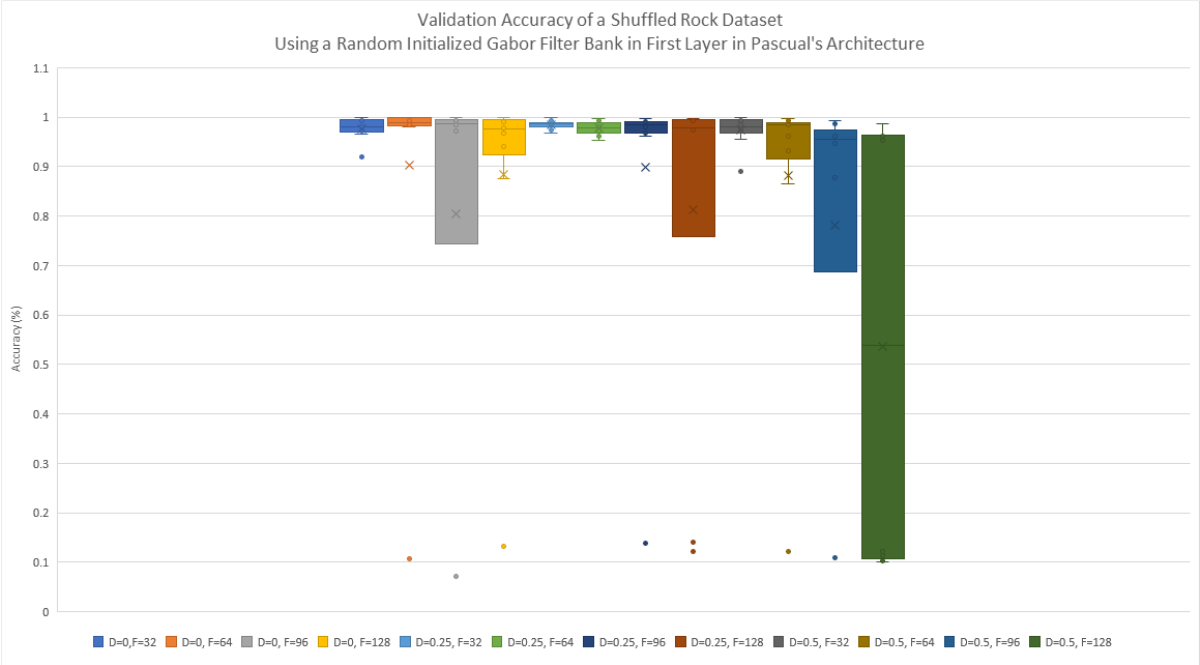
We will explore and solidify our findings from previous experiments for using a shuffled dataset, a non-shuffled dataset, and using an non-trainable first layer along with a non-shuffled dataset. These results will be listed in the following subsections.

4.6.2 Results

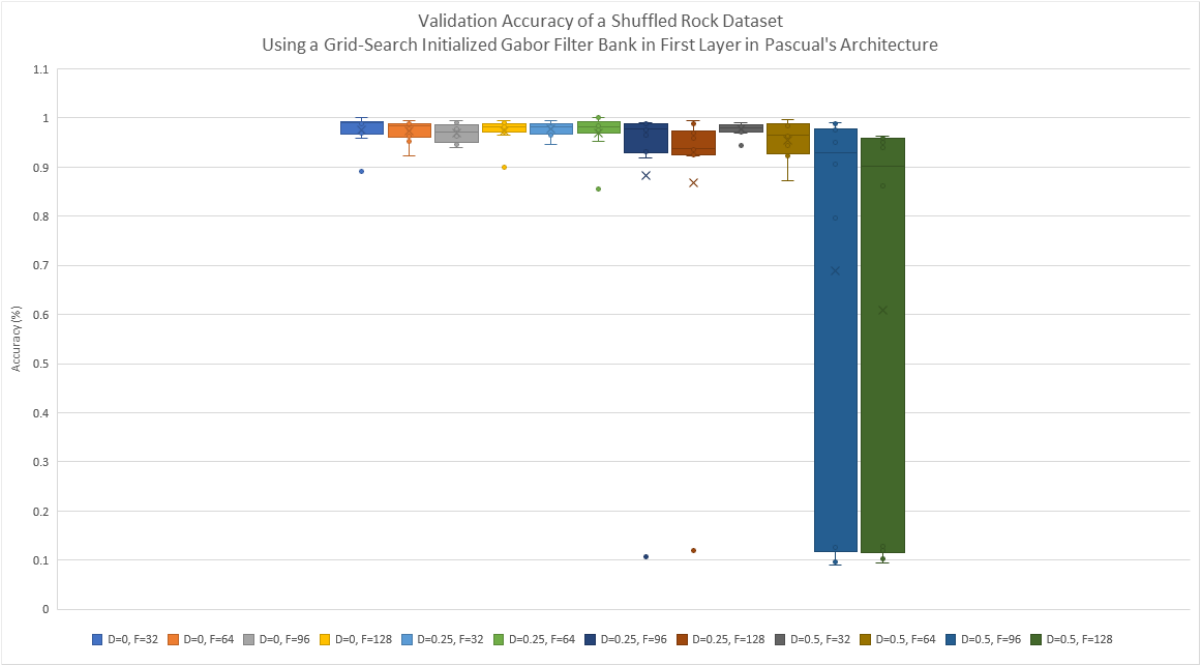
Shuffled Dataset

AlexisNet-9														
Dropout	Test Type	# of Filters	Training						Validation					
			Acc	Loss	Std. Dev Acc	Std. Dev Loss	Var Acc	Var Loss	Acc	Loss	Std. Dev Acc	Std. Dev Loss	Var Acc	Var Loss
0	Random	32	0.9891	0.0369	1.32E-02	4.53E-02	1.75E-04	2.05E-03	0.9772	0.0731	2.38E-02	8.28E-02	5.64E-04	6.85E-03
		64	0.9040	0.2470	2.73E-01	6.84E-01	7.46E-02	4.67E-01	0.9031	0.2449	2.80E-01	6.93E-01	7.84E-02	4.80E-01
		96	0.8203	0.4562	3.64E-01	9.15E-01	1.32E-01	8.37E-01	0.8041	0.4734	3.84E-01	9.15E-01	1.48E-01	8.36E-01
		128	0.9001	0.2557	2.76E-01	6.82E-01	7.59E-02	4.65E-01	0.8850	0.3174	2.67E-01	6.73E-01	7.15E-02	4.53E-01
	Uniform	32	0.9884	0.0370	1.08E-02	3.66E-02	1.17E-04	1.34E-03	0.9763	0.0718	3.25E-02	8.83E-02	1.06E-03	7.79E-03
		64	0.9857	0.0510	1.47E-02	5.44E-02	2.17E-04	2.96E-03	0.9738	0.0944	2.21E-02	9.39E-02	4.87E-04	8.82E-03
		96	0.9871	0.0393	1.20E-02	3.63E-02	1.44E-04	1.32E-03	0.9684	0.0998	2.00E-02	6.92E-02	3.98E-04	4.78E-03
		128	0.9812	0.0564	1.19E-02	2.90E-02	1.43E-04	8.39E-04	0.9734	0.0797	2.73E-02	6.11E-02	7.43E-04	3.73E-03
0.25	Random	32	0.9857	0.0480	1.24E-02	3.97E-02	1.54E-04	1.58E-03	0.9847	0.0497	8.89E-03	3.35E-02	7.91E-05	1.12E-03
		64	0.9819	0.0613	1.40E-02	5.57E-02	1.96E-04	3.10E-03	0.9778	0.0741	1.41E-02	4.29E-02	1.98E-04	1.84E-03
		96	0.8999	0.2580	2.74E-01	6.81E-01	7.52E-02	4.64E-01	0.8988	0.2704	2.68E-01	6.78E-01	7.17E-02	4.59E-01
		128	0.8019	0.5089	3.58E-01	8.89E-01	3.59E-01	8.98E-01	0.8128	0.4926	1.28E-01	7.91E-01	1.29E-01	8.06E-01
	Uniform	32	0.9855	0.0511	8.31E-03	2.85E-02	6.90E-05	8.14E-04	0.9775	0.0851	1.42E-02	4.38E-02	2.01E-04	1.92E-03
		64	0.9721	0.1023	2.16E-02	1.07E-01	4.66E-04	1.14E-02	0.9697	0.1413	4.21E-02	2.13E-01	1.78E-03	4.54E-02
		96	0.8881	0.2918	2.70E-01	6.72E-01	7.29E-02	4.51E-01	0.8822	0.3026	2.74E-01	6.69E-01	7.49E-02	4.48E-01
		128	0.8837	0.3061	2.66E-01	6.64E-01	7.06E-02	4.41E-01	0.8681	0.3418	2.65E-01	6.56E-01	7.00E-02	4.30E-01
0.5	Random	32	0.9851	0.0515	1.00E-02	3.02E-02	1.01E-04	9.13E-04	0.9741	0.1062	3.21E-02	1.78E-01	1.03E-03	3.18E-02
		64	0.8837	0.3207	2.68E-01	6.61E-01	7.17E-02	4.37E-01	0.8813	0.3300	2.70E-01	6.66E-01	7.28E-02	4.43E-01
		96	0.8007	0.5152	3.57E-01	8.86E-01	1.28E-01	7.85E-01	0.7806	0.5920	3.54E-01	8.62E-01	1.26E-01	7.43E-01
		128	0.5446	1.1493	4.44E-01	1.10E+00	1.97E-01	1.21E+00	0.5369	1.1527	4.51E-01	1.10E+00	2.03E-01	1.22E+00
	Uniform	32	0.9468	0.1577	1.80E-02	6.30E-02	3.22E-04	3.97E-03	0.9400	0.1845	5.21E-02	1.46E-01	2.71E-03	2.14E-02
		64	0.9628	0.1257	2.69E-02	9.35E-02	7.26E-04	8.75E-03	0.9553	0.1950	3.94E-02	2.28E-01	1.55E-03	5.22E-02
		96	0.7151	0.7371	4.05E-01	1.01E+00	1.64E-01	1.01E+00	0.6894	0.8166	4.08E-01	9.87E-01	1.66E-01	9.74E-01
		128	0.6210	0.9696	4.26E-01	1.05E+00	1.82E-01	1.11E+00	0.6078	0.9907	4.29E-01	1.04E+00	1.84E-01	1.09E+00

Table 4.52: The following results are the averages of 10 trials for each test using a simple rock dataset consisting 9 different rocks. Each test we run have a different dropout, $D = [0, 0.25, 0.5]$, and the number of filters, $F = [32, 64, 96]$. Each row provides the accuracy, loss, standard deviation, and variance for both training and validation results. Note: Uniform is the term we are using to describe Ozbulak's results in [16]. Here we have a shuffled dataset for each epoch for all trials.



(a)



(b)

Figure 4.45: Here we see a different representation of Table 4.52 showing the mean, maximum and minimum values as well as the mean of the validation accuracy. Within the legends is each permutation of the experiment using a shuffled rock dataset in both training and validation sets.

Discussion

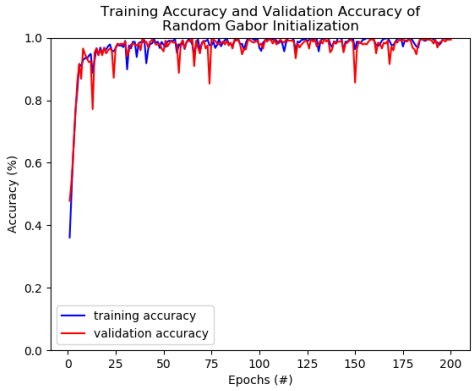
When using a shuffled rock dataset, we see a similar trend that we have found and explored in our experiments (Sections 4.3, 4.4, and 4.5). Adding dropout reduces the performance of the network and we can see this in Table 4.52, for all permutations of dropout and changing the number of filters. Visually, we can see a drop in Figure 4.46, although it is not noticeable in the graph because the network converges very quickly to a high accuracy (or low loss). As we increase the dropout, we also see that in some cases, the network cannot converge and does not train. We can see this directly within the Figure 4.45 where the models with the longer bars show a larger variance in the data.

Likely the most surprising finding that we see from using this dataset is a change in the trend where if we add more filters to layer 1, we see a drop in performance in terms of both accuracy and loss. This could be due to multiple reasons within the architecture, but this finding was also true with Pascual's results as well [17]. This is why he chose to use a smaller number of filters in his first layer. Perhaps it is due to the smaller network size when compared to our experiments above.

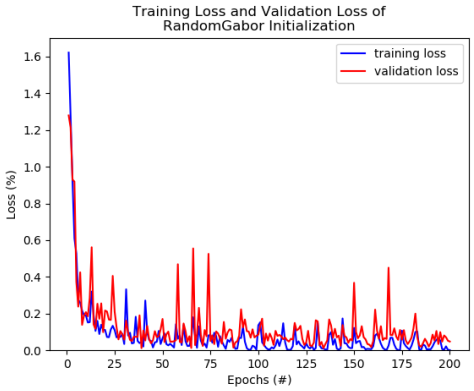
When comparing the different initialization methods (random versus grid-search), we see that when we shuffle the rock dataset in our tests that it prefers either method. Out of the 12 tests comparing the random method versus the grid-search method for the Gabor filter bank in the first layer, they both fair the same. From Table 4.52, we see that when we use a random initialization method, it can offer higher results in terms of validation accuracy, but it can also offer a very low validation accuracy. We see that using a grid-search initialization, that the results are consistently close to each other (for validation accuracy).

In summary, when we use the network from Pascual's network and replacing his first layer with our Gabor filter bank, we see that by adding dropout, it reduces the performance of the network, adding more filters in layer 1 decreases the performance, and using either method of initialization for the Gabor filter bank shows no performance boost over each other. Using the random initialization method can offer higher results, but can also offer lower results when

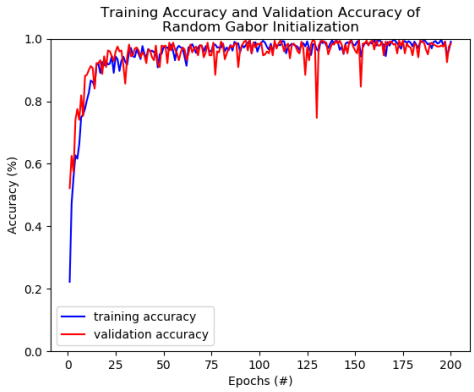
compared to the consistency of using a grid-search.



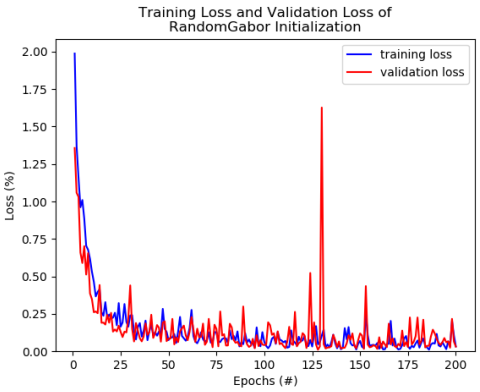
(a) Accuracy: $D = 0$, $F = 32$



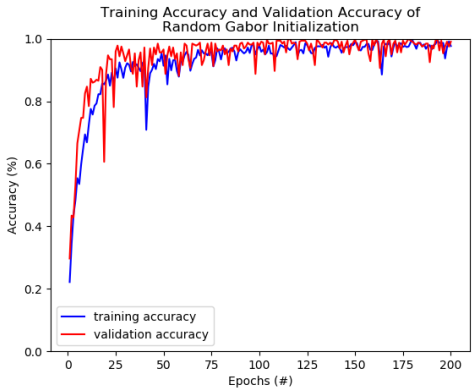
(b) Loss: $D = 0$, $F = 32$



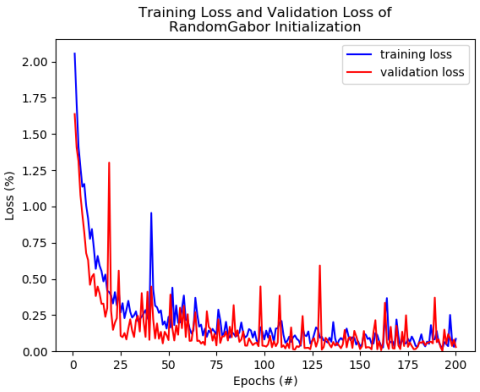
(c) Accuracy: $D = 0.25$, $F = 32$



(d) Loss: $D = 0.25$, $F = 32$

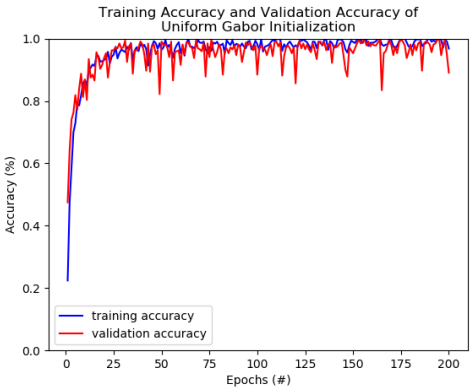


(e) Accuracy: $D = 0.50$, $F = 32$

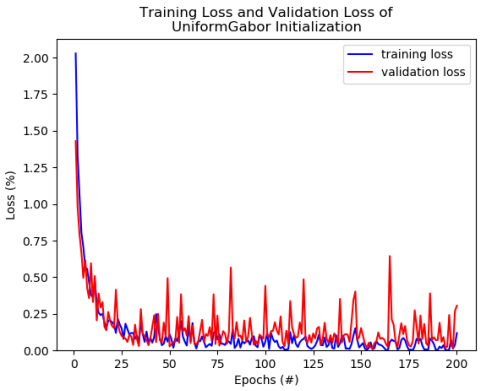


(f) Loss: $D = 0.50$, $F = 32$

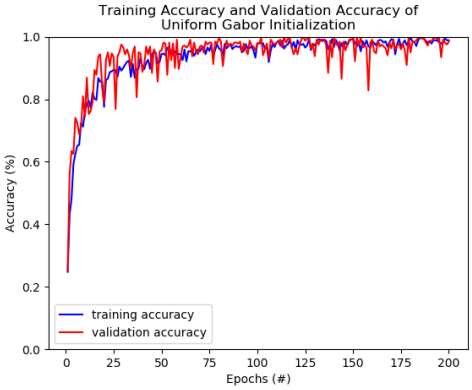
Figure 4.46: Training on the rock dataset provided by Pascual shuffling the dataset in both training and validation sets, we use his architecture but replace his first layer with our Gabor filter bank that is initialized randomly. We see that as we increase the dropout, we reduce the performance. The training converges much slower as we add more dropout.



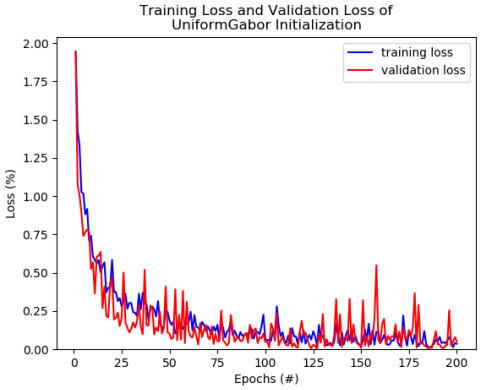
(a) Accuracy: $D = 0$, $F = 32$



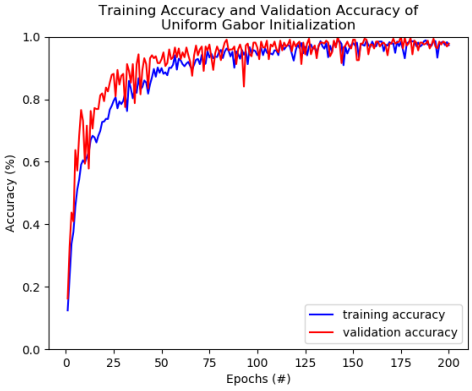
(b) Loss: $D = 0$, $F = 32$



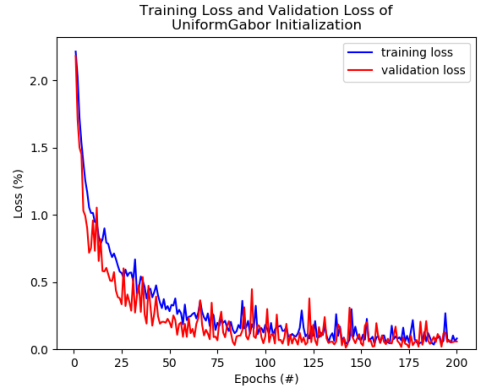
(c) Accuracy: $D = 0.25$, $F = 32$



(d) Loss: $D = 0.25$, $F = 32$



(e) Accuracy: $D = 0.50$, $F = 32$



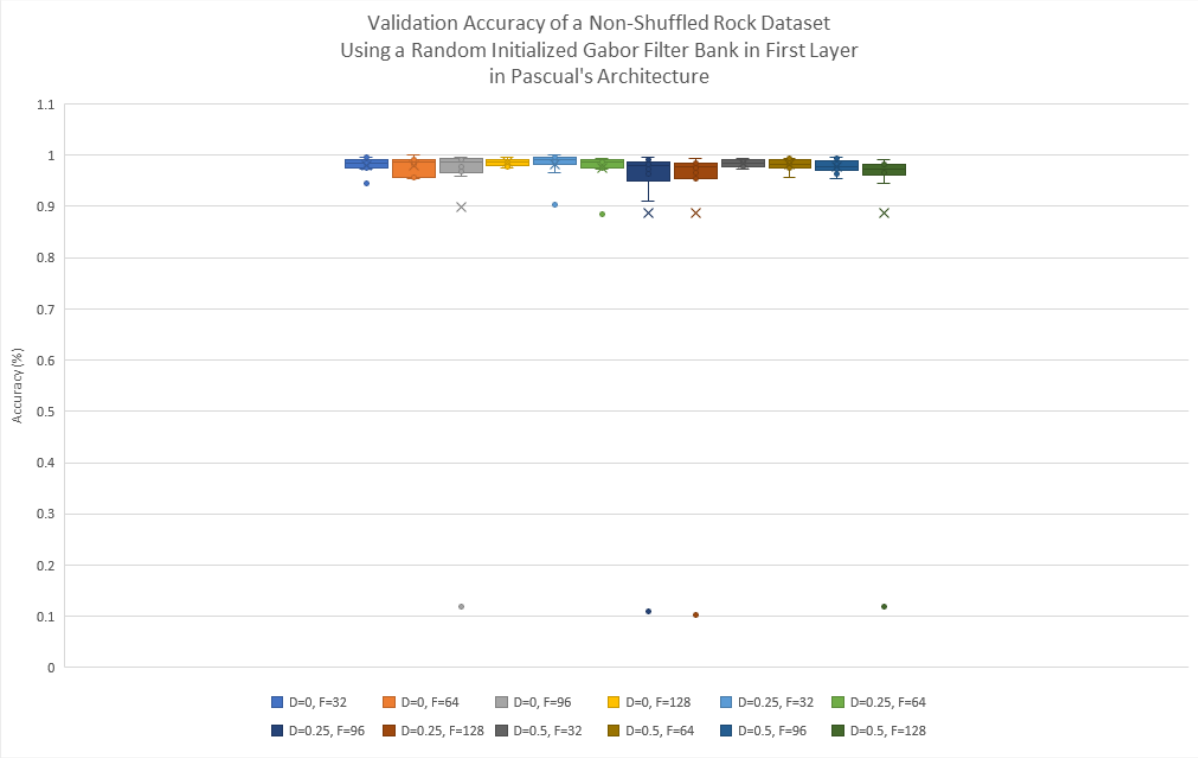
(f) Loss: $D = 0.50$, $F = 32$

Figure 4.47: Training on the rock dataset while shuffling the dataset in both training and validation sets, we use Pascual's architecture but replace his first layer with our Gabor filter bank that is initialized with a grid-search. We see that as we increase the dropout, we reduce the performance. The training converges much slower as we add more dropout.

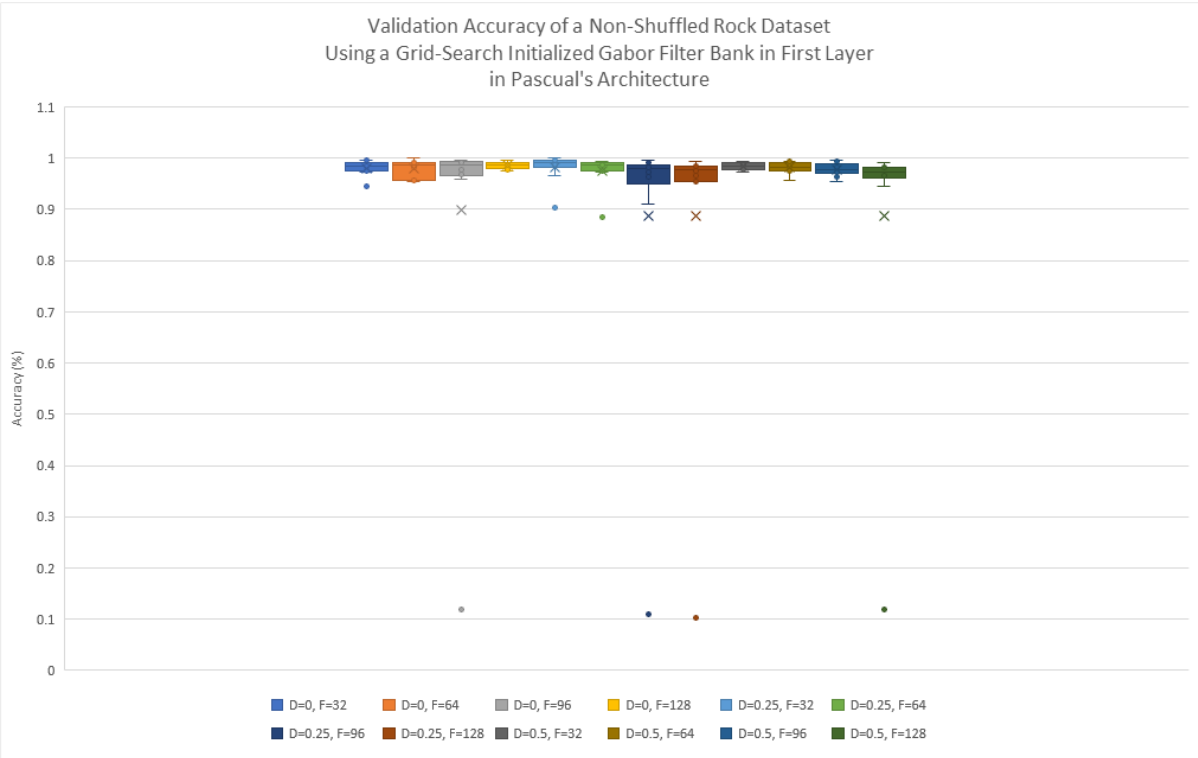
Non-Shuffled Dataset

AlexisNet-9														
Dropout	**Test Type**	**# of Filters**	**Training**						**Validation**					
			Acc	**Loss**	**Std. Dev Acc**	**Std. Dev Loss**	**Var Acc**	**Var Loss**	**Acc**	**Loss**	**Std. Dev Acc**	**Std. Dev Loss**	**Var Acc**	**Var Loss**
0	Random	32	0.9887	0.0377	0.0105	0.0346	0.0001	0.0012	0.9806	0.0632	0.0154	0.0688	0.0002	0.0047
		64	0.9854	0.0459	0.0128	0.0444	0.0002	0.0020	0.9797	0.0634	0.0178	0.0527	0.0003	0.0028
		96	0.9019	0.2495	0.2744	0.6836	0.0753	0.4673	0.8975	0.2641	0.2739	0.6805	0.0750	0.4631
		128	0.9897	0.0317	0.0093	0.0262	0.0001	0.0007	0.9863	0.0512	0.0071	0.0365	0.0001	0.0013
	Uniform	32	0.9909	0.0287	0.0078	0.0256	0.0001	0.0007	0.9728	0.1212	0.0394	0.2282	0.0016	0.0521
		64	0.9896	0.0308	0.0072	0.0213	0.0001	0.0005	0.9753	0.0797	0.0297	0.0996	0.0009	0.0099
		96	0.9888	0.0376	0.0109	0.0414	0.0001	0.0017	0.9834	0.0498	0.0128	0.0365	0.0002	0.0013
		128	0.9007	0.2532	0.2718	0.6818	0.0739	0.4649	0.8897	0.2911	0.2769	0.6753	0.0767	0.4561
0.25	Random	32	0.9839	0.0519	0.0115	0.0359	0.0001	0.0013	0.9813	0.0771	0.0291	0.1547	0.0008	0.0239
		64	0.9816	0.0626	0.0125	0.0393	0.0002	0.0015	0.9747	0.0935	0.0326	0.1141	0.0011	0.0130
		96	0.8909	0.2942	0.2691	0.6692	0.0724	0.4478	0.8869	0.3048	0.2743	0.6719	0.0752	0.4514
		128	0.8946	0.2739	0.2737	0.6753	0.0749	0.4561	0.8872	0.3020	0.2758	0.6687	0.0761	0.4472
	Uniform	32	0.9785	0.0677	0.0176	0.0545	0.0003	0.0030	0.9738	0.1008	0.0167	0.0556	0.0003	0.0031
		64	0.9851	0.0547	0.0106	0.0518	0.0001	0.0027	0.9706	0.1151	0.0337	0.1539	0.0011	0.0237
		96	0.9787	0.0687	0.0127	0.0426	0.0002	0.0018	0.9809	0.0573	0.0138	0.0432	0.0002	0.0019
		128	0.8880	0.2969	0.2684	0.6666	0.0721	0.4444	0.8481	0.4418	0.2642	0.6613	0.0698	0.4373
0.5	Random	32	0.9790	0.0740	0.0141	0.0563	0.0002	0.0032	0.9844	0.0525	0.0074	0.0254	0.0001	0.0006
		64	0.9697	0.1057	0.0216	0.0846	0.0005	0.0072	0.9822	0.0797	0.0115	0.0765	0.0001	0.0058
		96	0.9676	0.0982	0.0201	0.0550	0.0004	0.0030	0.9778	0.0776	0.0140	0.0531	0.0002	0.0028
		128	0.8829	0.3266	0.2680	0.6626	0.0718	0.4390	0.8881	0.2809	0.2706	0.6738	0.0732	0.4541
	Uniform	32	0.9764	0.0853	0.0095	0.0473	0.0001	0.0022	0.9691	0.1051	0.0163	0.0609	0.0003	0.0037
		64	0.9719	0.0936	0.0111	0.0375	0.0001	0.0014	0.9706	0.0960	0.0178	0.0570	0.0003	0.0032
		96	0.7779	0.5969	0.3460	0.8474	0.1197	0.7181	0.7934	0.5402	0.3558	0.8750	0.1266	0.7656
		128	0.7685	0.5973	0.3422	0.8499	0.1171	0.7223	0.7694	0.5821	0.3582	0.8616	0.1283	0.7424

Table 4.53: The following results are the averages of 10 trials for each test using a simple rock dataset consisting 9 different rocks. Each test we run have a different dropout, $D = [0, 0.25, 0.5]$, and the number of filters, $F = [32, 64, 96]$. Each row provides the accuracy, loss, standard deviation, and variance for both training and validation results. Note: Uniform is the term we are using to describe Ozbulak's results in [16]. The results are from using a non-shuffled dataset instead of being shuffled like experiment C (Section 4.3).



(a)



(b)

Figure 4.48: Here we see a different representation of Table 4.52 showing the mean, maximum and minimum values as well as the mean of the validation accuracy. Within the legends is each permutation of the experiment using a non-shuffled rock dataset in both training and validation sets.

Discussion

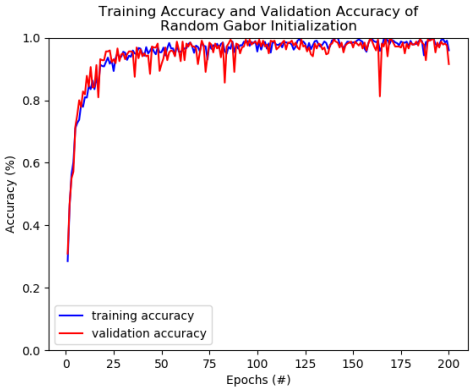
In this experiment we compare the difference between the datasets, but keep everything else the same as the last test. We see that when we add dropout, it shows a similar result when the dataset is shuffled or not shuffled. Adding dropout results in a drop in performance for the network. This is consistent among all experiments done. We can see the results in Table 4.53 that by adding 25% or 50% dropout, we see a reduction in terms of validation accuracy. The reduction in performance is minimal, like the previous test as well.

Adding more filters to the network in layer 1 results in a similar fashion as seen previously. We see that the results are lower which is opposite of what we have found in our tests using our architecture (Section 3.3). These findings are consistent with Pascual's work. Having a lower number of filters performs better in all tests we do in this section and we can see this in Table 4.53.

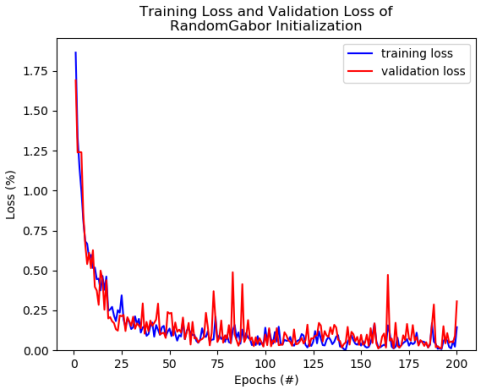
In our findings for using the different initialization methods when the dataset is not shuffled for both training and validation sets, we see that the network favours using a random initialization for the Gabor filter bank in the first layer. In the 12 tests comparing the initialization methods, using the random distribution wins 10 times out of the 12 tests. What's more fascinating is that the network performance is boosted by roughly 1% in terms of validation accuracy when compared to using the grid-search initialization using the exact same hyperparameters.

In summary, we find that when we use a non-shuffled rock dataset using Pascual's architecture (Section 3.3.2, we see similar results obtained from using a shuffled rock dataset. There are 2 noticeable differences: first, when we use a different initialization method (random versus grid-search) for the Gabor filter in the first layer, using a non-shuffled dataset prefers the random search more than using a grid-search (according to our 10 trials). Using the random search also offers a performance boost over a grid-search. Secondly, we see that there is a performance difference on average that using a non-shuffled version of the dataset performs better than using a shuffled dataset. We use shuffling to avoid overfitting in the network, but in all of the tests done, we find that there is no overfitting occurring, even with no dropout added

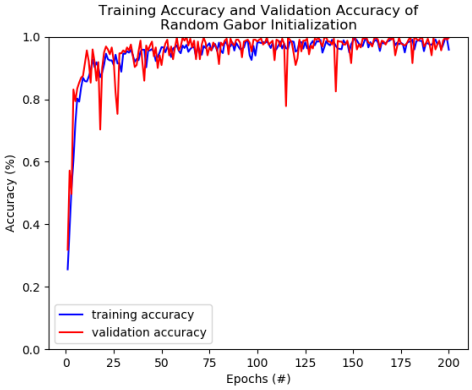
(shown in Table 4.57.



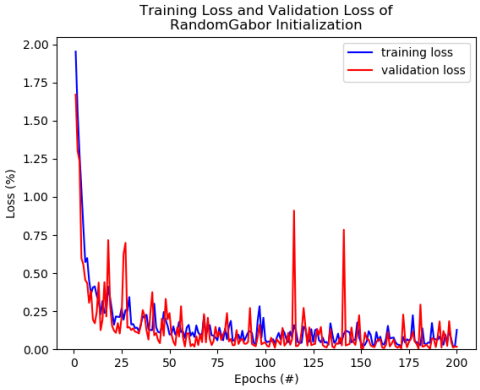
(a) Accuracy: $D = 0$, $F = 32$



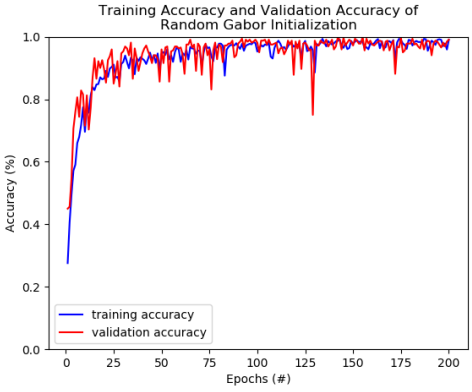
(b) Loss: $D = 0$, $F = 32$



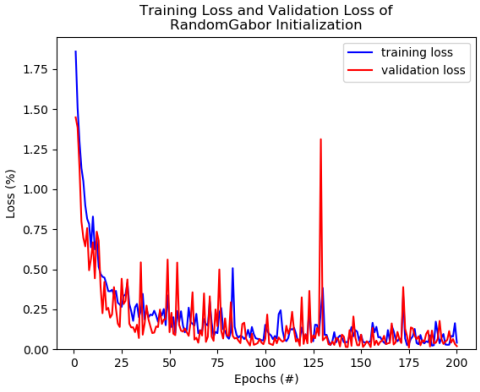
(c) Accuracy: $D = 0.25$, $F = 32$



(d) Loss: $D = 0.25$, $F = 32$

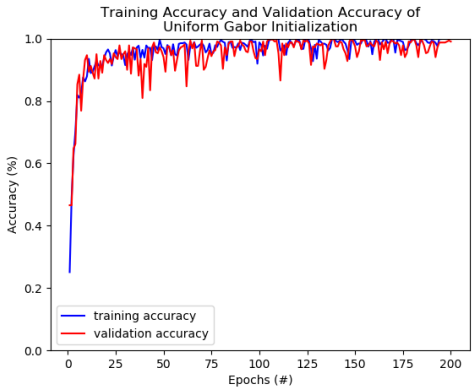


(e) Accuracy: $D = 0.50$, $F = 32$

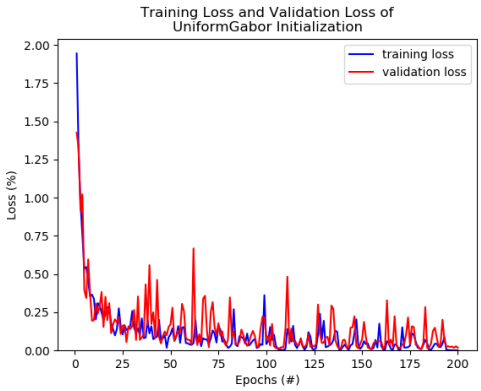


(f) Loss: $D = 0.50$, $F = 32$

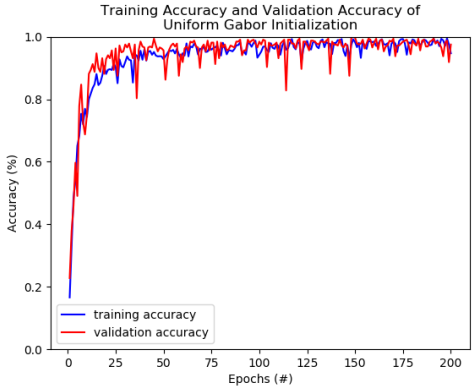
Figure 4.49: Training on the rock dataset provided by Pascual without shuffling the dataset in both training and validation sets, we use his architecture but replace his first layer with our Gabor filter bank that is initialized randomly here. We see that as we increase the dropout, we reduce the performance. The training converges much slower as we add more dropout.



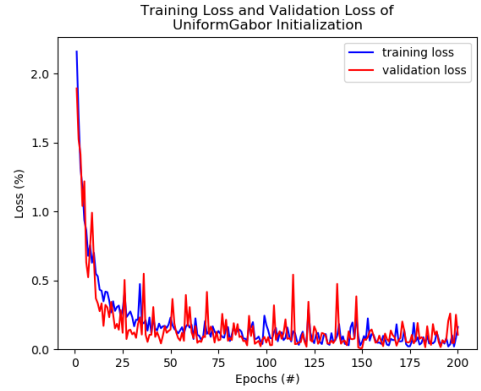
(a) Accuracy: $D = 0$, $F = 32$



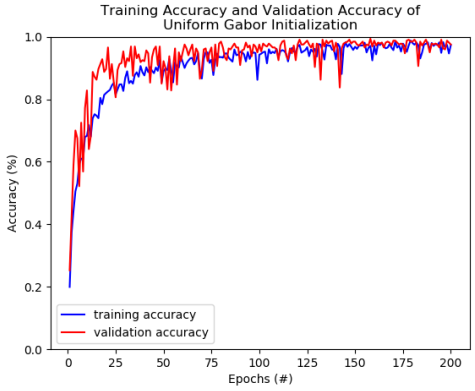
(b) Loss: $D = 0$, $F = 32$



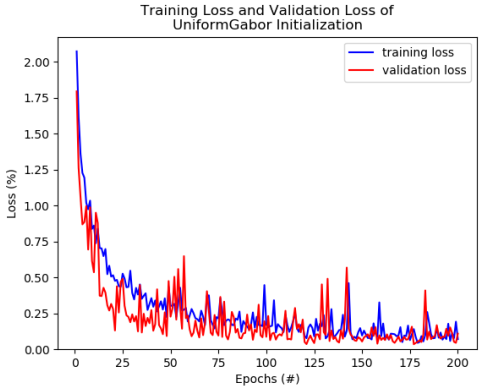
(c) Accuracy: $D = 0.25$, $F = 32$



(d) Loss: $D = 0.25$, $F = 32$



(e) Accuracy: $D = 0.50$, $F = 32$



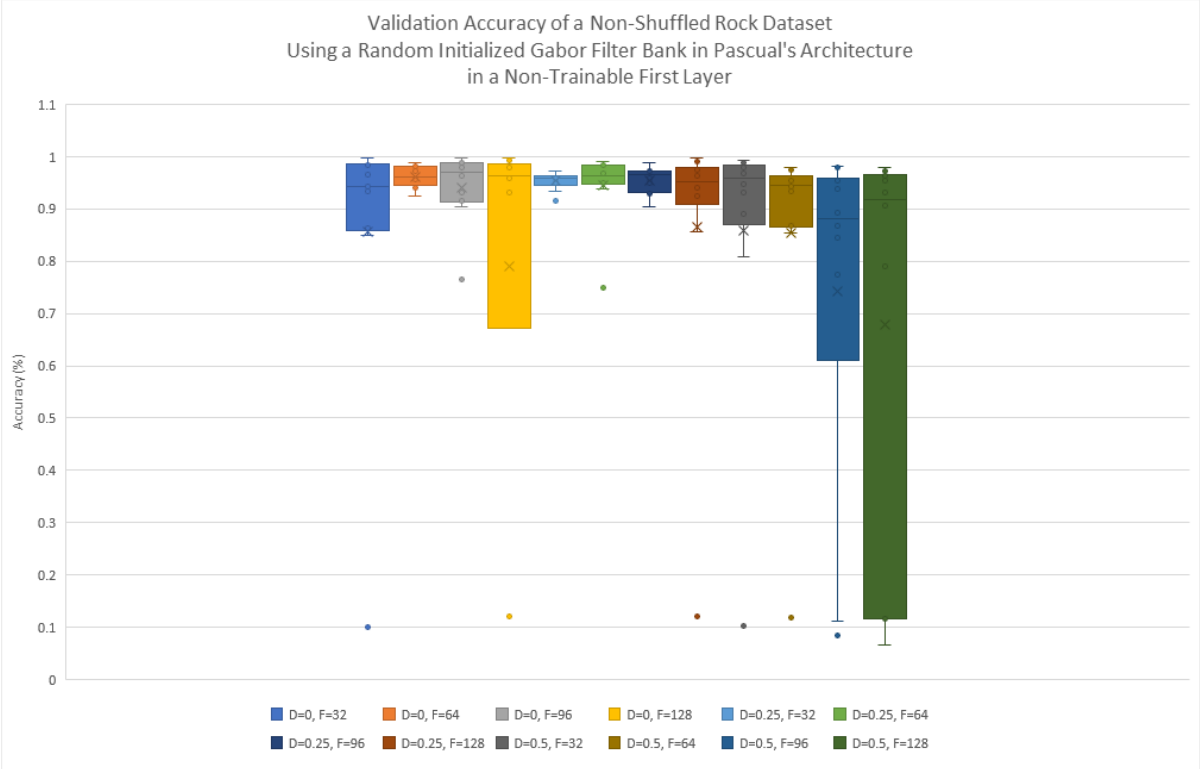
(f) Loss: $D = 0.50$, $F = 32$

Figure 4.50: Training on the rock dataset provided by Pascual without shuffling the dataset in both training and validation sets, we use his architecture but replace his first layer with our Gabor filter bank that is initialized with a grid-search. We see that as we increase the dropout, we reduce the performance. The training converges much slower as we add more dropout.

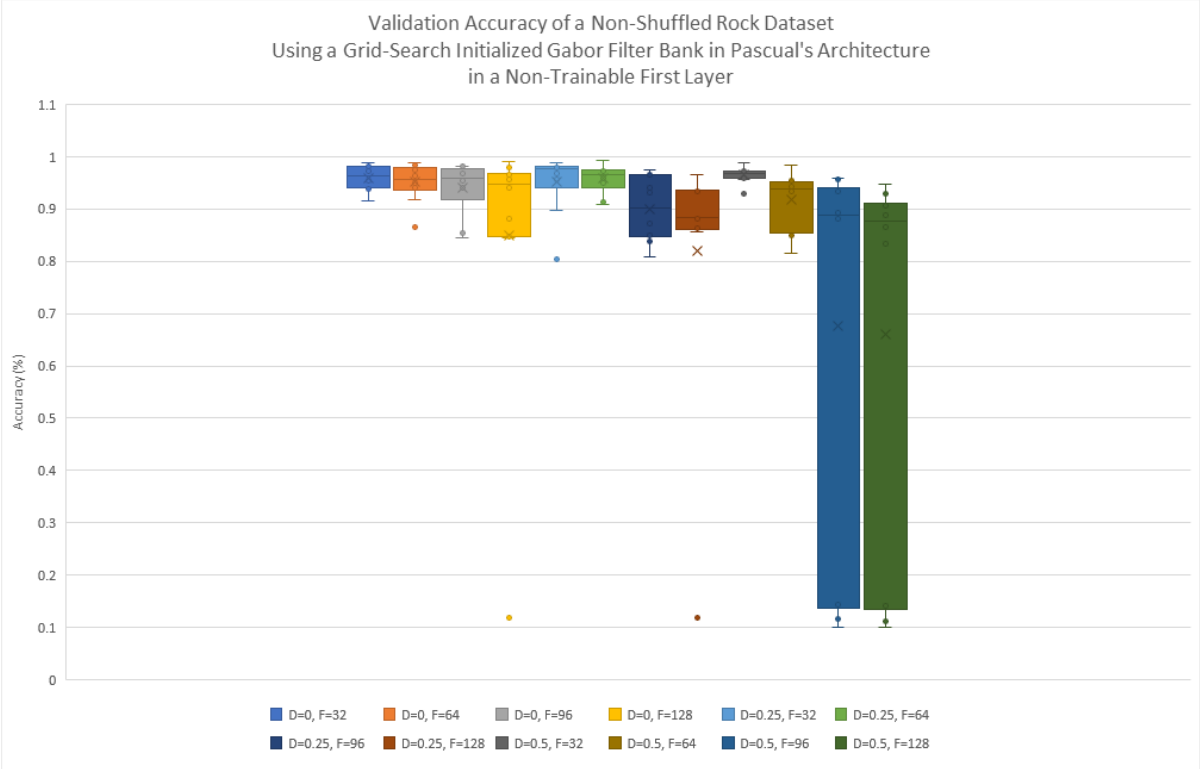
Frozen First Layer, Non-Shuffled Dataset

AlexisNet-9														
Dropout	**Test Type**	**# of Filters**	**Training**						**Validation**					
			Acc	**Loss**	**Std. Dev Acc**	**Std. Dev Loss**	**Var Acc**	**Var Loss**	**Acc**	**Loss**	**Std. Dev Acc**	**Std. Dev Loss**	**Var Acc**	**Var Loss**
0	Random	32	0.8668	0.3405	0.2669	0.6679	0.0712	0.4462	0.8566	0.4079	0.2704	0.6451	0.0731	0.4162
		64	0.9871	0.0401	0.0113	0.0339	0.0001	0.0011	0.9613	0.1668	0.0199	0.1048	0.0004	0.0110
		96	0.9821	0.0556	0.0138	0.0469	0.0002	0.0022	0.9413	0.2253	0.0700	0.2701	0.0049	0.0730
		128	0.8094	0.4854	0.3628	0.9003	0.1317	0.8106	0.7909	0.5572	0.3543	0.8749	0.1255	0.7655
	Uniform	32	0.9832	0.0466	0.0069	0.0195	0.0000	0.0004	0.9591	0.1505	0.0248	0.1029	0.0006	0.0106
		64	0.9782	0.0668	0.0092	0.0311	0.0001	0.0010	0.9513	0.2249	0.0370	0.1941	0.0014	0.0377
		96	0.9771	0.0669	0.0297	0.0799	0.0009	0.0064	0.9413	0.1965	0.0506	0.1774	0.0026	0.0315
		128	0.8589	0.3598	0.2596	0.6518	0.0674	0.4248	0.8484	0.4244	0.2619	0.6392	0.0686	0.4085
0.25	Random	32	0.9729	0.0857	0.0251	0.0690	0.0006	0.0048	0.9531	0.1898	0.0167	0.0723	0.0003	0.0052
		64	0.9583	0.1268	0.0212	0.0630	0.0004	0.0040	0.9450	0.2331	0.0706	0.3387	0.0050	0.1147
		96	0.9589	0.1170	0.0425	0.1071	0.0018	0.0115	0.9550	0.1838	0.0265	0.1127	0.0007	0.0127
		128	0.8804	0.3189	0.2651	0.6595	0.0703	0.4350	0.8663	0.3897	0.2647	0.6538	0.0701	0.4275
	Uniform	32	0.9689	0.0992	0.0128	0.0464	0.0002	0.0022	0.9513	0.1745	0.0584	0.2352	0.0034	0.0553
		64	0.9741	0.0824	0.0131	0.0412	0.0002	0.0017	0.9581	0.2033	0.0274	0.1614	0.0008	0.0260
		96	0.9195	0.2164	0.0611	0.1437	0.0037	0.0206	0.9006	0.2697	0.0620	0.1462	0.0038	0.0214
		128	0.8502	0.3935	0.2584	0.6412	0.0668	0.4111	0.8200	0.5179	0.2491	0.6055	0.0621	0.3667
0.5	Random	32	0.8567	0.3852	0.2577	0.6391	0.0664	0.4084	0.8591	0.4521	0.2715	0.6996	0.0737	0.4894
		64	0.8556	0.3875	0.2669	0.6473	0.0712	0.4191	0.8531	0.4227	0.2614	0.6522	0.0683	0.4253
		96	0.7421	0.6666	0.3322	0.8191	0.1104	0.6709	0.7428	0.6949	0.3456	0.8110	0.1195	0.6577
		128	0.6847	0.8153	0.3883	0.9588	0.1507	0.9194	0.6791	0.8499	0.4041	0.9473	0.1633	0.8973
	Uniform	32	0.9495	0.1561	0.0126	0.0440	0.0002	0.0019	0.9647	0.1076	0.0158	0.0427	0.0002	0.0018
		64	0.9206	0.2135	0.0715	0.1574	0.0051	0.0248	0.9169	0.2707	0.0554	0.1639	0.0031	0.0269
		96	0.6630	0.8564	0.3753	0.9263	0.1408	0.8581	0.6769	0.8355	0.3855	0.9416	0.1486	0.8867
		128	0.6528	0.8822	0.3683	0.9106	0.1356	0.8293	0.6616	0.9101	0.3767	0.8901	0.1419	0.7922

Table 4.54: The following results are the averages of 10 trials for each test using a simple rock dataset consisting 9 different rocks. Each test we run have a different dropout, $D = [0, 0.25, 0.5]$, and the number of filters, $F = [32, 64, 96]$. Each row provides the accuracy, loss, standard deviation, and variance for both training and validation results. Note: Uniform is the term we are using to describe Ozbulak's results in [16]. The results are from using a non-shuffled dataset instead of being shuffled like experiment C (Section 4.3). The first layer in the CNN is also untrainable to see how well the Gabor filter bank can extract features without learning.



(a)



(b)

Figure 4.51: Here we see a different representation of Table 4.52 showing the mean, maximum and minimum values as well as the mean of the validation accuracy. Within the legends is each permutation of the experiment using a non-shuffled rock dataset in both training and validation sets. The first layer within the network of each test is also frozen so that it cannot learn during training.

Discussion

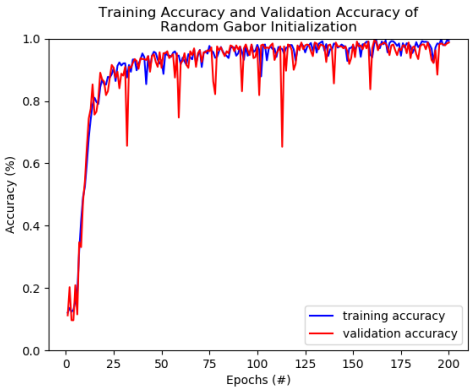
Following the trend within this section, we see that by freezing the first layer and disallowing the layer to learn through training, adding dropout still reduces the performance of the network. It does not appear if we look at the results from using 32 filters within the first layer, but when we increase the number of filters and increase dropout, we see that the performance in terms of validation accuracy drops. This is consistent among all experiments and tests when we use the Gabor filter bank in the first layer. The accuracy and loss still obtain a good results which shows that the Gabor filter is indeed a contender in extracting features and using it within the network to train the CNN. We can see the drop in performance in Figures 4.52 and 4.53.

When we change the number of filters in the first layer, we can see that the performance drops in all aspects of the network. This is a consistent pattern found throughout this experiment. This is also consistent with Pascual, where he had found that increasing the number of filters also decreases the performance of his network. This can be seen for both initialization methods with the Gabor filter as well as the different levels of dropout used ($D = [0, 0.25, 0.50]$).

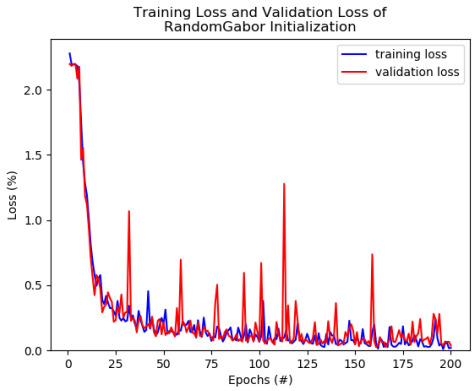
Comparing the initialization methods for the Gabor filter, we see that when we work with Pascual's network, by substituting the first layer with our Gabor filter bank, we see that the rock datasets does not have a preference. In the 12 tests where we compare the different initialization methods, they are equal. Random initialized tests appear to show higher results but also show lower results than the grid-search. The grid-search initialized tests show that the results obtained are consistent and are close to each other more so than the random initialized tests (Figure 4.51).

In comparison to the network where we enable the first layer to learn through training, these results obtained are fairly similar. The network approaches convergence just as fast as having a network where we disable the ability to learn through training for the first layer. We can see these similarities in Figures 4.52 and 4.53 for the non-trainable first layer network and Figures 4.49 and 4.50 for the trainable first layer network.

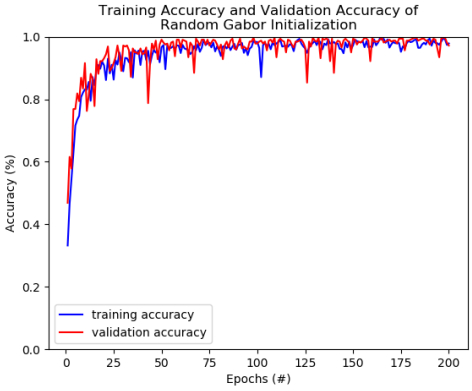
In summary, freezing the first layer in the network using a Gabor filter bank affects the performance slightly which shows that the Gabor filter is superb in extraction of features regardless of training or not. Adding dropout to the network in this experiment shows a drop in performance in all aspects of the network. Increasing the number of filter in the first layer for the Gabor filter bank shows a decrease in performance. Using either method of initialization for the Gabor filter bank will generate a good result, but using a grid-search shows more consistency. Random search here shows it can obtain higher results than grid-search but also lower results than the grid-search.



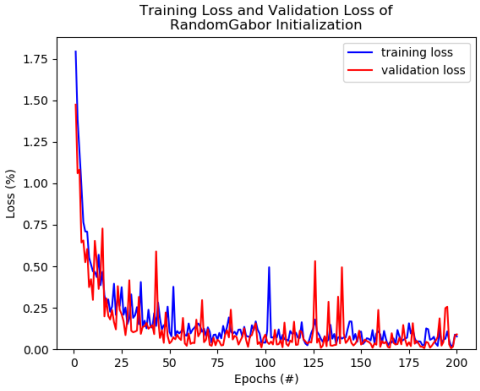
(a) Accuracy: $D = 0$, $F = 32$



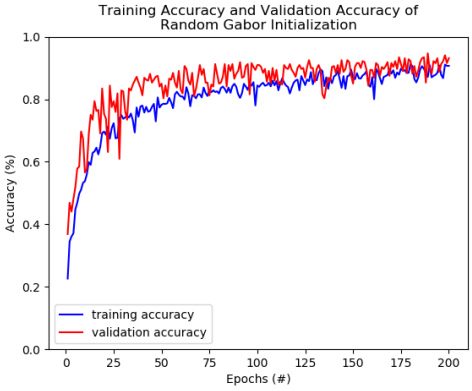
(b) Loss: $D = 0$, $F = 32$



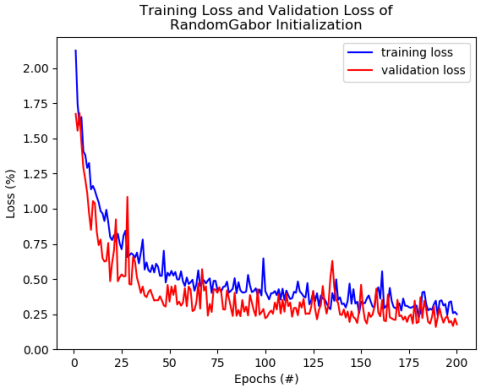
(c) Accuracy: $D = 0.25$, $F = 32$



(d) Loss: $D = 0.25$, $F = 32$

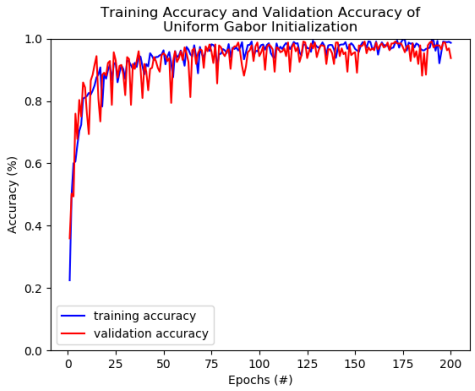


(e) Accuracy: $D = 0.50$, $F = 32$

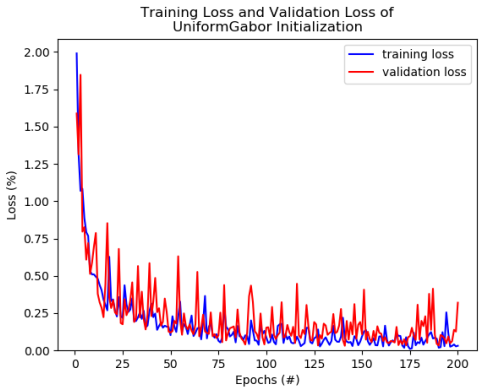


(f) Loss: $D = 0.50$, $F = 32$

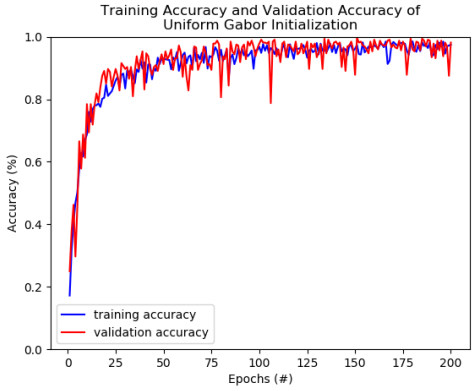
Figure 4.52: Training on the rock dataset [17] without shuffling the dataset in both training and validation sets, we use his architecture but replace his first layer with our Gabor filter bank that is initialized randomly here. We also disable the ability for the first layer in the network to learn. We see that as we increase the dropout, we reduce the performance. The training converges much slower as we add more dropout. We also see a drop in performance in comparison to having the first layer learn through training.



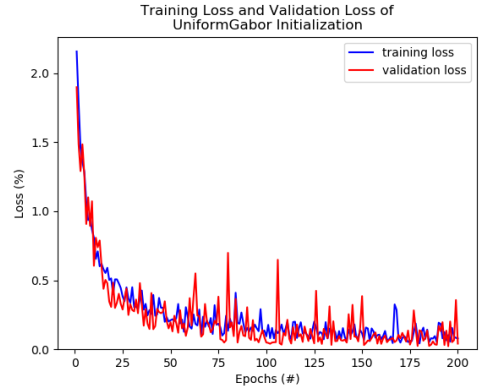
(a) Accuracy: $D = 0$, $F = 32$



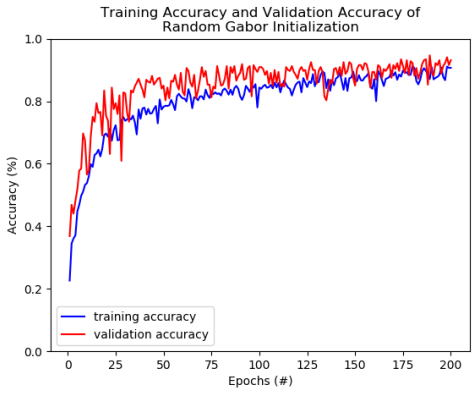
(b) Loss: $D = 0$, $F = 32$



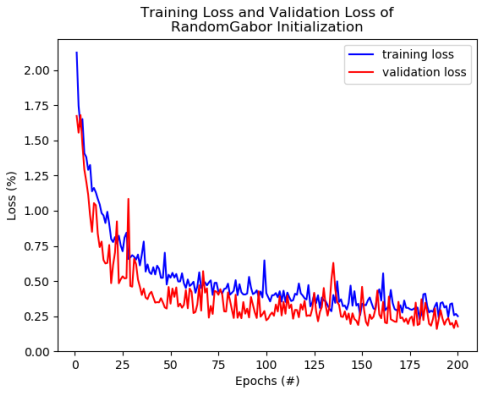
(c) Accuracy: $D = 0.25$, $F = 32$



(d) Loss: $D = 0.25$, $F = 32$



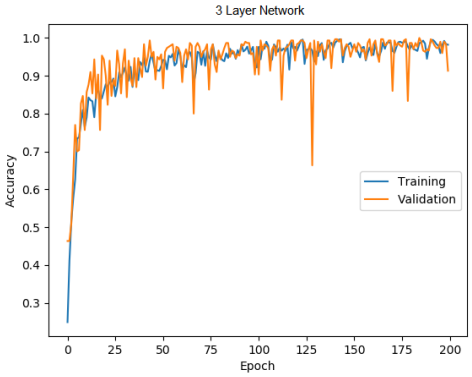
(e) Accuracy: $D = 0.50$, $F = 32$



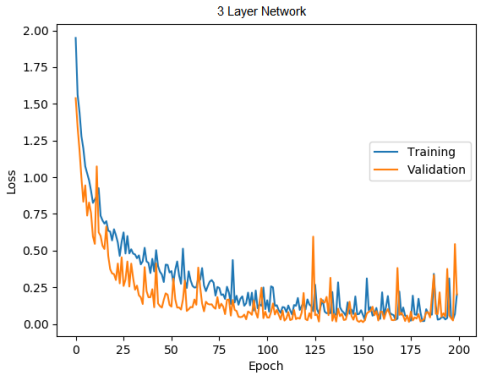
(f) Loss: $D = 0.50$, $F = 32$

Figure 4.53: Training on the rock dataset [17] without shuffling the dataset in both training and validation sets, we use his architecture but replace his first layer with our Gabor filter bank that is initialized with a grid-search. We also disable the ability for the first layer in the network to learn. We see that as we increase the dropout, we reduce the performance. The training converges much slower as we add more dropout. We also see a drop in performance in comparison to having the first layer learn through training.

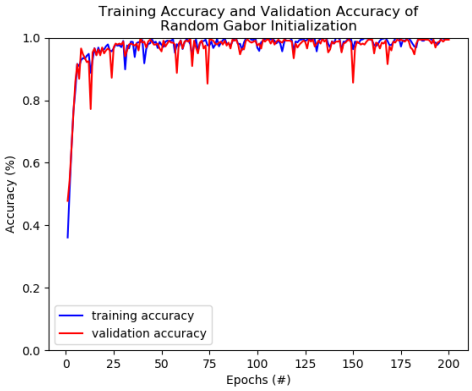
Pascual Comparison



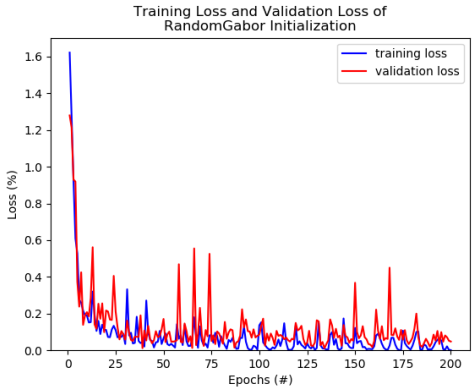
(a) Accuracy: $L = 3$, $F = 32$, $D = 0.50$



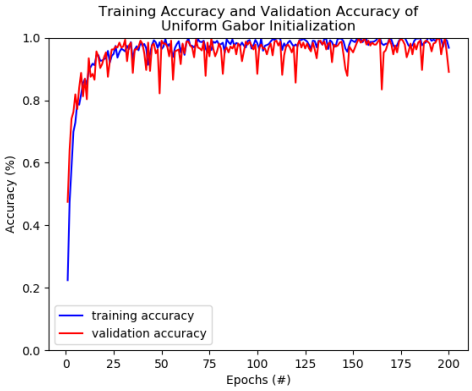
(b) Loss: $L = 3$, $F = 32$, $D = 0.50$



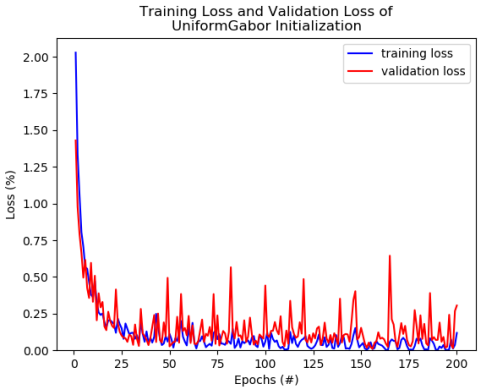
(c) Accuracy: $L = 3$, $F = 32$, $D = 0$



(d) Loss: $L = 3$, $F = 32$, $D = 0$



(e) Accuracy: $L = 3$, $F = 32$, $D = 0$



(f) Loss: $L = 3$, $F = 32$, $D = 0$

Figure 4.54: In figures (a) and (b), we show the training curve using Pascual's network. In figures (c), (d), (e), and (f), we see Pascual's network training on the Rock dataset but instead we use our random or grid-search initialized Gabor filter bank in the first layer in place of his Xavier initialized first layer. We see that our results show that the network converges faster (specifically in the first 25 epochs).

The following tables below (Tables 4.55 and 4.56), we will show how the training progresses for each of the networks created using Pascual's architecture as the base. We use our Gabor filter bank that is initialized either randomly or using a grid-search algorithm. We compare to his results where he uses an Xavier distribution within his first layer. All 3 networks use the same hyperparameters, and the number of filters in the first layer do not change (staying at 32 filters).

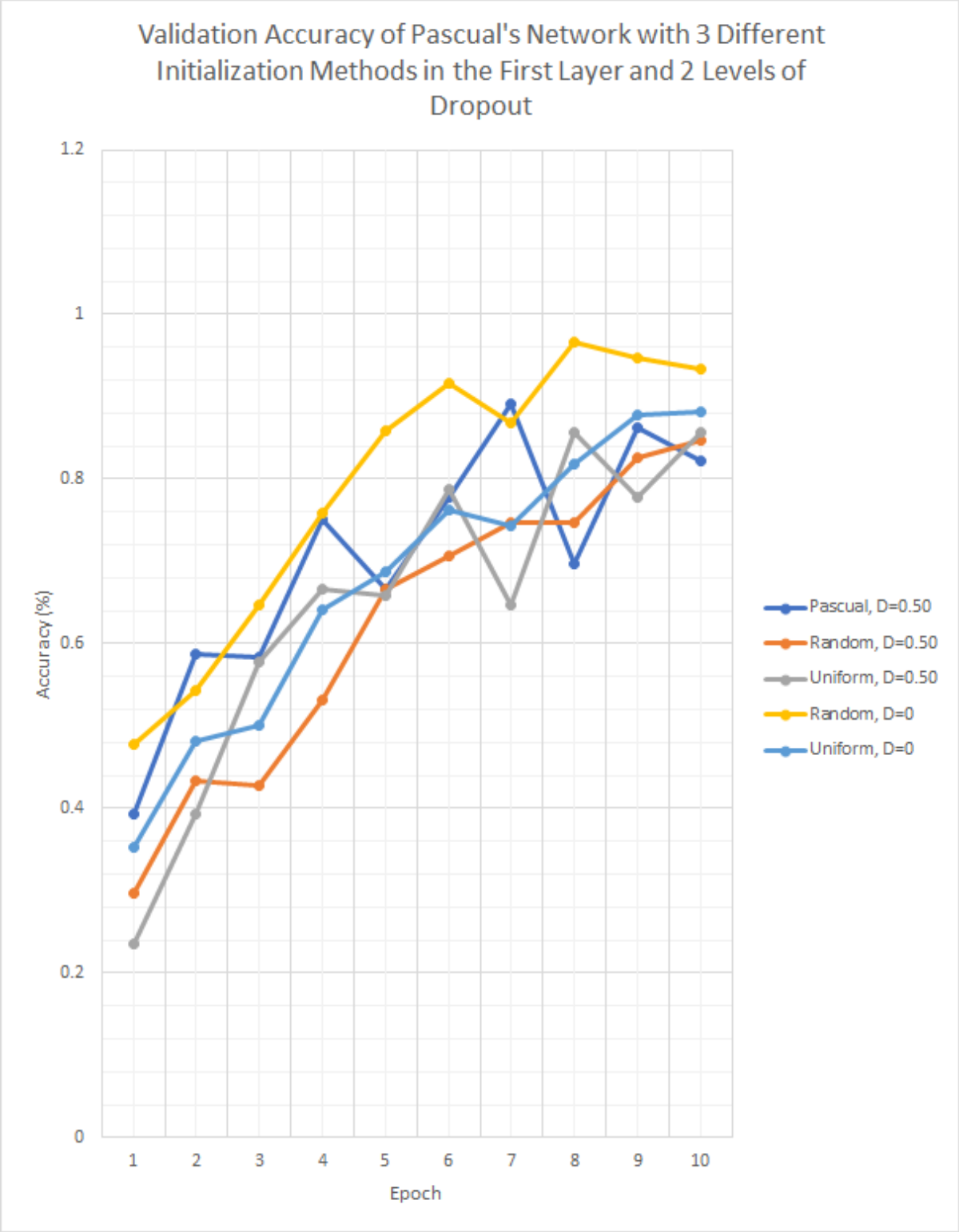


Figure 4.55: A visual representation of the results found in Table 4.55 showing that our network with the Gabor filter bank within the first layer trains faster within the initial epochs of training. Here, we show the validation accuracy results during training.

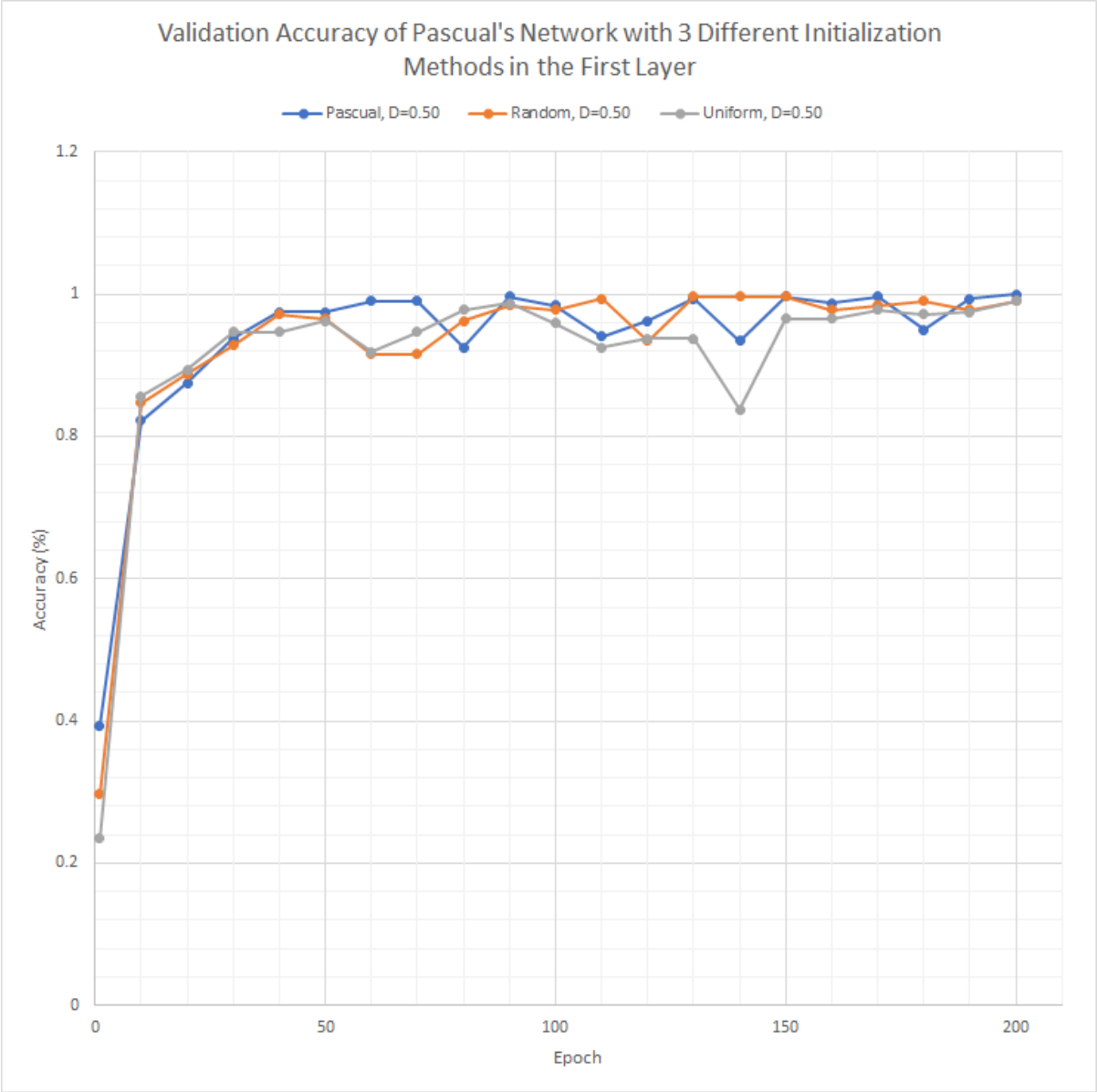


Figure 4.56: A visual representation of the results found in Table 4.56 showing that our network with the Gabor filter bank within the first layer trains faster within the initial epochs of training but in the end, Pascual's network without the Gabor filter performs the best, in terms of validation accuracy

AlexisNet-9								
Filter	**Dropout**	**Epoch**	**Training**			**Validation**		
			Pascual	**Random**	**Uniform**	**Pascual**	**Random**	**Uniform**
32	0.50	1	0.2565	0.2217	0.1701	0.3938	0.2969	0.2344
		2	0.4260	0.3443	0.3463	0.5875	0.4344	0.3938
		3	0.5151	0.4367	0.4575	0.5844	0.4281	0.5781
		4	0.6021	0.4829	0.5573	0.7500	0.5313	0.6656
		5	0.6316	0.5539	0.6524	0.6656	0.6656	0.6594
		6	0.6859	0.5352	0.7019	0.7781	0.7063	0.7875
		7	0.7160	0.5988	0.6524	0.8906	0.7469	0.6469
		8	0.7388	0.6477	0.7120	0.6969	0.7469	0.8563
		9	0.7843	0.6939	0.7435	0.8625	0.8250	0.7781
		10	0.7984	0.6685	0.7837	0.8219	0.8469	**0.8563**
	0	1	-	0.3610	0.2525	-	0.4781	0.3531
		2	-	0.5117	0.4555	-	0.5438	0.4813
		3	-	0.6484	0.5466	-	0.6469	0.5000
		4	-	0.7649	0.6792	-	0.7594	0.6406
		5	-	0.8319	0.6839	-	0.8594	0.6875
		6	-	0.9163	0.7334	-	0.9156	0.7625
		7	-	0.9089	0.8024	-	0.8688	0.7438
		8	-	0.9283	0.8205	-	0.9656	0.8188
		9	-	0.9350	0.8553	-	0.9469	0.8781
		10	-	0.9330	0.8855	-	**0.9344**	0.8813

Table 4.55: Comparing the results obtained from Pascual's work, here we see the training and validation results of the first 10 epochs. In Pascual's work, we find that the training is much slower in comparison to our network created with the Gabor filter bank in the first layer. The network used in this experiment are all the same except for how the first layer is initialized (either a random Gabor filter bank, grid-search Gabor filter bank, or an Xavier distribution). We show that without adding dropout to the network in our tests, the network converges even faster.

AlexisNet-9								
Filter	Dropout	Epoch	Training			Validation		
			Pascual	Random	Uniform	Pascual	Random	Uniform
32	0.50	1	0.2565	0.2217	0.1701	0.3938	0.2969	0.2344
		10	0.7984	0.6685	0.7837	0.8219	0.8469	0.8563
		20	0.8339	0.8667	0.8506	0.8750	0.8875	0.8938
		30	0.8647	0.9076	0.9042	0.9375	0.9281	0.9469
		40	0.9330	0.9290	0.9036	0.9750	0.9719	0.9469
		50	0.9451	0.9136	0.9056	0.9750	0.9656	0.9625
		60	0.9551	0.9263	0.9350	0.9906	0.9156	0.9188
		70	0.9692	0.9632	0.9444	0.9906	0.9156	0.9469
		80	0.9283	0.9337	0.9665	0.9250	0.9625	0.9781
		90	0.9759	0.9665	0.9739	0.9969	0.9844	0.9875
		100	0.9679	0.9779	0.9605	0.9844	0.9781	0.9594
		110	0.9310	0.9625	0.9732	0.9406	0.9938	0.9250
		120	0.9772	0.9444	0.9719	0.9625	0.9344	0.9375
		130	0.9766	0.9752	0.9772	0.9938	0.9969	0.9375
		140	0.9732	0.9712	0.9377	0.9344	0.9969	0.8375
		150	0.9712	0.9886	0.9250	0.9969	0.9969	0.9656
		160	0.9705	0.9826	0.9786	0.9875	0.9781	0.9656
		170	0.9839	0.9712	0.9732	0.9969	0.9844	0.9781
		180	0.9839	0.9946	0.9612	0.9500	0.9906	0.9719
		190	0.9745	0.9658	0.9772	0.9938	0.9781	0.9750
		200	0.9826	0.9766	0.9853	**1.0000**	0.9906	0.9906

Table 4.56: Comparing the results obtained from Pascual's work, here we see the training and validation results of the of all 200 epochs. In Pascual's work, we find that the training is much slower in comparison to our network created with the Gabor filter bank in the first layer for the initial epochs, but afterwards, Pascual's network starts surpass the performance of our network. The network used in this experiment are all the same except for how the first layer is initialized (either a random Gabor filter bank, grid-search Gabor filter bank, or an Xavier distribution). Pascual's network eventually beats our network at the end of its training.

4.6.3 Summary

[Confirm that dropout adds no benefits] We see that by adding dropout results in a drop in performance for the network. We can see these results across all aspects (accuracy, loss for both training and validation). This is a consistent trend when we use the Gabor filter bank in the first layer rather than using a defaulted initialization method such as the Xavier distribution.

[New: Increasing filters drops performance] Adding more filters in the first layer to the network seems to add more complexity and ultimately reduces the performance of the network when we use both our Gabor filter bank and a normal distribution (Xavier distribution) like Pascual's network. This is a slight advantage because it reduces the amount of parameters that are used during training which reduces the complexity of the CNN.

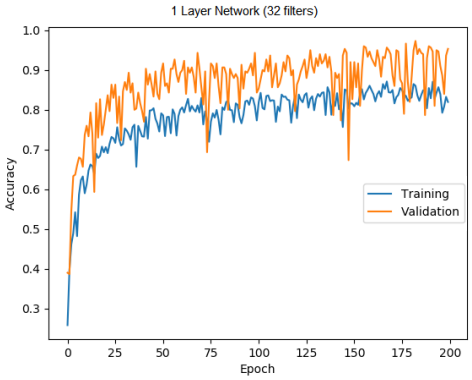
[New: Rock dataset →random GCNN] There is no clear advantage of using either initialization method for the Gabor filter bank. It appears that when the dataset is not shuffled, the random search for the Gabor filter performs better overall while boosting the performances over the grid-search initialization method of the Gabor filter. Grid-search initialization offers a consistent approach to obtaining the results from training, while using a random search can provide higher values in terms of accuracy and a lower error as well. Although random search can provide higher results, it can also produce lower results when comparing to the grid-search.

[New: Non-shuffled datasets slightly increase performance] In terms of validation accuracy, we see better results when the dataset is not shuffled. When the dataset is shuffled, results become more unreliable, especially as we increase the dropout and increase the number of filters. We can see these directly within the results, or visually from Figures 4.45, 4.48, and 4.51.

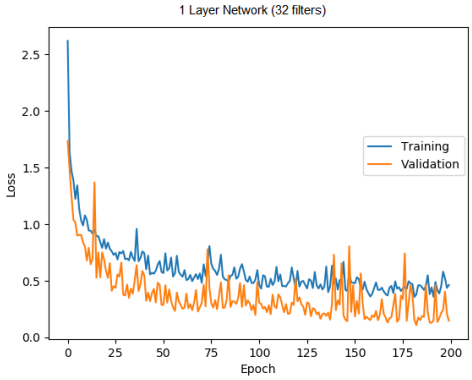
[New: GCNN initially trains faster than AlexisNet] When comparing the results that Pascual obtained from his network, we see that it converges very quickly and reaches a high accuracy (99.7% accuracy). When we compare our work to his, we see that our network fails to reach his performance but we obtain a faster convergence in the beginning phases of training. During the first couple of epochs, our Gabor CNN manages to reach a higher accuracy faster while maintaining a lower loss. We can see this in Figure 4.54 showing both random initialization and a grid-search initialization of the Gabor filter bank in the first layer. The architecture is the same and more specifically, the dropout used is the same at 50% and the number of filters in the first layer is the same (32 filters).

[Summary of Experiment F] In summary for experiment F for using Pascual's network

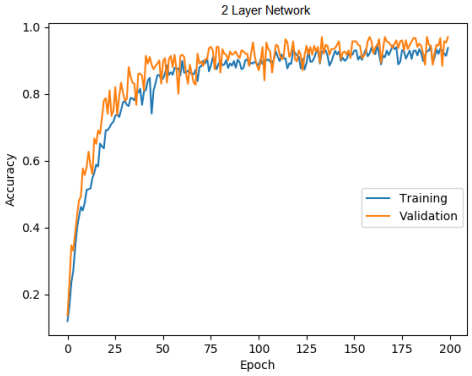
and dataset, we see that it is a simple dataset like the MNIST dataset, while colourful like the CIFAR-10 dataset. It was expected to act more like the CIFAR-10 dataset, but in reality, the dataset was too simple so the network converged fast with high accuracy. We also see that the network performs better with the rock dataset not shuffled. Adding dropout and increasing the number of filters reduces the performance of the network. Using either method of initialization for the Gabor filter in the first layer is optimal. Freezing the first layer in the network will reduce the performance but still shows that the Gabor filter is able to extract features at a high impact rate.



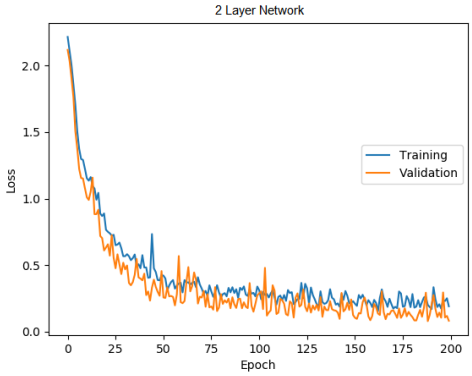
(a) Accuracy: $L = 1$, $F = 32$



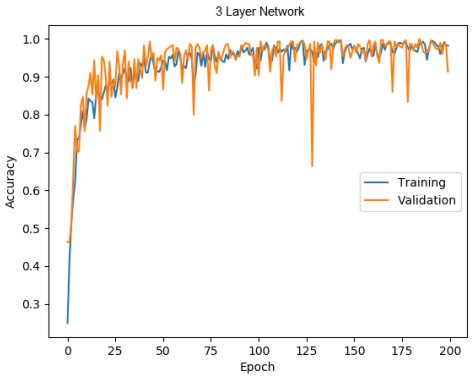
(b) Loss: $L = 1$, $F = 32$



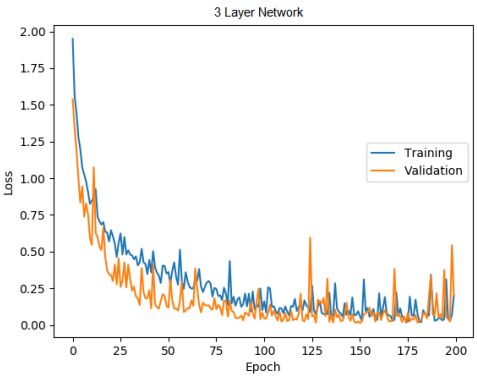
(c) Accuracy: $L = 2$, $F = 32$



(d) Loss: $L = 2$, $F = 32$



(e) Accuracy: $L = 3$, $F = 32$



(f) Loss: $L = 3$, $F = 32$

Figure 4.57: We show Pascual's progression of improved results on the rock dataset. In his findings, he obtains his best score using a smaller number of filters (here he is using 32 filters) in his first layer and simultaneously increasing the number of layers from 1 to 3. In his architecture, he uses a dropout of 50% for all of his experiments.

Experiment	Model	Dropout	Testing		Standard Deviation	
			Accuracy	Loss	Accuracy	Loss
Pascual-A	1 layer, 32 filters	0.50	0.9363	0.1851	0.0518	0.0184
	1 layer, 64 filters		0.9080	0.2499	0.0663	0.1963
	1 layer, 96 filters		0.8933	0.2827	0.0207	0.2860
Pascual-B	2 layer, 32 filters		0.9700	0.0763	0.0262	0.0637
	3 layer, 32 filters		**0.9960**	**0.0160**	**0.0055**	**0.0195**
Pham	3 layer, 32 filters, shuffled, random initialization	0	0.9772	0.0731	0.0238	0.0828
	3 layer, 32 filters, non-shuffled, random initialization		0.9806	0.0632	0.0154	0.0688
	3 layer, 32 filters, non-shuffled, random initialization, untrainable first layer		0.8566	0.4079	0.2704	0.6451
	3 layer, 32 filters, shuffled, grid-search initialization		0.9763	0.0718	0.0325	0.0883
	3 layer, 32 filters, non-shuffled, grid-search initialization		0.9728	0.1212	0.0394	0.2282
	3 layer, 32 filters, non-shuffled, grid-search initialization, untrainable first layer		0.9591	0.1505	0.0248	0.1029
	3 layer, 32 filters, shuffled, random initialization	0.50	0.9741	0.1062	0.0321	0.1783
	3 layer, 32 filters, non-shuffled, random initialization		0.9844	0.0525	0.0074	0.0254
	3 layer, 32 filters, non-shuffled, random initialization, untrainable first layer		0.8591	0.4521	0.2715	0.6996
	3 layer, 32 filters, shuffled, grid-search initialization		0.9400	0.1845	0.0521	0.1463
	3 layer, 32 filters, non-shuffled, grid-search initialization		0.9691	0.1051	0.0163	0.0609
	3 layer, 32 filters, non-shuffled, grid-search initialization, untrainable first layer		0.9647	0.1076	0.0158	0.0427

Table 4.57: The following table is the summary of results obtained from Alexis' work (Table 3.2) and our work. We compare the differences of his results using either 1 to 3 layers (with a variation of the number of filters in the first layer) and compare with our network with a replaced first layer using the Gabor filter bank that is either initialized with a random distribution or a grid-search. There are 2 variations of our work that contains 2 different levels of dropout. We have found earlier that adding dropout to our network reduces the performance of the network which continues the trend of using the Gabor filter in the first layer.

4.7 Summary of All Experiments

Experiment A

In our first experiment, we see whether the parameters within the Gabor filter impact the network's training or not by setting static filters in the first layer based on the Gabor filter. We find that the parameters that change the frequency (orientation of the filter) have the most impact on the images that it is training on. Those parameters are θ and ψ which have more impact on the network than the other 3 parameters in the Gabor filter (σ, λ, and γ). There are certain values that we have found that these parameters should not be (see Table 4.14). These values are mostly because of the way the 2-D formula works for the Gabor filter (Section 1.2.3).

Experiment B

In the second experiment we give these parameters more space within the filter space to train on. We initialize the Gabor filters bank using a range provided in Table 4.2 initializing one parameter at a time, while keeping the rest of the parameters static. From this test, we solidify our findings where we see that θ and ψ have the most impact on the network. We also found that with a different number of filters used, θ and ψ show no noticeable effect to the network but the other 3 parameters, σ, λ, and γ, give a better result using less filters. We concluded with using experiment A and B by saying that the parameters that largely affect the CNN are θ and ψ, however the other parameters should still exist and be non-zero.

Experiment C, D, and E

In the third, fourth, and fifth experiment, we get more serious with the Gabor filter bank and the CNN. Here, we use 2 different initialization methods for the Gabor filter bank and insert the bank in the first layer of the CNN. Then we give the CNN 3 different levels of dropout ($D = [0, 0.25, 0.50]$) and 3 different numbers of filters within the first layer ($F = [32, 64, 96]$). We have 18 different permutations or tests, but we compare the results in terms of initialization

methods as listed in Section 4.3. The datasets that we tested on are MNIST, CIFAR-10, and CIFAR-100. The datasets are shuffled in experiment C, not shuffled in experiment D, and lastly for experiment E, we use a non-shuffled dataset while disabling the ability to learn in the first layer for the CNN.

We find that when we add dropout to the CNN, the performance of the network decreases dramatically. For MNIST, it is not as noticeable and the drop is insignificant but for the other 2 datasets, the drop becomes more visible. We conclude by saying that when training with the Gabor filter in a CNN, adding dropout adds no benefit. In cases where it helps reduce overfitting, the accuracy is still reduced when training at the point of overfitting when compared to having no dropout added. For the most extreme cases when training on CIFAR-100, we see a drop in performance by almost half when compared to having no dropout at all.

We also find that increasing the number of filters within all datasets improves the network's performance. When testing on MNIST, we see a small insignificant boost, but for CIFAR-10 and CIFAR-100 we can see an improvement by up to 5% in terms of validation accuracy. This boost happens for all levels of dropout and for either initialization method used for the Gabor filter.

Using the random initialization method for the Gabor filter bank in the first layer is favoured when we are working the the MNIST dataset. Using the grid-search initialization method for the Gabor filter bank is favoured for both CIFAR-10 and CIFAR-100 datasets. Not only do CIFAR-10 and CIFAR-100 favour grid-search for the Gabor filter, it also gives a performance boost by up to 5% in terms of validation accuracy.

When we compare the results from looking at a shuffled versus a non-shuffled dataset, we cannot see any noticeable patterns when comparing the validation accuracy. When we compare using the trainable first layer versus not having a trainable first layer, we see a dip in the performance, although the results are still close. We can conclude by saying that the Gabor filter is strong and robust at extracting features within the images regardless of the image it receives.

Experiment F

In the final experiment, F, we use Pascual's architecture and his dataset as well. We compare what he found with his 3-layer network and compare it by substituting our Gabor filter bank in his first layer using the same hyperparameters as his architecture. The only change we do in order to compare directly is the initialization of the first layer. He used an Xavier distribution, we used the Gabor filter bank initialized either randomly or with a grid-search.

We found that when we add dropout, it reduces the performance of the network, just like what we have found in experiments C through E. Training on the rock dataset never overfits so there is no benefit in adding dropout to this Pascual's architecture.

Adding filters to the first layer using either initialization method offers a lower performance overall. This was also found in Pascual's work as well. We conclude with his statement here that adding more filters does not benefit the system and should be avoided. This is completely opposite to what we have found in our experiments but it could be because of how simple the architecture is using a 3 layer network. Using our findings from experiment B, we see that using less filters for σ, λ, and γ gives a higher performance in terms of validation accuracy. This could be the one of the reasons why increasing the number of filters decreases the performance.

Using either method of initialization for the Gabor filter does show a difference, but there is not enough data to say anything conclusive. The only thing we can say with the results we have is that either method is fine. Using a random initialization method for the Gabor filter can sometimes offer a higher performance in terms of validation accuracy, but can also offer a lower validation accuracy. Using a grid-search offers a consistent performance but doesn't high the highs like the random search.

Using a non-shuffled rock dataset, we gain more accuracy than when we are shuffling the dataset. This pattern is more clear using this rock dataset, but for the experiments from C through E, we cannot find any clear patterns. Freezing the first layer in the network reduces the performance which is also found in our experiments (C through E). Although the performance is reduced when we freeze the first layer, it is not a large performance drop, which shows that

the Gabor filter is very robust when extracting features.

Comparing our results to Pascual's results, we find that using the Gabor filter bank in the first layer rather than the Xavier distribution, we see a speed increase as the network converges much faster in the earlier phases of training (up to 10 to 20 epochs). Once the early phases of the training passes, Pascual's network trains reaches a higher accuracy with a consistently low loss.

Summary of All Experiments

In conclusion, using the Gabor filter as a feature extractor in the CNN can be very powerful and can match other networks that do not use the Gabor filter. Using dropout always reduces the performance, even when overfitting occurs during training. Adding filters to the network's first layer appears to be dataset-specific, but for MNIST, CIFAR-10, and CIFAR-100, we see a trend where if we increase the filters, the performance is boosted. The opposite happens in Pascual's rock dataset. Random initialized Gabor filters used in the first layer favours more simpler datasets like MNIST, and in some cases, like Pascual's rock dataset. Grid-search is more favoured for complex datasets like CIFAR-10 and CIFAR-100 and also provides a significant boost in performance over using a random search.

Chapter 5

Conclusion

5.1 Summary

Based off of the Zeiler's findings [26] in Figure 2.12, we see that some of the filters that layer 1 learns is the Gabor filter itself. The Gabor feature is a very strong feature extractor able to find features that other filters may not see due to the natural rotational capabilities of the filter. We take this example of the CNN learning these Gabor filters in the first layer and use it to see the benefits of how strong the Gabor filter truly is. We use the Gabor filter and create a filter bank and substitute the first layer of CNN with it instead. By skipping the notion of learning Gabor filters, we can see performance boosts in the early phases of training.

In chapter 2, we discover the revival of convulational neural networks and how they have become better over the length of 7 years. Since AlexNet's achievement which included Hinton's work on dropout [23], many other researchers studying CNNs have brought up what we use today in our image identification tasks. These researchers have improved the performance on obtaining accurate results on the ImageNet which is considered a very hard task. From this, we find other ways to improve features within the CNN.

There are many new or improved additions that helped improve neural networks for what they are today. For example, Hinton's work on dropout was extraordinary in reducing overfit-

ting. The surprising thing about dropout in our network we created is that dropout does not improve the performance of our network, even when overfitting occurred.

Another addition to our work is the usage of using randomization over grid-search for parameterization. Bergstra and Bengio [1] found that by using a randomization over grid-search for finding parameters within a large space, it was more optimal to find parameters using a random approach. We use this finding in our experiments and we see the trend where random search offers in some cases, higher performances, but also in other cases can reduce the performance when compared to a grid-search. The grid-search initialization method we use shows consistency with the results we obtained. In more complex datasets, we see that the grid-search parameterization of the Gabor filter offers a performance boost over using the random search. This was more effective in CIFAR-10 and CIFAR-100.

We study different initialization techniques involving the newest addition of the Gabor filter. The Gabor filter was created by Dennis Gabor in 1946 and the specializes in extracting features using different frequencies within the filter. The rotational aspect of the filter is able to extract very good features and our results show very promising results. In our experiments, we disable the ability for the Gabor filter bank to learn and achieves results similar to those when learning is allowed. This work we do is similar to the work that Chen used in his research on using the median filter.

In chapter 3, we have stated our software and hardware requirements for the thesis listing all of the details used in our experiments. We explain how we are using the MNIST, CIFAR-10, CIFAR-100, and Pascual's rock dataset, and how much data we are training from. We see that Pascual's rock dataset resembles MNIST in terms of simplicity, but has colour like CIFAR-10 and CIFAR-100.

We have described 2 architectures that we used throughout our experiments. The first is similar to the AlexNet architecture of 5 layers with dropout, and the second is Pascual's 3-layer network with dropout. We test using different dropout percentages, and using different number of filters within the first layer. The first layer of our network is changed to our Gabor

filter bank.

The Gabor filter bank is used with either Ozbulak's grid-search algorithm or a randomization for the parameters within the Gabor filter bank. We show how we use the Gabor filter bank within the CNN, where we insert the filter bank within the first layer of the CNN, in place of using other initialization methods (such as the Xavier distribution).

In our thesis, we provided 6 experiments listed as experiments A through F (sections 4.1, 4.2, 4.3, 4.4, 4.5, and 4.6) using the Gabor filter as an initialization technique for the CNN. The first 2 experiments, find that the Gabor parameters indeed impact the filter. This is based on the 2-D equation found in Section 1.2.3 and we find that in fact, some parameters cannot be zero. Some of the parameters cannot be within a range or otherwise the filter becomes useless. θ and ψ impact the Gabor filter the most because of the way they extract features. The other 3 parameters, σ λ, and γ show that as long as they are non-zero, they will be okay. There were some optimal ranges found in our tests, but there were no conclusive statements that we could say due to a small number of trials done.

In the third, fourth, and fifth experiment, we replace the first layer within our custom-made CNN with a Gabor filter layer, and initialize the Gabor layer with either a grid-search or a random search algorithm based on Bergstra and Bengio. Testing with the number of trials given in each experiment, we found on average that for simpler datasets, MNIST favoured using the random-search approach more than the grid-search. It also produced a small insignificant performance boost when training on MNIST. Grid-search was more favoured for the complex datasets such as CIFAR-10 and CIFAR-100 and also produced a significant performance boost in terms of validation accuracy for the network. Using more complex datasets (with more channels, color, and classes) drops the performance of our Gabor CNN but this is common for a simple CNN architecture.

The findings from these experiments show that whenever we add dropout, we lose a significant amount of performance when comparing with the same architecture but with no dropout included. In the case where we saw CIFAR-10 and in some cases CIFAR-100 overfitting, by

adding dropout it reduced the overfitting. The results from adding dropout however lowered the performance overall when compared to having no dropout. As a conclusive statement for dropout in our simple Gabor CNN, adding dropout is not necessary, adds no benefit, and should not be used.

Changing the number of filters for the Gabor filter bank shows that as we increase the filters, we increase the performance of the network. This is very clear in Figure 4.43a where it shows the increased number of filters, for all levels of dropout. This pattern was seen in all of our experiments except for the last experiment, F. We also saw that when we are training on MNIST, increasing the filters for the Gabor filter bank did not provide a significant performance boost when compared to training on CIFAR-10 and CIFAR-100. This could be that with more Gabor filters, more features could be extracted and hence provide a better result.

In comparison to using either initialization methods for the Gabor filter (grid-search versus random-search), we find that it is dataset-specific. Following what Bergstra and Bengio have found, we see that a randomization of parameters in the Gabor filter does indeed help produce higher results more than grid-search for the Gabor parameters. The downside to this, is that in some of the trials we run, the performance can score really low in terms of accuracy. In some cases, we see that the network cannot even begin training because the randomly initialized Gabor filter bank is using parameters that are ineffective or not useful at all. The grid-search initialized Gabor filter however is normally consistent in obtaining results. Furthermore, grid-search offers a performance boost for complex datasets like CIFAR-10 and CIFAR-100. Perhaps in future tests, we can use other complex datasets to see if the Gabor CNN is only useful for simpler datasets or not.

5.2 Summary of Contributions

The following contributions are as follows:

1. **Gabor filters can be used as a substitute instead of learning in CNNs for the first**

layer. We have found that the Gabor filter used as a substitution for the first layer in a CNN is roughly 5% less accurate when under the same hyperparameters, CNN architectures, and training parameters (epoch, trials, etc.) comparing to a CNN that is able to learn through training under normally used methods (i.e. Xavier uniform distribution is used to initialize the weights in the layers).

2. **The Gabor filter bank used in a CNN prefers simple datasets.** We have found that for simple datasets like MNIST, and Pascual's dataset, using a smaller number of filters in the Gabor filter bank results in better accuracy. For more complex datasets, adding more filters to the Gabor filter bank results in a higher performance.

3. **Gabor parameters do matter.** The parameters in the Gabor filter each have a different impact on training where certain values decrease the performance or increase performance. We found that θ and ψ have a higher impact on the training and have similar impact with one another. We also found that the other 3 parameters are similar to each other (σ, λ, and γ), where they cannot be a zero value. If they are non-zero, they behave similarly to each other when used in training for the CNN.

4. **The Gabor filter bank used in a CNN is best used on with objects that are similar (rocks) or objects with clear edges (numerical black/white digits).** The Gabor filter bank when substituted in the first layer of our custom 5 layer CNN shows that it is able to converge similarly to a normal CNN (initialized with Xavier uniform distribution in the first layer rather than our Gabor filter bank). It is especially stronger when training on simpler datasets with smaller number of classes, or objects with strong edges, and similar features (rocks) rather than complex datasets that are colourful, contain a large number of classes that are uniquely different from each other and also with features that are different from each object (cat versus airplane).

5. **The Gabor filter bank used in a CNN trains faster initially.** We show that the Gabor filter bank is able to train faster in the early phases of training by up to 10% in terms

of accuracy under the same amount of epochs. This can help convergence and reduce training time. We show that during training while using the Gabor filter bank, the initial learning period (up to 10 epochs of training) performs better than the Xavier distribution that is used in Pascual's network.

6. **Using a random initialized Gabor filter bank in the CNN is faster on simpler datasets.** Using the Gabor filter bank, we show that using a random-search or a grid-search for the parameters in the Gabor filter are dataset specific. For simpler datasets, random-search for the Gabor filter is more optimal while grid-search is more optimal for more complex datasets, especially because it gives a performance boost over using a random-search. Using random-search can provide higher results, but also lower results in some cases. Using a grid-search results in consistent results when compared to random-search.

7. **GCNNs using Dropout adds no benefit using the datasets we tested.** Adding dropout when using a Gabor filter bank in the first layer for a CNN is not necessary as it adds no benefit to the network. Even when there is visible evidence of overfitting (like training on CIFAR-10) and dropout should be used to reduce overfitting, when the network is finished training, the performance is lower than using the same network without dropout.

5.3 Future Work

We have found that the Gabor filter is a very strong contender for extracting features. By freezing the first layer and consequently using raw Gabor filters in the first layer of a CNN, we see that it is able to achieve results similar the Xavier distribution and also results where we enable the first layer to learn. Following our simple architecture using 5 layers, and in Pascual's architecture of 3 layers, we can extend the Gabor filter using it the architectures that have been created recently like DenseNet, GoogleNet, and find whether the Gabor filter can be seeded into these networks.

Seeing the results from working on Pascual's dataset, we have seen a speed increase for

training in the early epochs, which helps convergence speed. We could potentially create a more customized network where we use the Gabor filter in the earlier stages and later switch out the layer mid training. We have seen that in Pascual's dataset when we use our Gabor CNN, that it trained much faster but in the end resulted with a slightly lower accuracy.

Training on different datasets would give us a more clear picture of how the Gabor filter works as well. We have seen patterns emerging from the 4 datasets but in some cases, like working with Pascual's dataset, the patterns are dataset specific. Perhaps, by expanding our datasets using the same architecture we can see whether if the Gabor filter used in the first layer is dataset specific and not just a general filter bank we can use for all datasets.

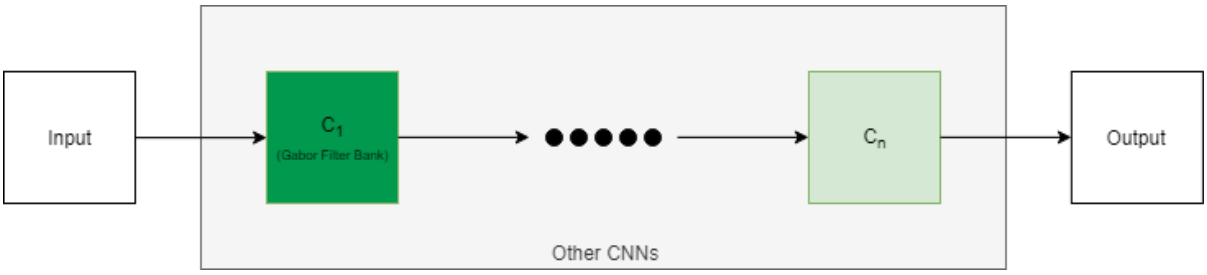


Figure 5.1: Future work of including the Gabor filter within present architectures such as GoogleNet, VGG-16/VGG-19, DenseNet, etc. Here, C1 is the first layer within the network and the other layers would remain the same as what they would be originally.

Bibliography

[1] James Bergstra and Yoshua Bengio. Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

[2] Andres Calderon, Sergio Roa, and Jorge Victorino. Handwritten Digit Recognition using Convolutional Neural Networks and Gabor filters. *International Congress of Computational Intelligence*, 2003.

[3] Konstantinos G Derpanis. Gabor filters. *York University*, 2007.

[4] D Gabor. Theory of communication. Part 1: The analysis of information. *Journal of the Institution of Electrical Engineers - Part III: Radio and Communication Engineering*, 93(26):429–441(12), nov 1946.

[5] Ross Girshick. Fast R-CNN. *Proceedings of the IEEE International Conference on Computer Vision*, 2015 Inter:1440–1448, 2015.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[7] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0, 2012.

[8] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-Janua:2261–2269, 2017.

[9] D. H. Hubel and T. N. Wiesel. Receptive Fields of Single Neurones in the Cat's Striate Cortex. *Journal of Physiology*, 148:574–591, 1959.

[10] Jiansheng Chen, Xiangui Kang, Ye Liu, and Z. Jane Wang. Median Filtering Forensics Based on Convolutional Neural Networks. *Ieee Signal Processing Letters*, 22(11):1849–1853, 2015.

[11] Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. *University of Toronto*, 44(8):1–60, 2009.

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *ImageNet Classification with Deep Convolutional Neural Networks*, pages 1097–1105, 2012.

[13] Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-Based Learning Applied to Document Recognition. *proc. OF THE IEEE*, 1998.

[14] Min Lin, Qiang Chen, and Shuicheng Yan. Network In Network. *International Conference on Learning Representations*, 2013.

[15] Shangzhen Luan, Baochang Zhang, Siyue Zhou, Chen Chen, Jungong Han, Wankou Yang, and Jianzhuang Liu. Gabor Convolutional Networks. *Pattern Recognition Letters*, 2018.

[16] Gökhan Özbulak, Hazm Kemal Ekenel, and Assoc Prof. Gabor Initialized Convolutional Neural Networks for Transfer Learning. *Signal Processing and Communications Application Conference*, 26, 2018.

[17] A. D. Pascual. Autonomous and Real Time Classification of Rock Images. *Electronic Thesis and Dissertation Repository*, 6059, 2019.

[18] N Petkov and M.B. Wieling. Gabor filter for image processing and computer vision, 2008.

[19] Syed Shakib Sarwar, Priyadarshini Panda, and Kaushik Roy. Gabor Filter Assisted Energy Efficient Fast Learning Convolutional Neural Networks. *IEEE ACM International Symposium on Low Power Electronics and Design*, pages 1–6, 2017.

[20] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. *International Conference on Learning Representations*, 2013.

[21] Lei Shu. *Automatic Image Classification for Planetary Exploration*. PhD thesis, Western University, 2018.

[22] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *International Conference on Learning Representations*, 2014.

[23] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfittin. *Journal of Machine Learning Research 15*, 15:1929–1958, 2014.

[24] Christian Szegedy, Scott Reed, Pierre Sermanet, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *IEEE CVPR*, 2014.

[25] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *NIPS*, 27, 2014.

[26] Matthew D Zeiler and Rob Fergus. Visualizing and Understanding Convolutional Networks. *European Conference on Computer Vision*, 2014.

[27] Matthew D. Zeiler, Graham W. Taylor, and Rob Fergus. Adaptive deconvolutional networks for mid and high level feature learning. *Proceedings of the IEEE International Conference on Computer Vision*, pages 2018–2025, 2011.

Appendix A

Building the Keras Environment

A.1 Development Platform

A.1.1 Hardware

CPU	Intel i8700k 6-core processor @ 3.7GHz
GPU	MSI Seahawk 1080Ti @ 1600Mhz with 11GB GDDR5X
RAM	32GB @ 3200Mhz Dual Channel
OS	Windows 10

A.1.2 Software

Python Version	Anaconda 3.6
Deep Learning Framework	Keras with Tensorflow 1.9.0 backend
Programming Language	Python 3.6
GPU Language	Cuda ToolKit 9.0 with CuDNN 7.1.4

A.1.3 Setup Environment

The following environment that I developed my code on can be found on my GitHub. Here is the link to the master's thesis which will provide the thesis and the code for all results obtained throughout the thesis.

1. Clone the GitHub repository **https://github.com/longmpham/mastersthesis.git** or download the files you need if cloning is not your first choice.

2. Follow the directory "**./Installations/Install Keras with Anaconda/spec-file.txt**" to obtain the specifications sheet for the Python Environment used in Anacondda 3.6.

3. If you would like to create the environment, please follow the instructions provided in the same directory. You will need to install Python, Cuda, CuDNN, Keras, Tensorflow. These are the major ones.

To run the programs used to obtain the results, please follow the instructions below:

1. Go to the Code directory

2. To use the GCNN code, go deeper into the directory following the GCNN directory. There you will find files labelled as "**long_gcnn_[dataset].py**" and "**gabor_init.py**".

3. In most of the directories, you will need to adjust the hyperparameters and match the file directories, and setup the hyperparameters such as filter size, dropout, initialization method, etc. **Please make sure you change your hyperparameters to get proper results! Do not forget!**

4. To run the code, type in your Anaconda environment (or preferred Python environment) the following: "**python long_gcnn_[dataset].py**". This file will depend on the "**gabor_init.py**" so make sure you set the file directories correctly. I cannot stress this enough. For some reason, Python does not allow adding strings to function calls (that I know of).

5. Let it run (hardware dependent)! You can also limit your GPU's processing power or limit the number of GPUs used. This can be controlled at the top of the code.

Curriculum Vitae

Name:	Long Pham
Post-Secondary Education and Degrees:	Western University London, ON, CA 2012 - 2016 B.E.Sc.
Related Work Experience:	Teaching Assistant Western University 2016 - 2019