Pattern Recognition and Machine Learning
Convolutional Neural Networks

Pavan Gurudath

March 4, 2018

Abstract

Convolutional Neural Networks (ConvNets or CNNs) is a category of Neural Networks that are used in areas such as image recognition and classification. It is a modular sort of classifier. One of the eye-opening developments in Machine Learning/Computer vision is that a cascade or chain of very simple image processing operations can outperform more complicated state-of-the-art object recognition algorithms. Of course, the specific image processing operations used have to be well-chosen. The real magic in a CNN is learning the underlying convolution filters. A certain number of layers are chained together in order to form what is known as deep layer. These long chains are formed by building small and simple computational building blocks. Much of the heavy computational and big-data requirements for training CNNs (so-called deep learning) are concentrated on this learning process. Among various networks, Convolutional neural networks has demonstrated high performance on image classification. CNNs are successfully used in various situations such as identifying faces, objects and traffic signs for autonomous driving, powering vision in robots.

In this project, we utilise the Wallpaper dataset that was used in Project-2 in order to build networks that can classify the different classes of wallpaper patterns.

Contents

1 Introduction **3**

2 Approach **4**
2.1 Data 4
2.1.1 Original Dataset 4
2.1.2 Augmented Dataset 5
2.1.3 AlexNet Dataset 5
2.1.4 Resized Augmented Dataset 6
2.2 Methods 7
2.2.1 CNN Architecture 7
2.2.2 Training and Testing of CNN 9
2.2.3 Augmenting the Training and Testing data 9
2.2.4 Wide Network 11
2.2.5 Skinny Network 12

3 Results **13**
3.1 Training and Testing of Original Dataset 13
3.2 Augmentation of Dataset 22
3.3 Wide network 27
3.3.1 Wide network on Augmented Dataset 27
3.3.2 Wide Network on Original Dataset 32
3.4 Skinny network 37
3.4.1 Skinny network on Augmented Dataset 37
3.4.2 Skinny network on Original Dataset 41

4 Extra Credits **45**
4.1 Transfer Learning using AlexNet 45
4.2 t-SNE multidimensional reduction on fully connected layer activations of network trained on the augmentations 49
4.3 Visualization of first layer of filters 51

5 Conclusion **52**

1 Introduction

The concept of classification refers to the identification of the category of a new observation. Image classification is the task of taking an input image and outputting a class to which that particular input image belongs to. The Convolutional Neural Network performs this task of learning which class a type of image belongs to and thereby being able to predict the class of any new input image. In this project, we perform the tasks of training the Wallpaper dataset whose properties have been discussed in Section 2.1. While the given dataset includes 17,000 images, a CNN requires a larger dataset and more variety of the same class in the type of images that is being inputted, in order for it to be able to predict the new input image's class.

This project lets us explore the way a CNN behaves and to understand its functioning. We train the network on different variations of its hidden layers and on different datasets. In this project, the main chain of events include the following:

- To understand the importance of the **learning parameter** α using the given network and dataset.
- To **augment** the given dataset in order to create a larger dataset.
- To build and train on two networks, **wide and skinny**, using the augmented dataset.
- To train the new networks on the original dataset.

Due to the unavailability of GPU access and the inability of the systems in Westgate to stay awake for not more than 10minutes, the networks were trained under a constraint of borrowing other's laptop. Therefore, the network parameters for the bigger dataset could not be tuned to achieve maximum accuracy. However, the conclusions could be drawn.

2 Approach

The section 2 details the approach that has been taken in order to achieve the aforementioned objectives in section 1. Section 4 details the methods as well as the results of the extra credit questions that were included in the project description. Section 2.1 introduces the properties of the *Wallpaper* dataset that has been used in this project and also describes the way in which the datasets have been created for different requirements of different networks. Section 2.2 contains all the methods that are necessary to perform this project. Section 2.2.1 details the layers of the Convolutional Neural Network. Section 2.2.4 and Section 2.2.5 details the parameters that were used to build the Wide and Skinny network respectively.

2.1 Data

This project makes use of the Wallpaper dataset that was used in Project 2 of this course. The datasets are loaded into test and train category. For any CNN, the input is that of an image, and the given dataset consists of 17000 images. For various methods, different datasets of different sizes and numbers had to be created and are detailed in the following subsections. In all the datasets, the classification groups are as follows:

P1	**P2**	**PM**
PG	**CM**	**PMM**
PMG	**PGG**	**CMM**
P4	**P4M**	**P4G**
P3	**P3M1**	**P31M**
P6	**P6M**	

2.1.1 Original Dataset

The original wallpaper dataset that was given for this project could be described as follows:

The data in this set consists of images belonging to 17 different Wallpaper Groups.
Number of Classes: 17
Number of images per class : 1000

Training parameters: .data\wallpapers\train
Number of training observations: 17000
Number of samples per class in train data-set: 1000
Size of images: 256 $\times$256

Testing parameters: .data\wallpapers\test
Number of testing observations: 17000
Number of samples per class in test data-set: 1000
Size of images: 256 $\times$256

This given dataset is small dataset for a CNN and it has been augmented to using the given *augment.m* MATLAB file. The images are rotated, scaled and translated by certain number of pixels in the x and y direction. The augmented dataset is detailed in 2.1.2

2.1.2 Augmented Dataset

In order to build the Wide and Skinny network, we build the following dataset.
Number of Classes: 17
Number of images per Class: 5000
Features of the dataset:

Training parameters: .data\wallpapers\train_aug
Number of training observations: 85000
Number of samples per class in train data-set: 5000
Number of samples per original image in train data-set: 5
Size of images: 128 $\times$128

Testing parameters: .data\wallpapers\test_aug
Number of testing observations: 85000
Number of samples per class in test data-set: 5000
Number of samples per original image in train data-set: 5
Size of images: 128 $\times$128

2.1.3 AlexNet Dataset

In order to perform the extra credit of AlexNet, we create the following dataset. We create this new dataset because AlexNet was itself built on a $227 \times 227 \times 3$ set of images and its input layer requires this configuration.
Number of Classes: 17
Number of images per Class: 5000
Features of the dataset:

Training parameters: .data\wallpapers\train_alex
Number of training observations: 85000
Number of samples per class in train data-set: 5000
Number of samples per original image in train data-set: 5
Size of images: 227 $\times$227 $\times$ 3

Testing parameters: .data\wallpapers\test_alex
Number of testing observations: 85000
Number of samples per class in test data-set: 5000
Number of samples per original image in train data-set: 5
Size of images: 227 $\times$227 $\times$ 3

2.1.4 Resized Augmented Dataset

In order to perform the task of training the Wide and Skinny network from the original dataset, we create a new dataset of 128×128 since our Wide and Skinny network were built on this configuration for the sake of computational efficiency. Therefore we have the new dataset as follows:
Number of Classes: 17
Number of images per Class: 1000
Features of the dataset:

Training parameters: .data\wallpapers\train_128
Number of training observations: 17000
Number of samples per class in train data-set: 1000
Number of samples per original image in train data-set: 1
Size of images: 128 $\times$128

Testing parameters: .data\wallpapers\test_128
Number of testing observations: 17000
Number of samples per class in test data-set: 1000
Number of samples per original image in train data-set: 1
Size of images: 128 $\times$128

2.2 Methods

2.2.1 CNN Architecture

A Convolutional Network is a sequence of layers, and every layer of a ConvNet transforms one volume of activations to another through a differentiable function. There are three main types of layers to build ConvNet architectures: **Convolutional Layer**,**Pooling Layer**, and **Fully-Connected Layer**. These layers are stacked to form a full Convolutional Network architecture.

Every convolutional network would consist of the following:

- **Input:** This layer holds the raw pixel values of the image. The image could be of any size and is of the form $N \times M \times D$, where N and M are the size of the image and D is the number of channels. Channels is just a word to describe the number of values associated with each 2D spatial pixel location in an image, so, for example, a grayscale image has 1 channel whereas a color image has 3 channels (red, green, blue).

- **Conv:** This layer is the first layer after feeding in the input for any CNN. It computes the output of neurons, each computing a dot product between the weights and a small region that are connected to in the input volume. These small regions are known as *filters*. After sliding the filter over all the locations, we obtain an output matrix of numbers, which is referred to as an activation map or feature map.

- **ReLU:** ReLU stands for Rectified Linear Unit, but despite the fancy name it is just a thresholding operation where any negative numbers in the input become 0 in the output. If the input is an array of size $N \times M \times D$, then the output is an array of the same size with each value in the output array is defined as

$$output(i, j, k) = max(input(i, j, k), 0)$$

- **Pool:** This layer is responsible for the downsizing operation along the spatial dimensions $i.e.$width and height. Reducing the dimensionality of the images is a necessity in convnet since were dealing with high dimensional data and a lot of filters, which implies that we will have huge amount of parameters in our convnet. In this layer, we define the stride $i.e.$the number of pixels by which the filter should move across in order to obtain the summary of that filter. This summarization of the filter could be the average, maximum or minimum value of that patch. In our network, we use the maxPool which simply takes the maximum of the patch from the matrix.

- **Fully-Connected:** If the input is an array of size $N \times M \times D1$ and output is an array of size $1 \times 1 \times D2$ then the fully connected layer applies a filter bank of D2 linear filters and D2 scalar bias values to compute output values. However, unlike convolution layers, each filter is of the same size as that of an input image, and is applied in a way that computes a single, scalar valued output. Because there are D2 filters and bias values, the output image will contain D2 scalar values. These D2 values corrosponds to the class scores.

- **Softmax:** The softmax takes a vector of arbitrary real numbers (here class scores) and converts them into numbers that can be viewed as probabilities *i.e.* they will all lie between 0 and 1 and sum up to 1. If the input is an array of size $1 \times 1 \times D$, then the output is of the same size and is calculated as follows:

$$output(1,1,k) = \frac{exp(\ln(1,1,k)-\alpha)}{\sum_{k=1}^{D} exp(\ln(1,1,k)-\alpha)}$$

The figure 1 depicts the various layers for an example case.

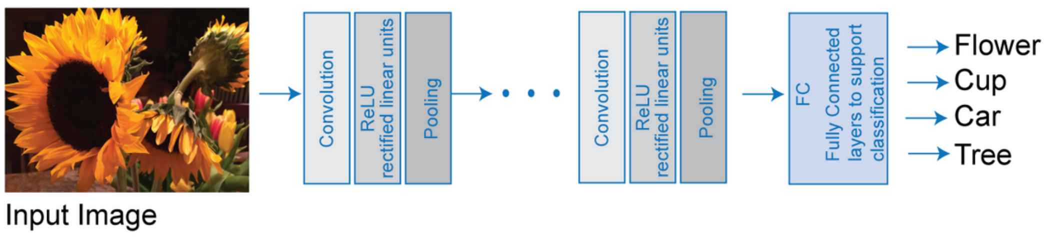
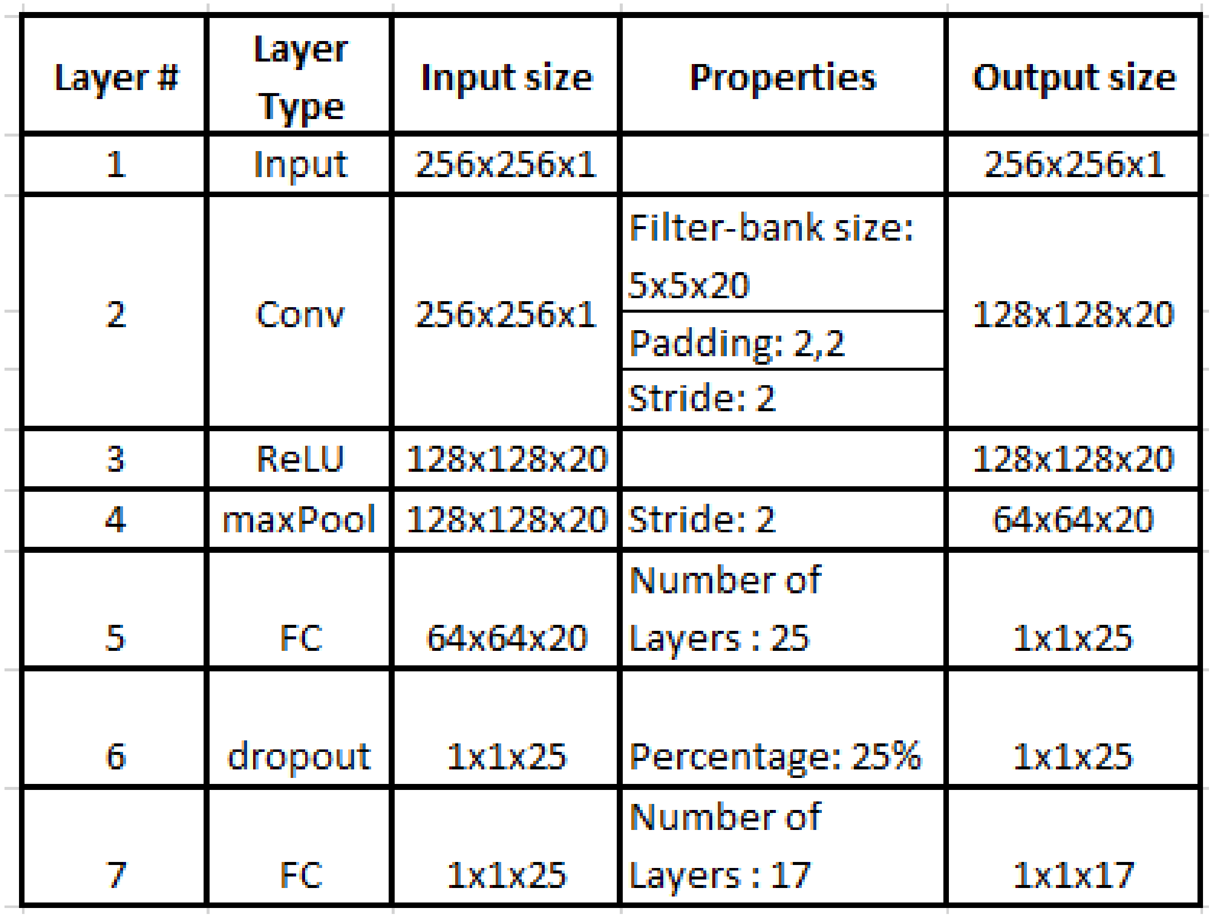


Figure 1: Convolutional Neural Network Layer representation

2.2.2 Training and Testing of CNN

In this case, we used the original dataset as listed in Section 2.1.1 that was provided to us along with the startercode. The conv network consisted of the following layers, with each layer performing the following functions as shown in Fig 2. Using this network, the original dataset was trained and tested by varying the learning parameter $\alpha = 1 \times 10^{-4}$ and $\alpha = 5 \times 10^{-4}$.



Layer #	Layer Type	Input size	Properties	Output size
1	Input	256x256x1		256x256x1
2	Conv	256x256x1	Filter-bank size: 5x5x20	128x128x20
			Padding: 2,2	
			Stride: 2	
3	ReLU	128x128x20		128x128x20
4	maxPool	128x128x20	Stride: 2	64x64x20
5	FC	64x64x20	Number of Layers : 25	1x1x25
6	dropout	1x1x25	Percentage: 25%	1x1x25
7	FC	1x1x25	Number of Layers : 17	1x1x17

Figure 2: Starter-code CNN network

2.2.3 Augmenting the Training and Testing data

Since the number of training and testing data are too small for a CNN, the data is augmeneted. This augmentation is done using the code that was provided. Every file from each of the corresponding folders of the train and test data were called and passed through the *augment.m* function. These files were augmented such that the input image were rotated, scaled as well as translated in both x and y direction. This was repeated 5 times and were saved under the corrosponding folders to produce a dataset as detailed in 2.1.2. The histogram of the rotation, scaling and translation are as shown in Fig **??**, **??** and **??**. Using the new augmented dataset, we perform the network training of Wide and Skinny as detailed in section 2.2.4 and 2.2.5 respectively.

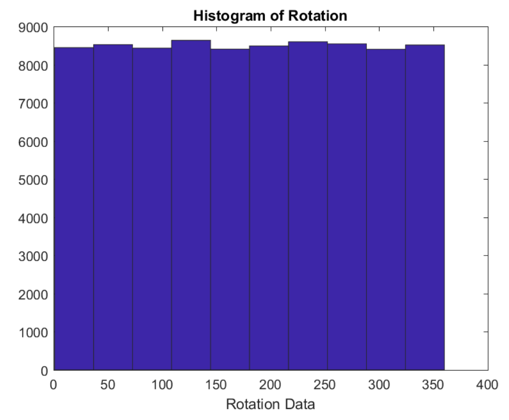


Figure 3: Histogram of Rotation

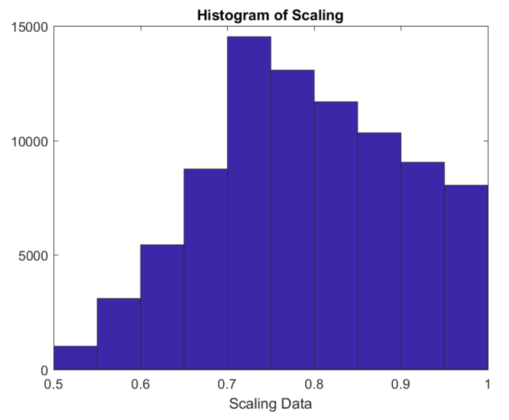


Figure 4: Histogram of Scaling

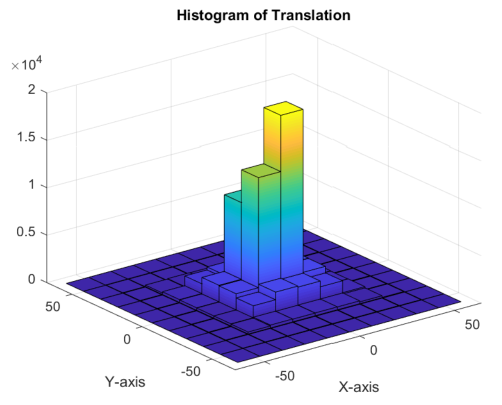
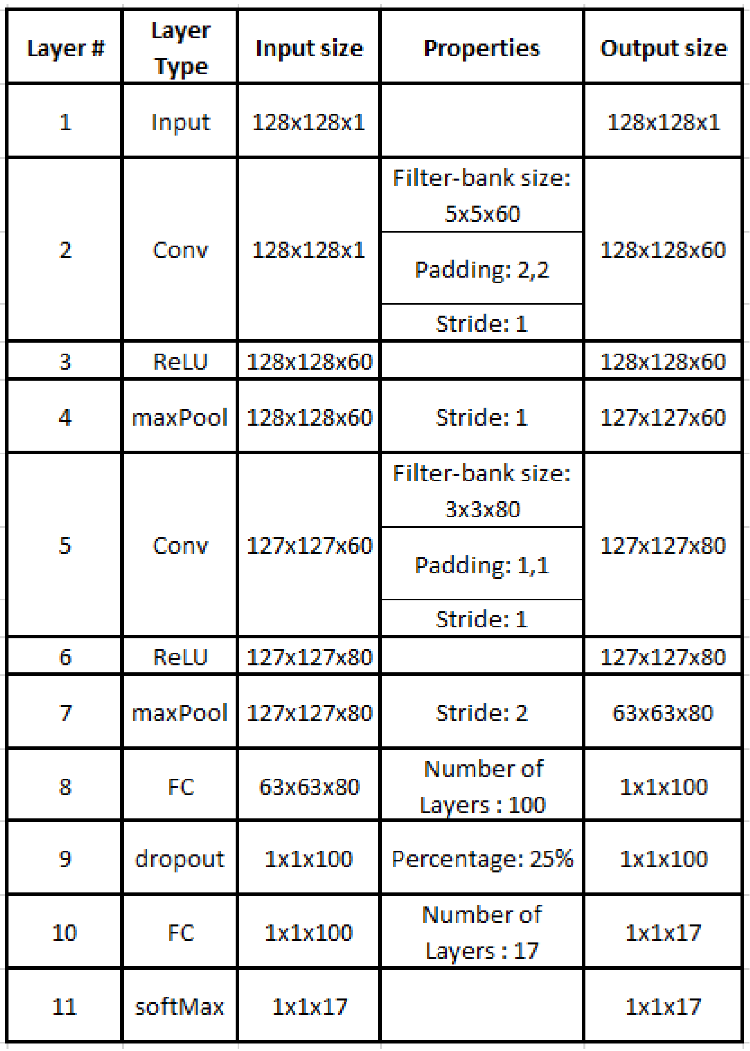


Figure 5: Histogram of Translation

2.2.4 Wide Network

Now using the augmented dataset of 85,000 images we create a new network known as the Wide Network. Wide network is basically a network when in comparison to the starter code, has fewer layers but more filters. The conv network consisted of the following layers, with each layer performing the following functions as shown in Fig 6. Using this network, the augmented dataset was trained and tested. Once the network was trained and after obtaining the necessary accuracies and the confusion matrix and the classification table, we load the workspace onto the command window and run it on the original dataset. We see that there is an improved accuracy as detailed in the results section. This network was trained with the parameters as follows:

$\alpha = 1 \times 10^{-3}$
Number of epochs = 10
batch size = 100



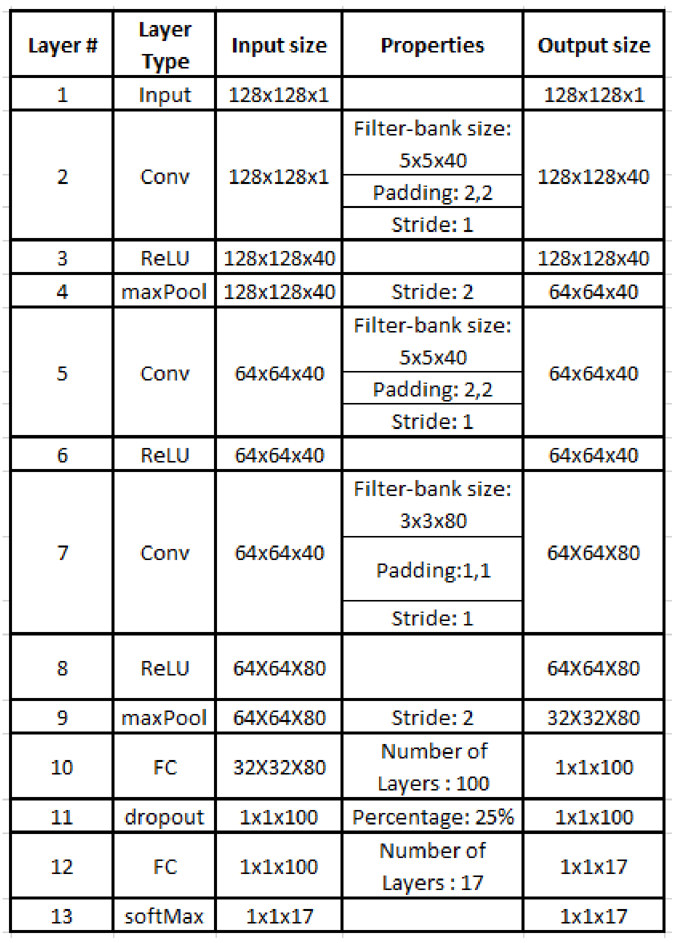
Layer #	Layer Type	Input size	Properties	Output size
1	Input	128x128x1		128x128x1
2	Conv	128x128x1	Filter-bank size: 5x5x60 Padding: 2,2 Stride: 1	128x128x60
3	ReLU	128x128x60		128x128x60
4	maxPool	128x128x60	Stride: 1	127x127x60
5	Conv	127x127x60	Filter-bank size: 3x3x80 Padding: 1,1 Stride: 1	127x127x80
6	ReLU	127x127x80		127x127x80
7	maxPool	127x127x80	Stride: 2	63x63x80
8	FC	63x63x80	Number of Layers : 100	1x1x100
9	dropout	1x1x100	Percentage: 25%	1x1x100
10	FC	1x1x100	Number of Layers : 17	1x1x17
11	softMax	1x1x17		1x1x17

Figure 6: Wide Network

2.2.5 Skinny Network

Using the augmented dataset of 85,000 images we create a new network known as the Skinny Network. Skinny network is basically a network when in comparison to the starter code, has more layers but not many filters. The conv network consisted of the following layers, with each layer performing the following functions as shown in Fig 7. Using this network, the augmented dataset was trained and tested. Once the network was trained and after obtaining the necessary accuracies and the confusion matrix and the classification table, we load the workspace onto the command window and run it on the original dataset. We see that there is an improved accuracy as detailed in the results section. This network was trained with the parameters as follows:

$\alpha = 1 \times 10^{-3}$
Number of epochs = 10
batch size = 100



Layer #	Layer Type	Input size	Properties	Output size
1	Input	128x128x1		128x128x1
2	Conv	128x128x1	Filter-bank size: 5x5x40 Padding: 2,2 Stride: 1	128x128x40
3	ReLU	128x128x40		128x128x40
4	maxPool	128x128x40	Stride: 2	64x64x40
5	Conv	64x64x40	Filter-bank size: 5x5x40 Padding: 2,2 Stride: 1	64x64x40
6	ReLU	64x64x40		64x64x40
7	Conv	64x64x40	Filter-bank size: 3x3x80 Padding:1,1 Stride: 1	64X64X80
8	ReLU	64X64X80		64X64X80
9	maxPool	64X64X80	Stride: 2	32X32X80
10	FC	32X32X80	Number of Layers : 100	1x1x100
11	dropout	1x1x100	Percentage: 25%	1x1x100
12	FC	1x1x100	Number of Layers : 17	1x1x17
13	softMax	1x1x17		1x1x17

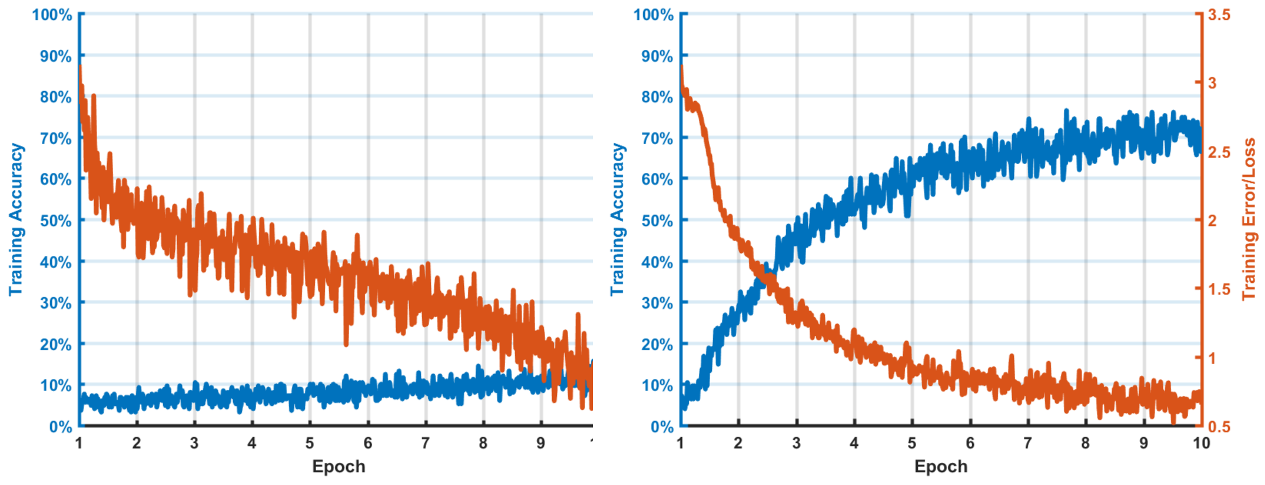
Figure 7: Skinny Network

3 Results

This section contains all the results that were tabulated.

3.1 Training and Testing of Original Dataset

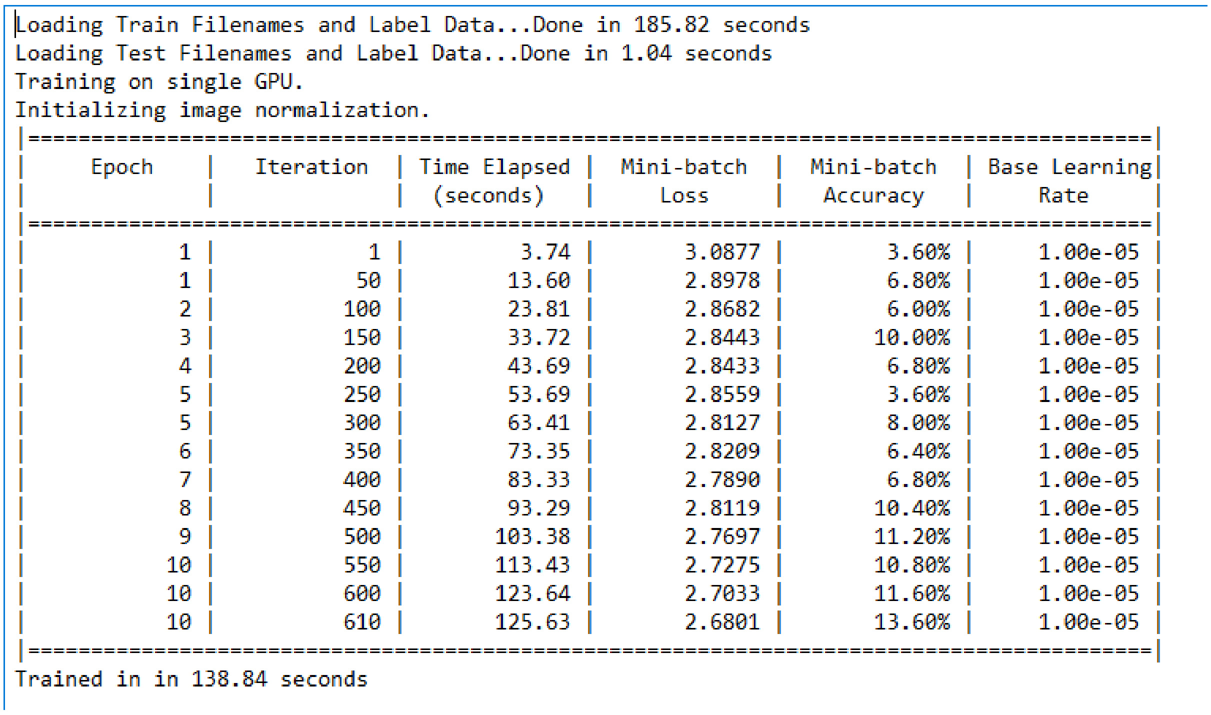
After running the code with the parameters of $\alpha_1 = 1 \times 10^{-5}$ and $\alpha_2 = 5 \times 10^{-4}$, we can clearly see that as the learning parameter is increased, the accuracy of the train, test and validation have significantly increased. This means that the rate equal to α_1 is very less and the network takes small steps to try to achieve the global minima. If the number of epochs was increased significantly for α_1, then the accuracy would increase to match that of α_2 but it would take a lot of time. The figures shown below further illustrate.



(a) $\alpha_1 = 1 \times 10^{-5}$ (b) $\alpha_2 = 5 \times 10^{-4}$

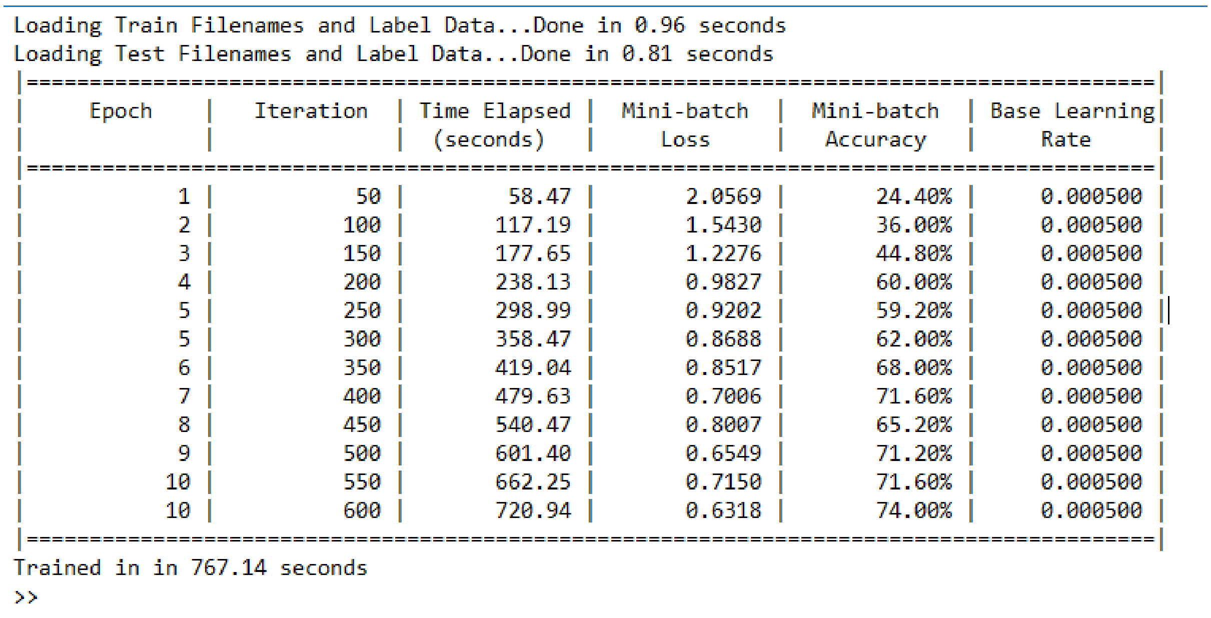
Figure 8: Error plot of CNN on original Dataset for different α

It can be seen from Fig 8a that for this learning rate, the training accuracy increases very little and does not rise as significantly as seen in 8b. This network was run for 10 epochs and the following tables were generated. Fig 9 is for α_1 while fig 10 is for the training on α_2.



```
Loading Train Filenames and Label Data...Done in 185.82 seconds
Loading Test Filenames and Label Data...Done in 1.04 seconds
Training on single GPU.
Initializing image normalization.
|========================================================================================|
|     Epoch     |   Iteration   | Time Elapsed |  Mini-batch  |  Mini-batch  | Base Learning|
|               |               |  (seconds)   |     Loss     |   Accuracy   |     Rate     |
|========================================================================================|
|             1 |             1 |         3.74 |       3.0877 |        3.60% |     1.00e-05 |
|             1 |            50 |        13.60 |       2.8978 |        6.80% |     1.00e-05 |
|             2 |           100 |        23.81 |       2.8682 |        6.00% |     1.00e-05 |
|             3 |           150 |        33.72 |       2.8443 |       10.00% |     1.00e-05 |
|             4 |           200 |        43.69 |       2.8433 |        6.80% |     1.00e-05 |
|             5 |           250 |        53.69 |       2.8559 |        3.60% |     1.00e-05 |
|             5 |           300 |        63.41 |       2.8127 |        8.00% |     1.00e-05 |
|             6 |           350 |        73.35 |       2.8209 |        6.40% |     1.00e-05 |
|             7 |           400 |        83.33 |       2.7890 |        6.80% |     1.00e-05 |
|             8 |           450 |        93.29 |       2.8119 |       10.40% |     1.00e-05 |
|             9 |           500 |       103.38 |       2.7697 |       11.20% |     1.00e-05 |
|            10 |           550 |       113.43 |       2.7275 |       10.80% |     1.00e-05 |
|            10 |           600 |       123.64 |       2.7033 |       11.60% |     1.00e-05 |
|            10 |           610 |       125.63 |       2.6801 |       13.60% |     1.00e-05 |
|========================================================================================|
Trained in in 138.84 seconds
```

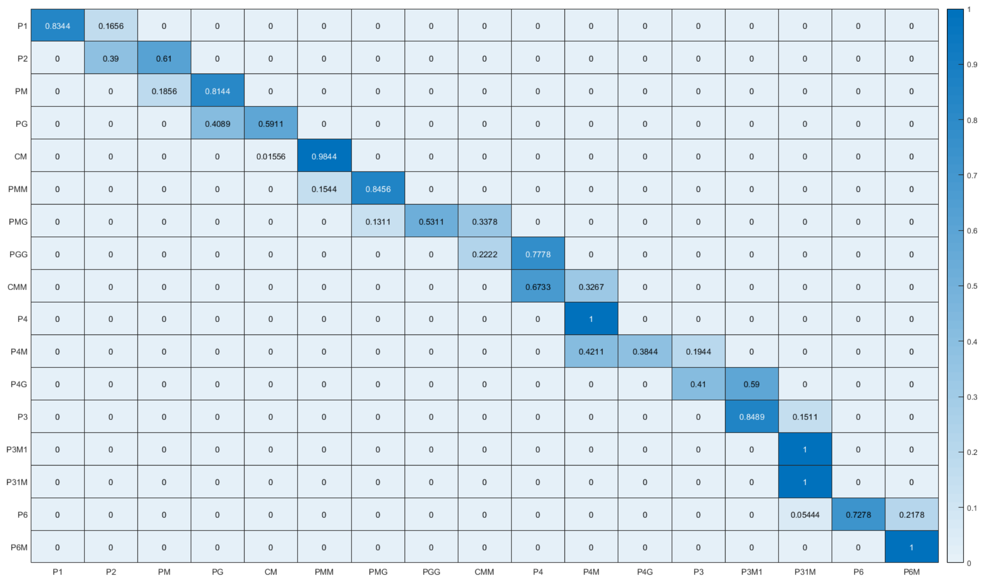
Figure 9: $\alpha_1 = 1 \times 10^{-5}$



```
Loading Train Filenames and Label Data...Done in 0.96 seconds
Loading Test Filenames and Label Data...Done in 0.81 seconds
|========================================================================================|
|     Epoch     |   Iteration   | Time Elapsed |  Mini-batch  |  Mini-batch  | Base Learning|
|               |               |  (seconds)   |     Loss     |   Accuracy   |     Rate     |
|========================================================================================|
|             1 |            50 |        58.47 |       2.0569 |       24.40% |     0.000500 |
|             2 |           100 |       117.19 |       1.5430 |       36.00% |     0.000500 |
|             3 |           150 |       177.65 |       1.2276 |       44.80% |     0.000500 |
|             4 |           200 |       238.13 |       0.9827 |       60.00% |     0.000500 |
|             5 |           250 |       298.99 |       0.9202 |       59.20% |     0.000500 ||
|             5 |           300 |       358.47 |       0.8688 |       62.00% |     0.000500 |
|             6 |           350 |       419.04 |       0.8517 |       68.00% |     0.000500 |
|             7 |           400 |       479.63 |       0.7006 |       71.60% |     0.000500 |
|             8 |           450 |       540.47 |       0.8007 |       65.20% |     0.000500 |
|             9 |           500 |       601.40 |       0.6549 |       71.20% |     0.000500 |
|            10 |           550 |       662.25 |       0.7150 |       71.60% |     0.000500 |
|            10 |           600 |       720.94 |       0.6318 |       74.00% |     0.000500 |
|========================================================================================|
Trained in in 767.14 seconds
>>
```

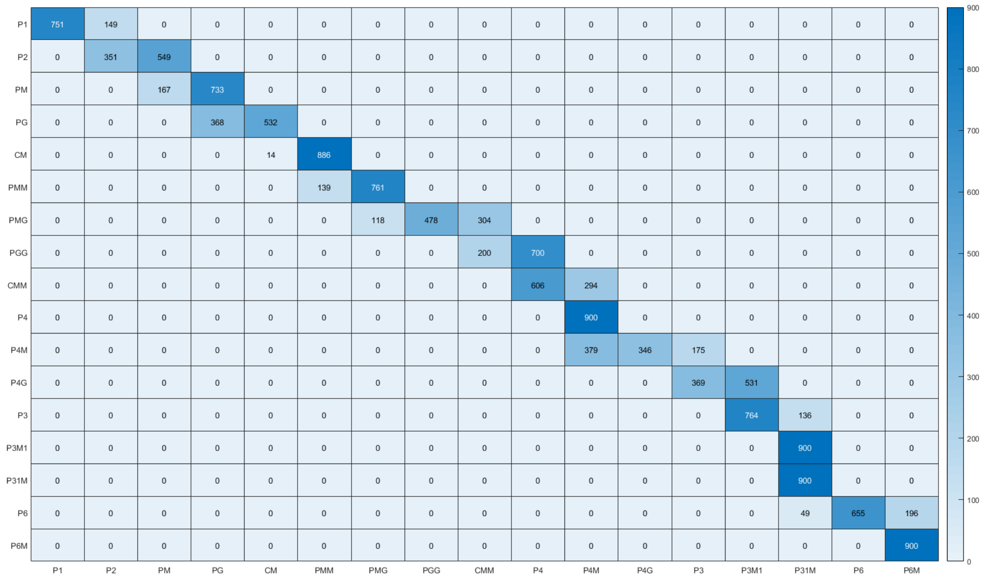
Figure 10: $\alpha_2 = 5 \times 10^{-4}$

The classification and confusion matrices for $\alpha_1 = 1 \times 10^{-5}$ are as follows for training in fig 11 and fig 12, testing in fig 13 and fig 14 and validation in fig 15 and fig 16 respectively.



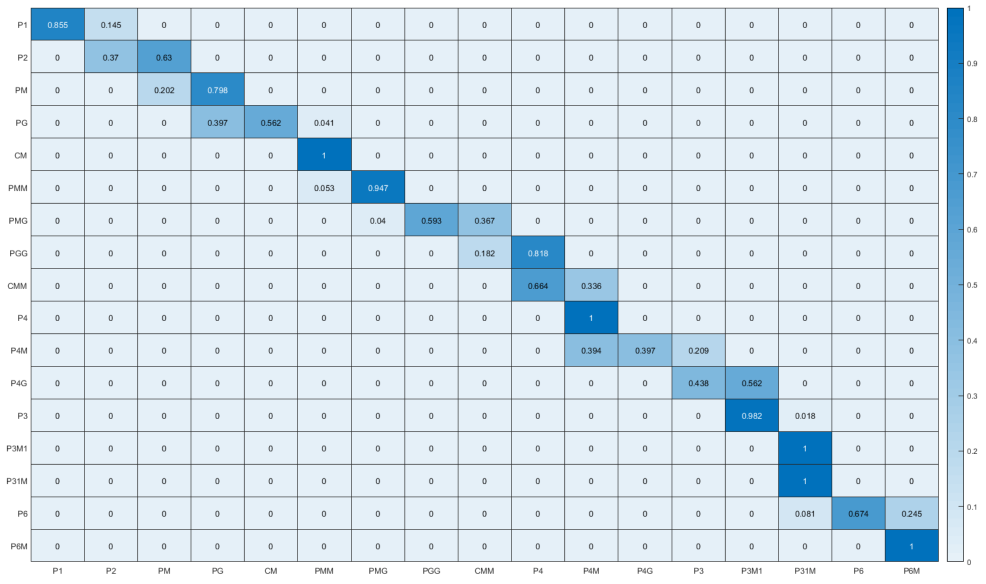
	P1	P2	PM	PG	CM	PMM	PMG	PGG	CMM	P4	P4M	P4G	P3	P3M1	P31M	P6	P6M
P1	0.8344	0.1656	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P2	0	0.39	0.61	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PM	0	0	0.1856	0.8144	0	0	0	0	0	0	0	0	0	0	0	0	0
PG	0	0	0	0.4089	0.5911	0	0	0	0	0	0	0	0	0	0	0	0
CM	0	0	0	0	0.01556	0.9844	0	0	0	0	0	0	0	0	0	0	0
PMM	0	0	0	0	0	0.1544	0.8456	0	0	0	0	0	0	0	0	0	0
PMG	0	0	0	0	0	0	0.1311	0.5311	0.3378	0	0	0	0	0	0	0	0
PGG	0	0	0	0	0	0	0	0	0.2222	0.7778	0	0	0	0	0	0	0
CMM	0	0	0	0	0	0	0	0	0	0.6733	0.3267	0	0	0	0	0	0
P4	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
P4M	0	0	0	0	0	0	0	0	0	0	0.4211	0.3844	0.1944	0	0	0	0
P4G	0	0	0	0	0	0	0	0	0	0	0	0	0.41	0.59	0	0	0
P3	0	0	0	0	0	0	0	0	0	0	0	0	0	0.8489	0.1511	0	0
P3M1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
P31M	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
P6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.05444	0.7278	0.2178
P6M	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 11: Classification Matrix for Training ($\alpha_1 = 1 \times 10^{-5}$)



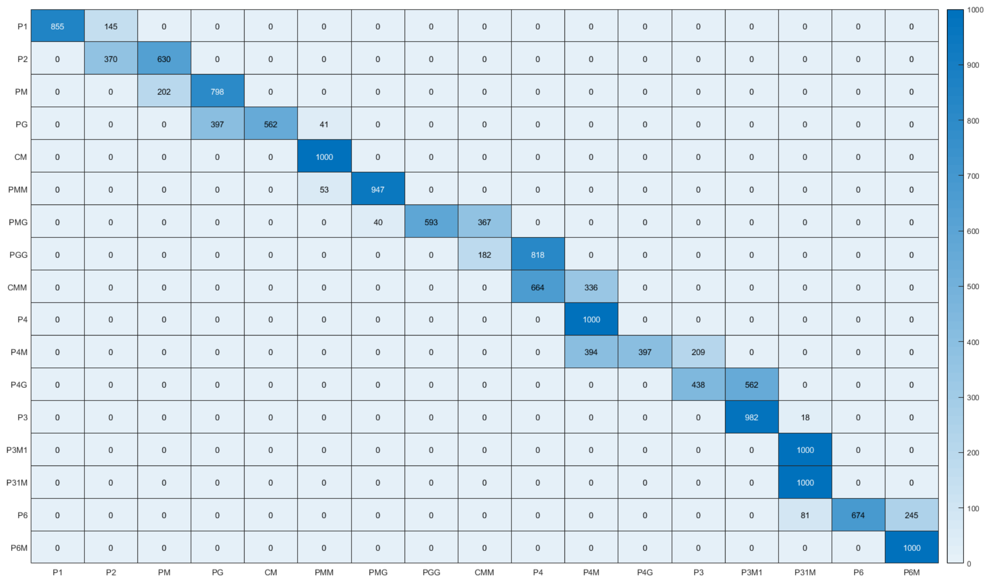
	P1	P2	PM	PG	CM	PMM	PMG	PGG	CMM	P4	P4M	P4G	P3	P3M1	P31M	P6	P6M
P1	751	149	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P2	0	351	549	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PM	0	0	167	733	0	0	0	0	0	0	0	0	0	0	0	0	0
PG	0	0	0	368	532	0	0	0	0	0	0	0	0	0	0	0	0
CM	0	0	0	0	14	886	0	0	0	0	0	0	0	0	0	0	0
PMM	0	0	0	0	0	139	761	0	0	0	0	0	0	0	0	0	0
PMG	0	0	0	0	0	0	118	478	304	0	0	0	0	0	0	0	0
PGG	0	0	0	0	0	0	0	0	200	700	0	0	0	0	0	0	0
CMM	0	0	0	0	0	0	0	0	0	606	294	0	0	0	0	0	0
P4	0	0	0	0	0	0	0	0	0	0	900	0	0	0	0	0	0
P4M	0	0	0	0	0	0	0	0	0	0	379	346	175	0	0	0	0
P4G	0	0	0	0	0	0	0	0	0	0	0	0	369	531	0	0	0
P3	0	0	0	0	0	0	0	0	0	0	0	0	0	764	136	0	0
P3M1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	900	0	0
P31M	0	0	0	0	0	0	0	0	0	0	0	0	0	0	900	0	0
P6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	49	655	196
P6M	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	900

Figure 12: Confusion Matrix for Training ($\alpha_1 = 1 \times 10^{-5}$)



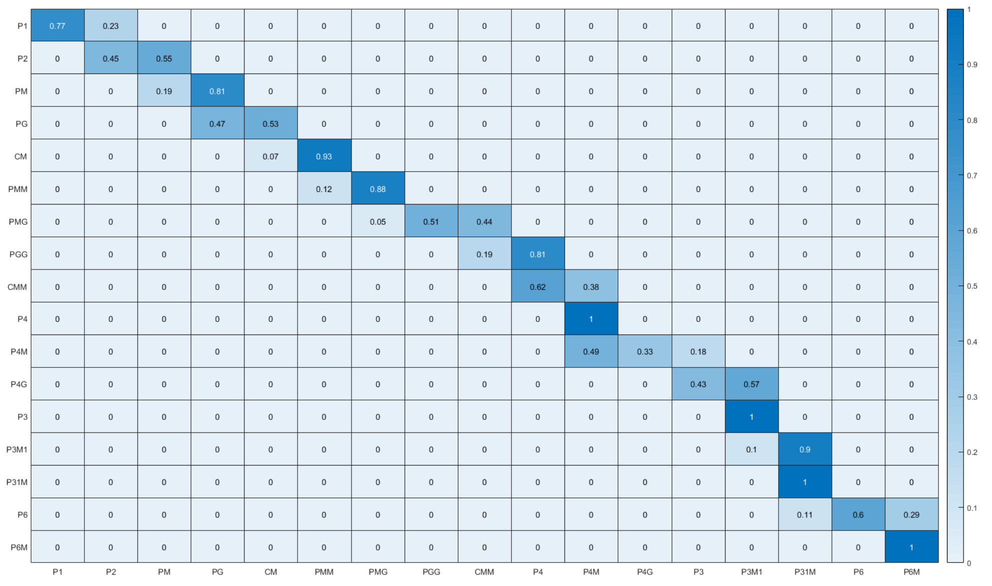
	P1	P2	PM	PG	CM	PMM	PMG	PGG	CMM	P4	P4M	P4G	P3	P3M1	P31M	P6	P6M
P1	0.855	0.145	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P2	0	0.37	0.63	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PM	0	0	0.202	0.798	0	0	0	0	0	0	0	0	0	0	0	0	0
PG	0	0	0	0.397	0.562	0.041	0	0	0	0	0	0	0	0	0	0	0
CM	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
PMM	0	0	0	0	0	0.053	0.947	0	0	0	0	0	0	0	0	0	0
PMG	0	0	0	0	0	0	0.04	0.593	0.367	0	0	0	0	0	0	0	0
PGG	0	0	0	0	0	0	0	0	0.182	0.818	0	0	0	0	0	0	0
CMM	0	0	0	0	0	0	0	0	0	0.664	0.336	0	0	0	0	0	0
P4	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
P4M	0	0	0	0	0	0	0	0	0	0	0.394	0.397	0.209	0	0	0	0
P4G	0	0	0	0	0	0	0	0	0	0	0	0	0.438	0.562	0	0	0
P3	0	0	0	0	0	0	0	0	0	0	0	0	0	0.982	0.018	0	0
P3M1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
P31M	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
P6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.081	0.674	0.245
P6M	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 13: Classification Matrix for testing ($\alpha_1 = 1 \times 10^{-5}$)



	P1	P2	PM	PG	CM	PMM	PMG	PGG	CMM	P4	P4M	P4G	P3	P3M1	P31M	P6	P6M
P1	855	145	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P2	0	370	630	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PM	0	0	202	798	0	0	0	0	0	0	0	0	0	0	0	0	0
PG	0	0	0	397	562	41	0	0	0	0	0	0	0	0	0	0	0
CM	0	0	0	0	0	1000	0	0	0	0	0	0	0	0	0	0	0
PMM	0	0	0	0	0	53	947	0	0	0	0	0	0	0	0	0	0
PMG	0	0	0	0	0	0	40	593	367	0	0	0	0	0	0	0	0
PGG	0	0	0	0	0	0	0	0	182	818	0	0	0	0	0	0	0
CMM	0	0	0	0	0	0	0	0	0	664	336	0	0	0	0	0	0
P4	0	0	0	0	0	0	0	0	0	0	1000	0	0	0	0	0	0
P4M	0	0	0	0	0	0	0	0	0	0	394	397	209	0	0	0	0
P4G	0	0	0	0	0	0	0	0	0	0	0	0	438	562	0	0	0
P3	0	0	0	0	0	0	0	0	0	0	0	0	0	982	18	0	0
P3M1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1000	0	0
P31M	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1000	0	0
P6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	81	674	245
P6M	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1000

Figure 14: Confusion Matrix for testing ($\alpha_1 = 1 \times 10^{-5}$)



	P1	P2	PM	PG	CM	PMM	PMG	PGG	CMM	P4	P4M	P4G	P3	P3M1	P31M	P6	P6M
P1	0.77	0.23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P2	0	0.45	0.55	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PM	0	0	0.19	0.81	0	0	0	0	0	0	0	0	0	0	0	0	0
PG	0	0	0	0.47	0.53	0	0	0	0	0	0	0	0	0	0	0	0
CM	0	0	0	0	0.07	0.93	0	0	0	0	0	0	0	0	0	0	0
PMM	0	0	0	0	0	0.12	0.88	0	0	0	0	0	0	0	0	0	0
PMG	0	0	0	0	0	0	0.05	0.51	0.44	0	0	0	0	0	0	0	0
PGG	0	0	0	0	0	0	0	0	0.19	0.81	0	0	0	0	0	0	0
CMM	0	0	0	0	0	0	0	0	0	0.62	0.38	0	0	0	0	0	0
P4	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
P4M	0	0	0	0	0	0	0	0	0	0	0.49	0.33	0.18	0	0	0	0
P4G	0	0	0	0	0	0	0	0	0	0	0	0	0.43	0.57	0	0	0
P3	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
P3M1	0	0	0	0	0	0	0	0	0	0	0	0	0	0.1	0.9	0	0
P31M	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
P6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.11	0.6	0.29
P6M	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 15: Classification Matrix for validation ($\alpha_1 = 1 \times 10^{-5}$)

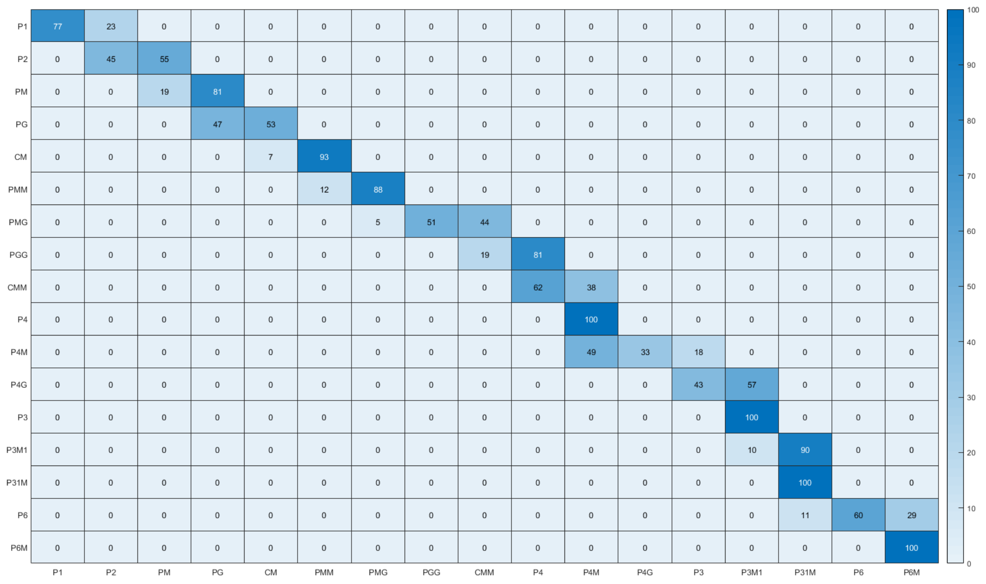


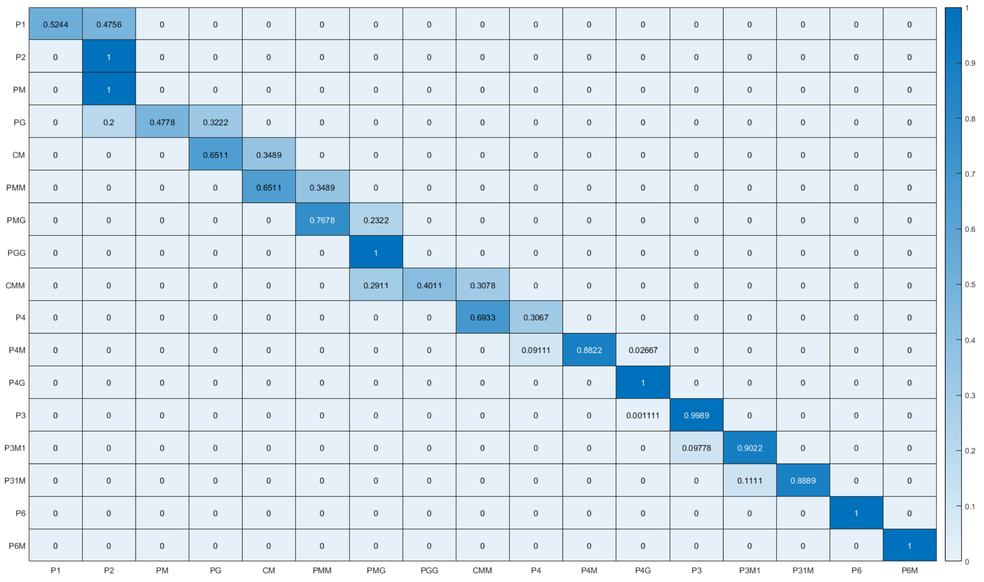
Figure 16: Confusion Matrix for validation ($\alpha_1 = 1 \times 10^{-5}$)

The classification and confusion matrices for $\alpha_2 = 5 \times 10^{-4}$ are as follows for training in fig 11 and fig 12, testing in fig 13 and fig 14 and validation in fig 15 and fig 16 respectively.

Therefore we notice that when we decrease the learning rate, the accuracy decreases considerably. The accuracies of training, testing and validation for α_1 and α_2 are as shown in Table 1

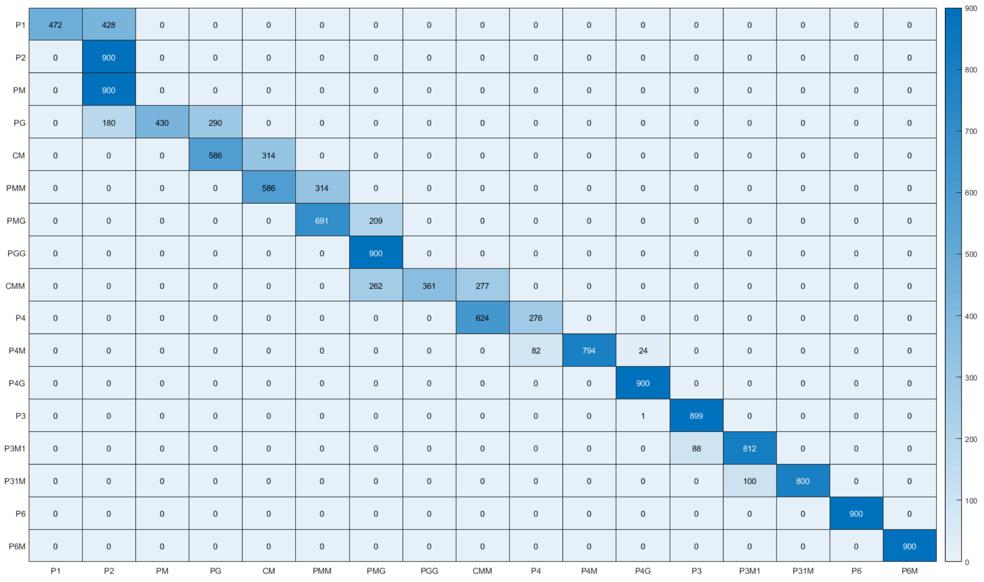
	α_1	α_2
Training	13.61	71.95
Testing	12.44	67.8
Validation	12.6	67.82

Table 1: Accuracy for α_1 and α_2



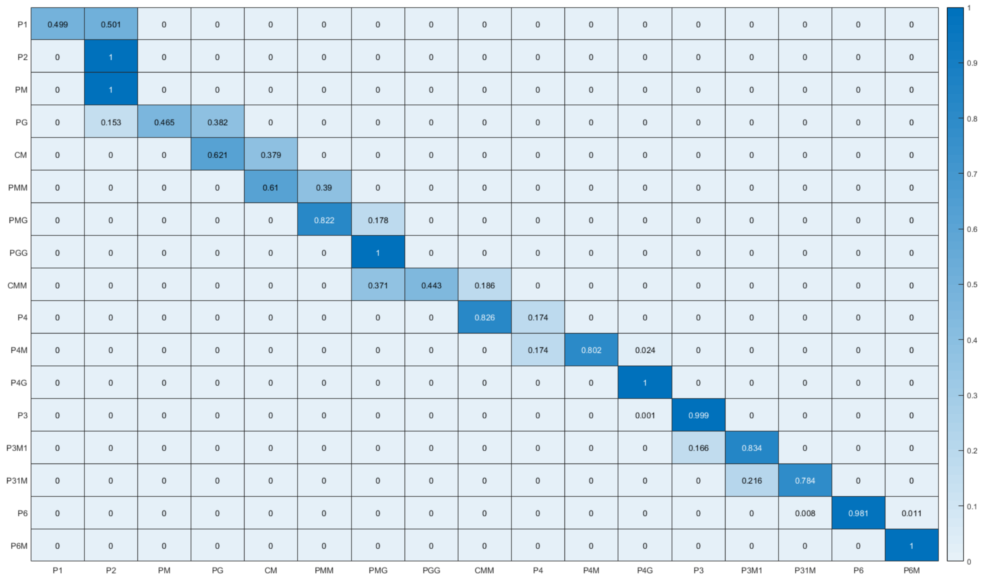
	P1	P2	PM	PG	CM	PMM	PMG	PGG	CMM	P4	P4M	P4G	P3	P3M1	P31M	P6	P6M
P1	0.5244	0.4756	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PM	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PG	0	0.2	0.4778	0.3222	0	0	0	0	0	0	0	0	0	0	0	0	0
CM	0	0	0	0.6511	0.3489	0	0	0	0	0	0	0	0	0	0	0	0
PMM	0	0	0	0	0.6511	0.3489	0	0	0	0	0	0	0	0	0	0	0
PMG	0	0	0	0	0	0.7678	0.2322	0	0	0	0	0	0	0	0	0	0
PGG	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
CMM	0	0	0	0	0	0	0.2911	0.4011	0.3078	0	0	0	0	0	0	0	0
P4	0	0	0	0	0	0	0	0	0.6933	0.3067	0	0	0	0	0	0	0
P4M	0	0	0	0	0	0	0	0	0	0.09111	0.8822	0.02667	0	0	0	0	0
P4G	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
P3	0	0	0	0	0	0	0	0	0	0	0	0.001111	0.9989	0	0	0	0
P3M1	0	0	0	0	0	0	0	0	0	0	0	0	0.09778	0.9022	0	0	0
P31M	0	0	0	0	0	0	0	0	0	0	0	0	0	0.1111	0.8889	0	0
P6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
P6M	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 17: Classification Matrix for Training ($\alpha_2 = 5 \times 10^{-4}$)



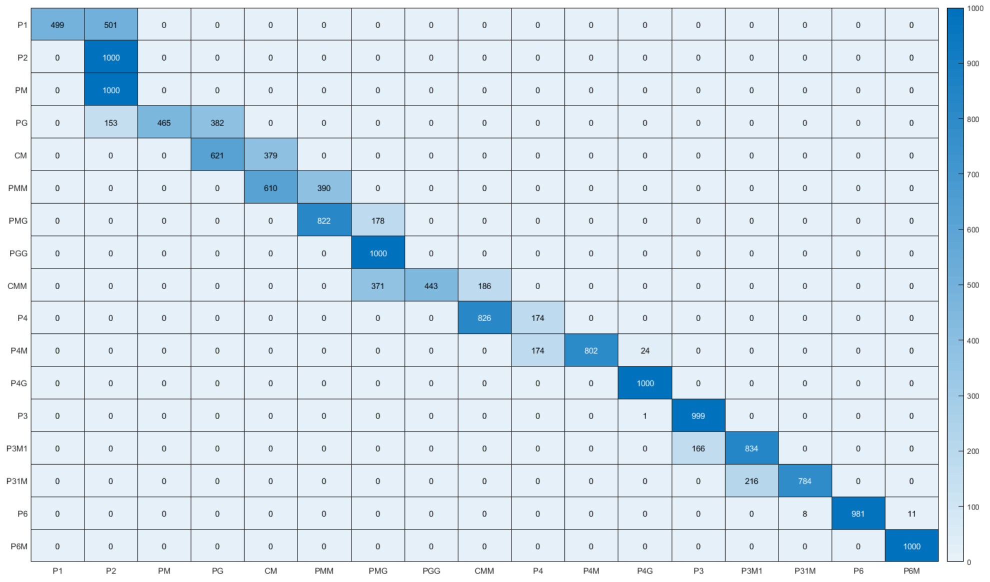
	P1	P2	PM	PG	CM	PMM	PMG	PGG	CMM	P4	P4M	P4G	P3	P3M1	P31M	P6	P6M
P1	472	428	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P2	0	900	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PM	0	900	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PG	0	180	430	290	0	0	0	0	0	0	0	0	0	0	0	0	0
CM	0	0	0	586	314	0	0	0	0	0	0	0	0	0	0	0	0
PMM	0	0	0	0	586	314	0	0	0	0	0	0	0	0	0	0	0
PMG	0	0	0	0	0	691	209	0	0	0	0	0	0	0	0	0	0
PGG	0	0	0	0	0	0	900	0	0	0	0	0	0	0	0	0	0
CMM	0	0	0	0	0	0	262	361	277	0	0	0	0	0	0	0	0
P4	0	0	0	0	0	0	0	0	624	276	0	0	0	0	0	0	0
P4M	0	0	0	0	0	0	0	0	0	82	794	24	0	0	0	0	0
P4G	0	0	0	0	0	0	0	0	0	0	0	900	0	0	0	0	0
P3	0	0	0	0	0	0	0	0	0	0	0	1	899	0	0	0	0
P3M1	0	0	0	0	0	0	0	0	0	0	0	0	88	812	0	0	0
P31M	0	0	0	0	0	0	0	0	0	0	0	0	0	100	800	0	0
P6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	900	0
P6M	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	900

Figure 18: Confusion Matrix for Training ($\alpha_2 = 5 \times 10^{-4}$)



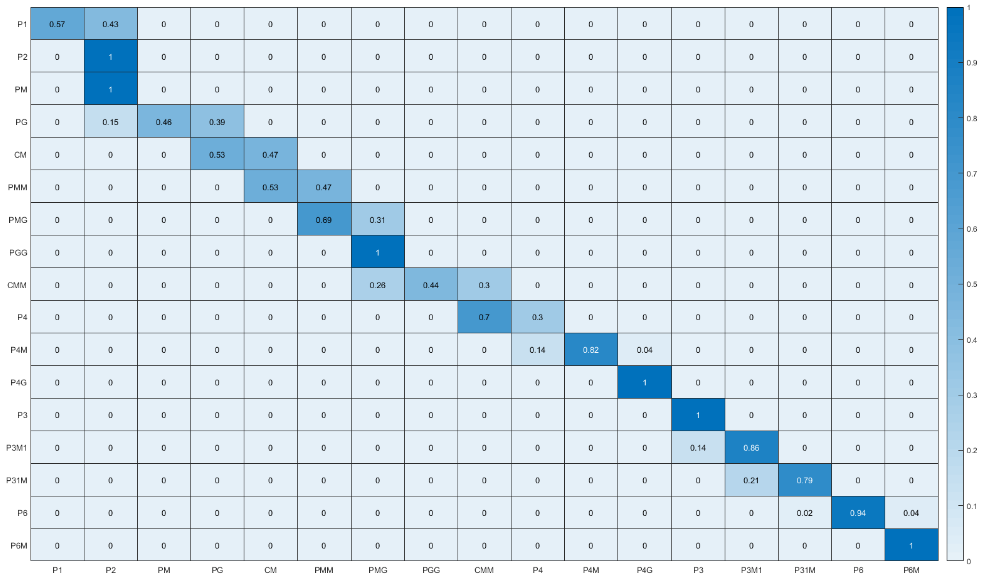
	P1	P2	PM	PG	CM	PMM	PMG	PGG	CMM	P4	P4M	P4G	P3	P3M1	P31M	P6	P6M
P1	0.499	0.501	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PM	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PG	0	0.153	0.465	0.382	0	0	0	0	0	0	0	0	0	0	0	0	0
CM	0	0	0	0.621	0.379	0	0	0	0	0	0	0	0	0	0	0	0
PMM	0	0	0	0	0.61	0.39	0	0	0	0	0	0	0	0	0	0	0
PMG	0	0	0	0	0	0.822	0.178	0	0	0	0	0	0	0	0	0	0
PGG	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
CMM	0	0	0	0	0	0	0.371	0.443	0.186	0	0	0	0	0	0	0	0
P4	0	0	0	0	0	0	0	0	0.826	0.174	0	0	0	0	0	0	0
P4M	0	0	0	0	0	0	0	0	0	0.174	0.802	0.024	0	0	0	0	0
P4G	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
P3	0	0	0	0	0	0	0	0	0	0	0	0.001	0.999	0	0	0	0
P3M1	0	0	0	0	0	0	0	0	0	0	0	0	0.166	0.834	0	0	0
P31M	0	0	0	0	0	0	0	0	0	0	0	0	0	0.216	0.784	0	0
P6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.008	0.981	0.011
P6M	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 19: Classification Matrix for testing ($\alpha_2 = 5 \times 10^{-4}$)



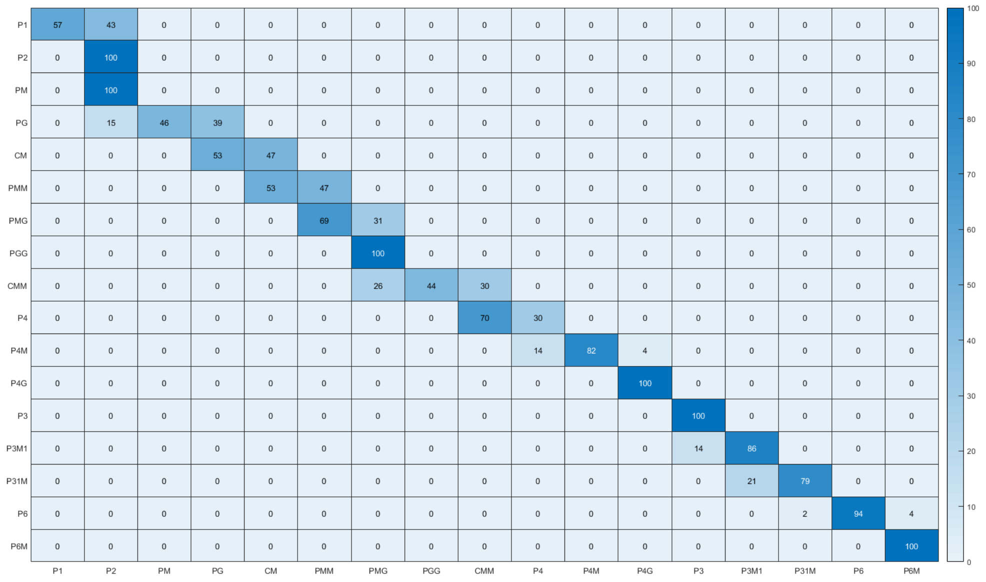
	P1	P2	PM	PG	CM	PMM	PMG	PGG	CMM	P4	P4M	P4G	P3	P3M1	P31M	P6	P6M
P1	499	501	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P2	0	1000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PM	0	1000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PG	0	153	465	382	0	0	0	0	0	0	0	0	0	0	0	0	0
CM	0	0	0	621	379	0	0	0	0	0	0	0	0	0	0	0	0
PMM	0	0	0	0	610	390	0	0	0	0	0	0	0	0	0	0	0
PMG	0	0	0	0	0	822	178	0	0	0	0	0	0	0	0	0	0
PGG	0	0	0	0	0	0	1000	0	0	0	0	0	0	0	0	0	0
CMM	0	0	0	0	0	0	371	443	186	0	0	0	0	0	0	0	0
P4	0	0	0	0	0	0	0	0	826	174	0	0	0	0	0	0	0
P4M	0	0	0	0	0	0	0	0	0	174	802	24	0	0	0	0	0
P4G	0	0	0	0	0	0	0	0	0	0	0	1000	0	0	0	0	0
P3	0	0	0	0	0	0	0	0	0	0	0	1	999	0	0	0	0
P3M1	0	0	0	0	0	0	0	0	0	0	0	0	166	834	0	0	0
P31M	0	0	0	0	0	0	0	0	0	0	0	0	0	216	784	0	0
P6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8	981	11
P6M	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1000

Figure 20: Confusion Matrix for testing ($\alpha_2 = 5 \times 10^{-4}$)



	P1	P2	PM	PG	CM	PMM	PMG	PGG	CMM	P4	P4M	P4G	P3	P3M1	P31M	P6	P6M
P1	0.57	0.43	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PM	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PG	0	0.15	0.46	0.39	0	0	0	0	0	0	0	0	0	0	0	0	0
CM	0	0	0	0.53	0.47	0	0	0	0	0	0	0	0	0	0	0	0
PMM	0	0	0	0	0.53	0.47	0	0	0	0	0	0	0	0	0	0	0
PMG	0	0	0	0	0	0.69	0.31	0	0	0	0	0	0	0	0	0	0
PGG	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
CMM	0	0	0	0	0	0	0.26	0.44	0.3	0	0	0	0	0	0	0	0
P4	0	0	0	0	0	0	0	0	0.7	0.3	0	0	0	0	0	0	0
P4M	0	0	0	0	0	0	0	0	0	0.14	0.82	0.04	0	0	0	0	0
P4G	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
P3	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
P3M1	0	0	0	0	0	0	0	0	0	0	0	0	0.14	0.86	0	0	0
P31M	0	0	0	0	0	0	0	0	0	0	0	0	0	0.21	0.79	0	0
P6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.02	0.94	0.04
P6M	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 21: Classification Matrix for validation ($\alpha_2 = 5 \times 10^{-4}$)

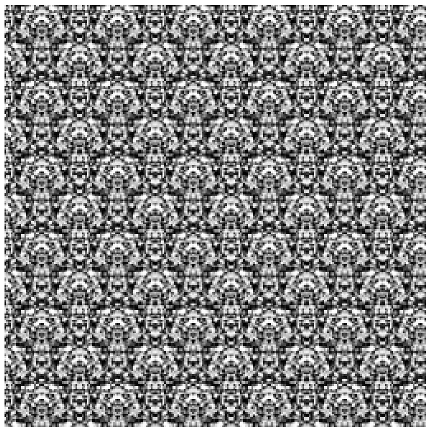


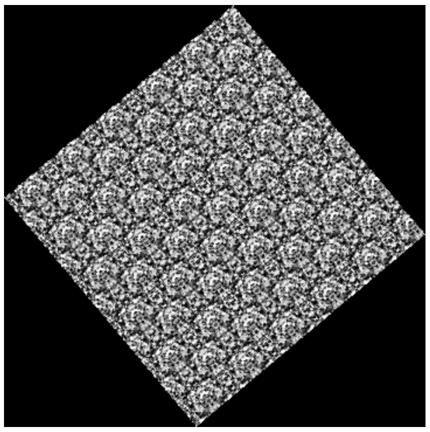
	P1	P2	PM	PG	CM	PMM	PMG	PGG	CMM	P4	P4M	P4G	P3	P3M1	P31M	P6	P6M
P1	57	43	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P2	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PM	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PG	0	15	46	39	0	0	0	0	0	0	0	0	0	0	0	0	0
CM	0	0	0	53	47	0	0	0	0	0	0	0	0	0	0	0	0
PMM	0	0	0	0	53	47	0	0	0	0	0	0	0	0	0	0	0
PMG	0	0	0	0	0	69	31	0	0	0	0	0	0	0	0	0	0
PGG	0	0	0	0	0	0	100	0	0	0	0	0	0	0	0	0	0
CMM	0	0	0	0	0	0	26	44	30	0	0	0	0	0	0	0	0
P4	0	0	0	0	0	0	0	0	70	30	0	0	0	0	0	0	0
P4M	0	0	0	0	0	0	0	0	0	14	82	4	0	0	0	0	0
P4G	0	0	0	0	0	0	0	0	0	0	0	100	0	0	0	0	0
P3	0	0	0	0	0	0	0	0	0	0	0	0	100	0	0	0	0
P3M1	0	0	0	0	0	0	0	0	0	0	0	0	14	86	0	0	0
P31M	0	0	0	0	0	0	0	0	0	0	0	0	0	21	79	0	0
P6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	94	4
P6M	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100

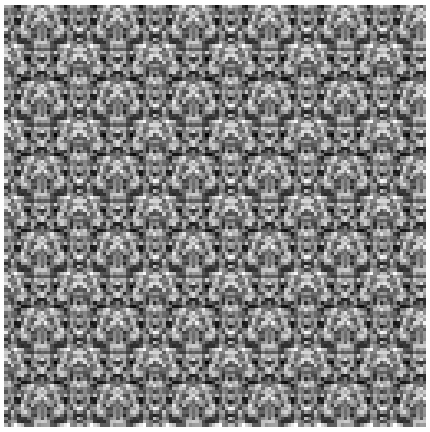
Figure 22: Confusion Matrix for validation ($\alpha_2 = 5 \times 10^{-4}$)

3.2 Augmentation of Dataset

To increase the dataset, we make use of the rotation, scaling and translation functions of matlab to obtain the augmented dataset. One such example for the input image ?? is as follows:


(a) CM_1 input image


(b) CM_1 rotated by 40 degrees


(c) CM_1 scaled to 50%

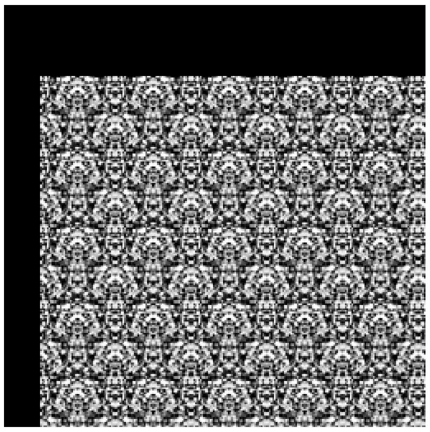
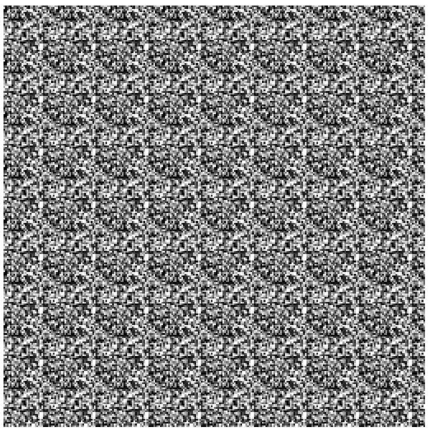
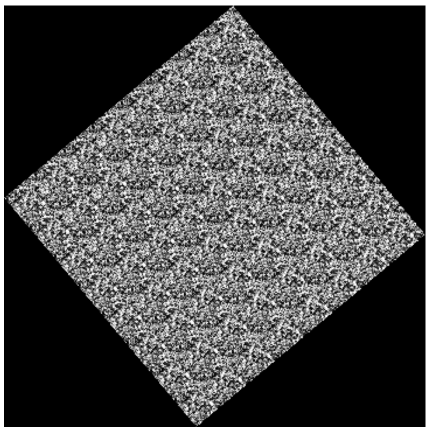
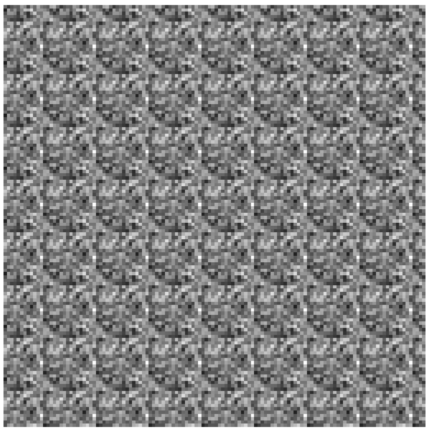

(d) CM_1 translated by [22,43]

Figure 23: CM_1.png augmentation

The following figures showcases the same for three such images.


(a) P1_1 input image


(b) P1_1 rotated by 40 degrees


(c) P1_1 scaled to 50%

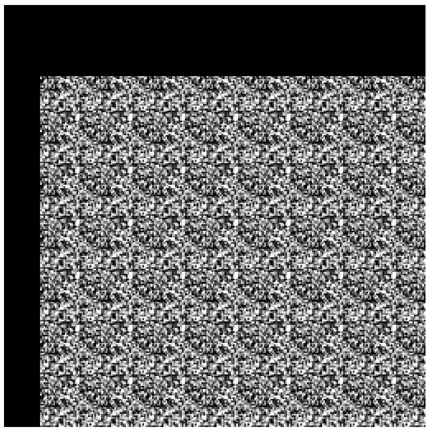
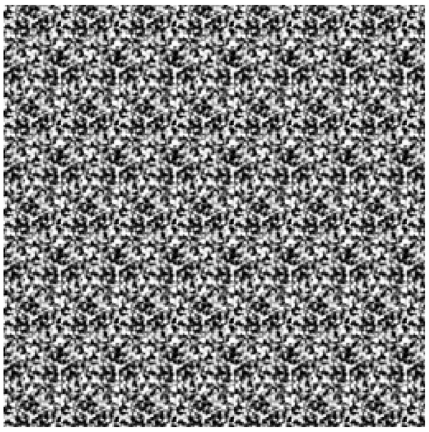
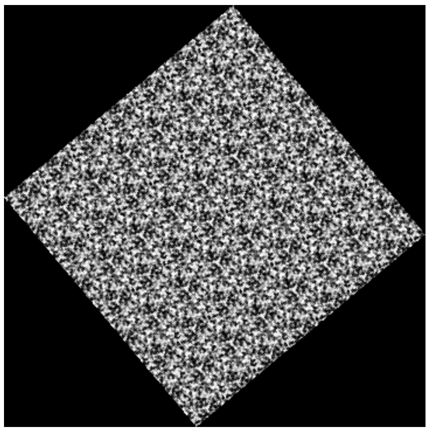
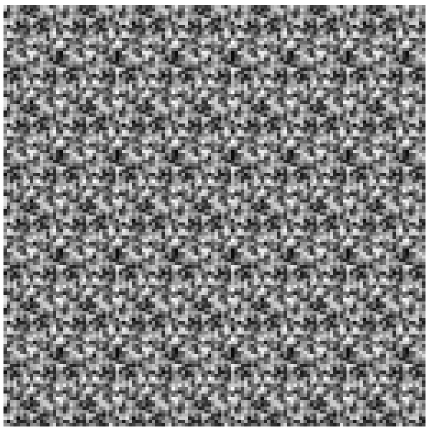

(d) P1_1 translated by [22,43]

Figure 24: P1_1.png augmentation


(a) P3_1 input image


(b) P3_1 rotated by 40 degrees


(c) P3_1 scaled to 50%

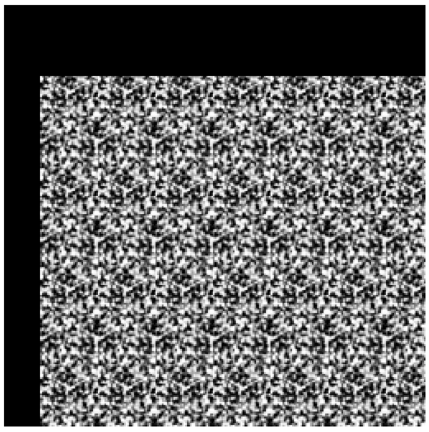
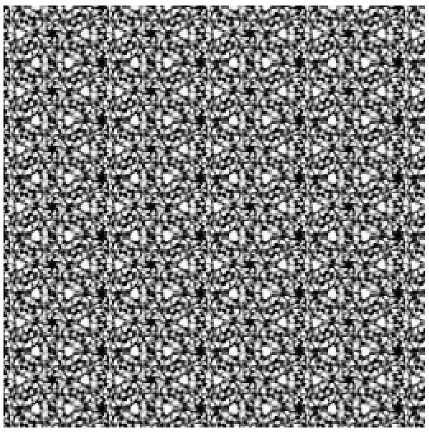
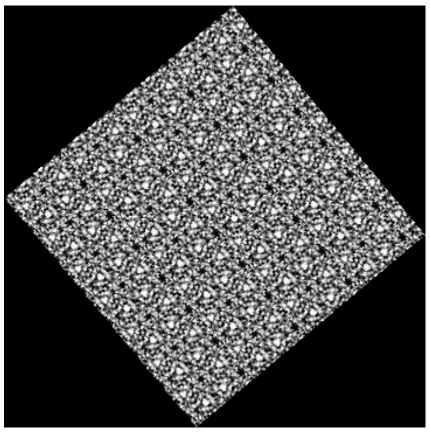
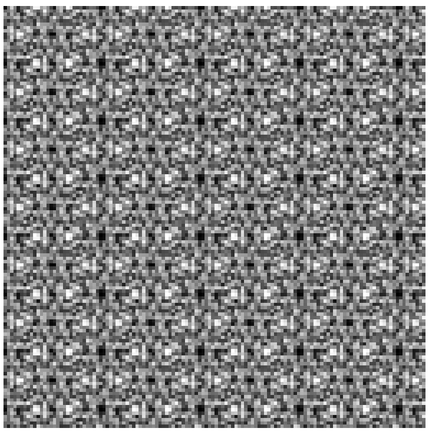

(d) P3_1 translated by [22,43]

Figure 25: P3_1.png augmentation


(a) P6_1 input image


(b) P6_1 rotated by 40 degrees


(c) P6_1 scaled to 50%

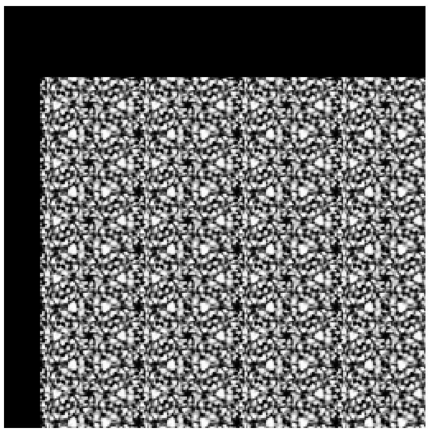
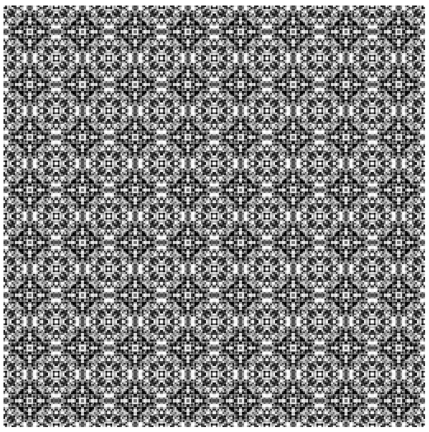
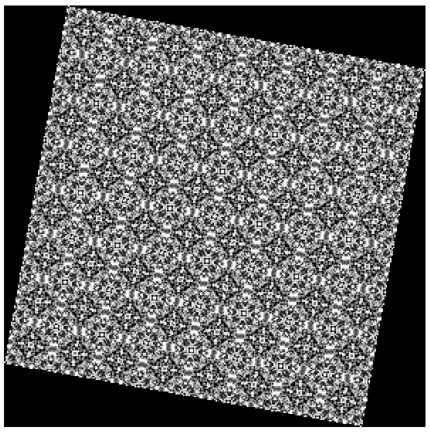
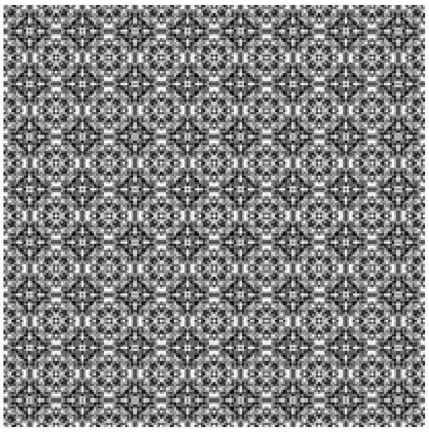

(d) P6_1 translated by [22,43]

Figure 26: P6_1.png augmentation


(a) P4M_1 input image


(b) P4M_1 rotated by 260 degrees


(c) P4M_1 scaled to 80%

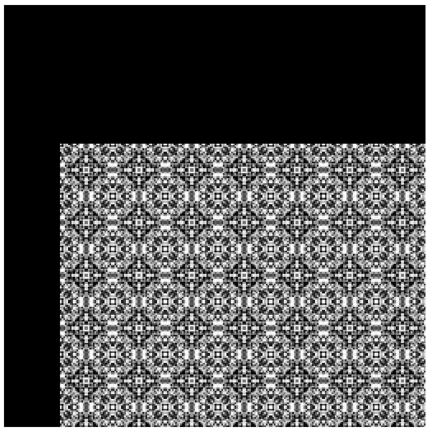

(d) P4M_1 translated by [34,84]

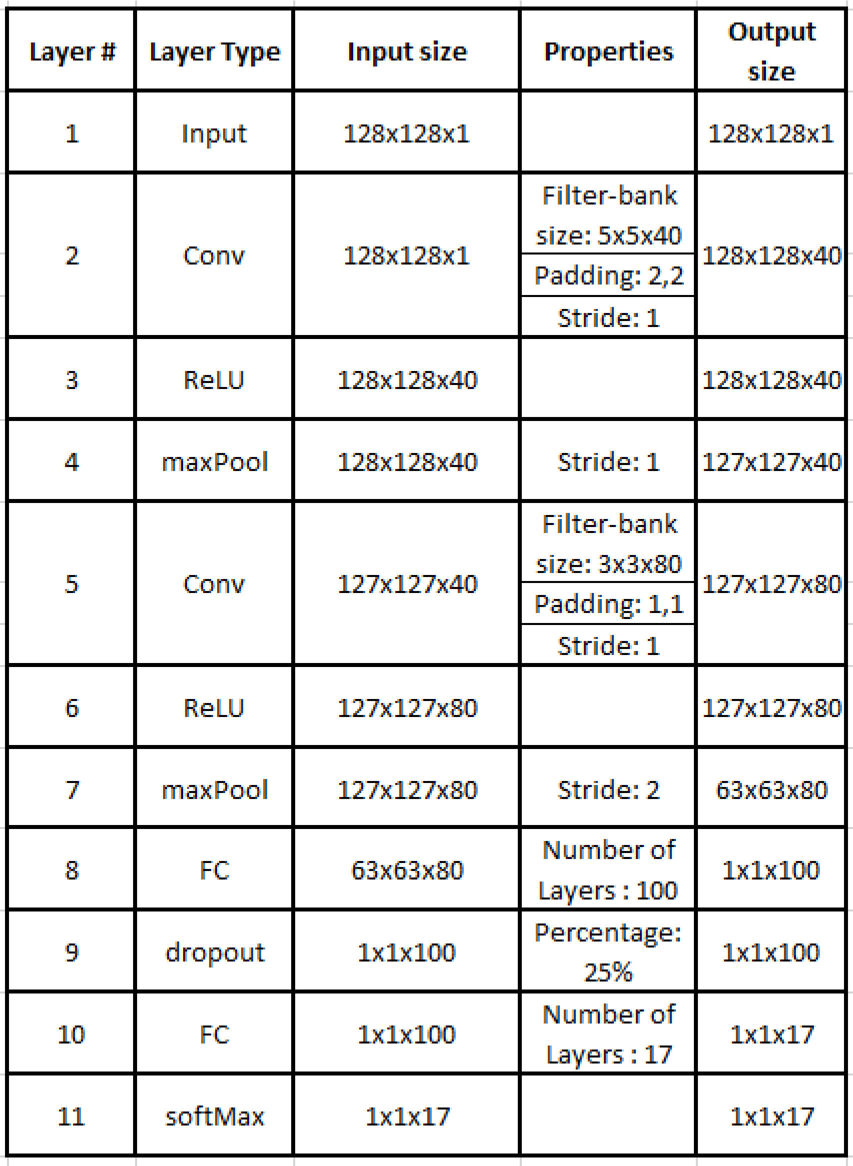
Figure 27: P4M_1.png augmentation

All these images shown above are the augmented datasets and once they are combined together one after the other in a random fashion in order to generate more number of data. By doing so, we are creating a larger dataset thereby being able to train the network in a better manner.

3.3 Wide network

3.3.1 Wide network on Augmented Dataset

The wide network was run for two sets of values. One of them is as explained in Fig 6 while the other consisted of the following configuration as shown in Fig 28. While the two configurations did not have much of an effect on the accuracy, further configurations could not be tried due to time and GPU constraints. The network that I have considered for plotting graphs and matrices is the one represented in 6 and the confusion matrix, classification matrix for training, testing and validation are with respect to this configuration. The classification and confusion matrices are as shown in Fig 29a and 29b for training, 30a and 30b for testing and 31a and 31b for validation respectively.



Layer #	Layer Type	Input size	Properties	Output size
1	Input	128x128x1		128x128x1
2	Conv	128x128x1	Filter-bank size: 5x5x40 Padding: 2,2 Stride: 1	128x128x40
3	ReLU	128x128x40		128x128x40
4	maxPool	128x128x40	Stride: 1	127x127x40
5	Conv	127x127x40	Filter-bank size: 3x3x80 Padding: 1,1 Stride: 1	127x127x80
6	ReLU	127x127x80		127x127x80
7	maxPool	127x127x80	Stride: 2	63x63x80
8	FC	63x63x80	Number of Layers : 100	1x1x100
9	dropout	1x1x100	Percentage: 25%	1x1x100
10	FC	1x1x100	Number of Layers : 17	1x1x17
11	softMax	1x1x17		1x1x17

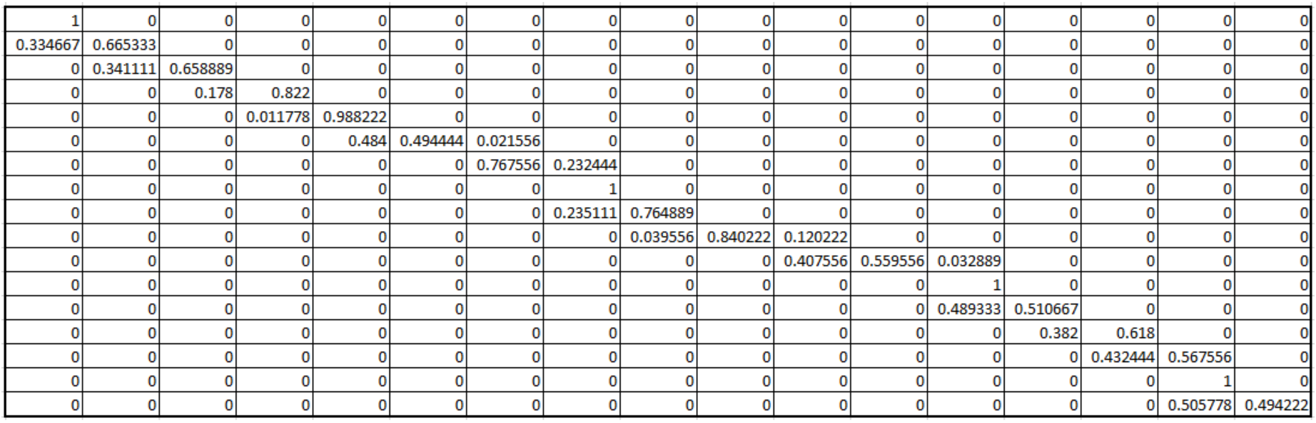
Figure 28: Wide network second Configuration

Table 2 shows the accuracy obtained on training, testing and validation of both configurations.

	Configuration 6	Configuration 28
Training	30.99	28.21
Testing	13.55	10.22
Validation	13.61	10.65

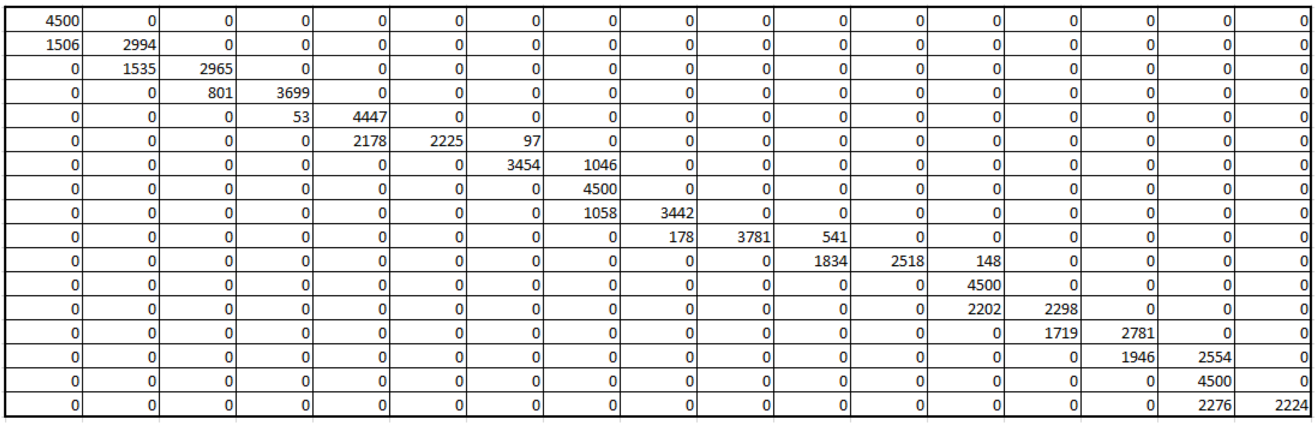
Table 2: Accuracy for wide Network with $\alpha = 1 \times 10^{-3}$ and number of Epochs = 10

The heat maps in this case weren't generated since they do not display properly on the report as well as on MATLAB their size needs to be increased.



1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0.334667	0.665333	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0.341111	0.658889	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0.178	0.822	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0.011778	0.988222	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0.484	0.494444	0.021556	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0.767556	0.232444	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0.235111	0.764889	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0.039556	0.840222	0.120222	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0.407556	0.559556	0.032889	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0.489333	0.510667	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0.382	0.618	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.432444	0.567556	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.505778	0.494222

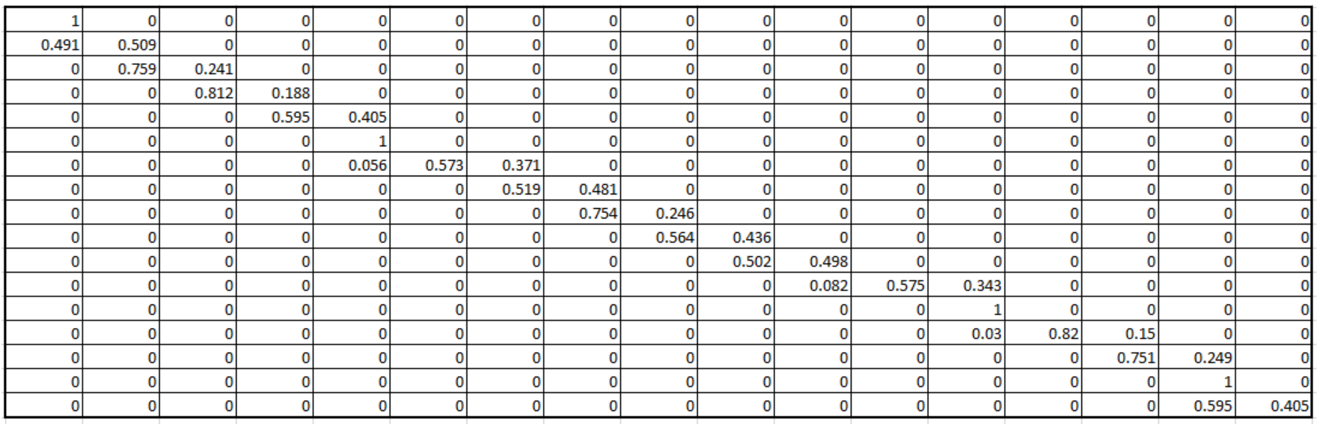
(a) Classification Matrix



4500	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1506	2994	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1535	2965	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	801	3699	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	53	4447	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	2178	2225	97	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	3454	1046	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	4500	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1058	3442	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	178	3781	541	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1834	2518	148	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	4500	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	2202	2298	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1719	2781	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1946	2554	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4500	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2276	2224

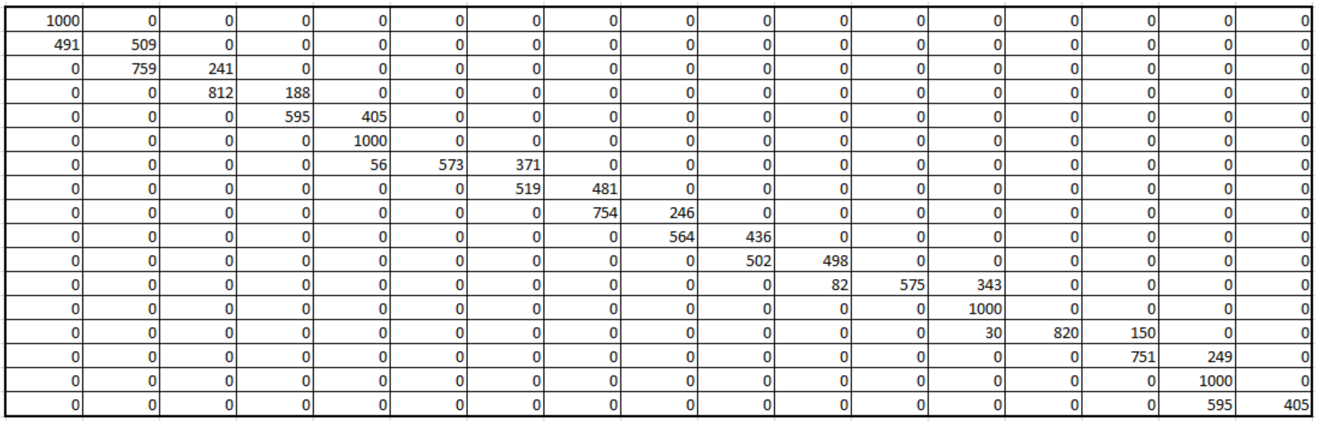
(b) Confusion Matrix

Figure 29: Wide network: Training on Augmented Dataset



1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0.491	0.509	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0.759	0.241	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0.812	0.188	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0.595	0.405	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0.056	0.573	0.371	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0.519	0.481	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0.754	0.246	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0.564	0.436	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0.502	0.498	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0.082	0.575	0.343	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0.03	0.82	0.15	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.751	0.249	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.595	0.405

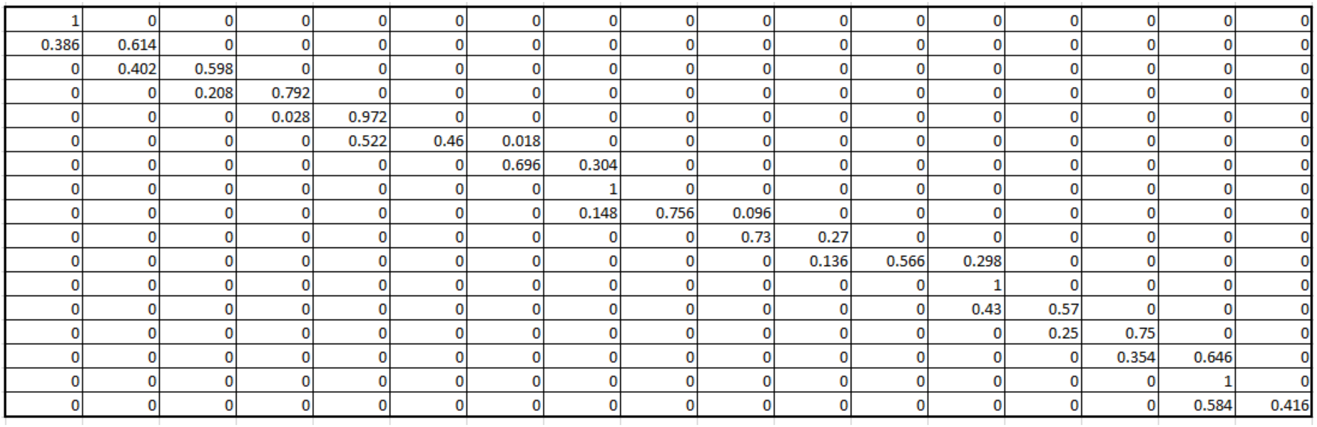
(a) Classification Matrix



1000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
491	509	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	759	241	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	812	188	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	595	405	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1000	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	56	573	371	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	519	481	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	754	246	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	564	436	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	502	498	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	82	575	343	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1000	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	30	820	150	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	751	249	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1000	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	595	405

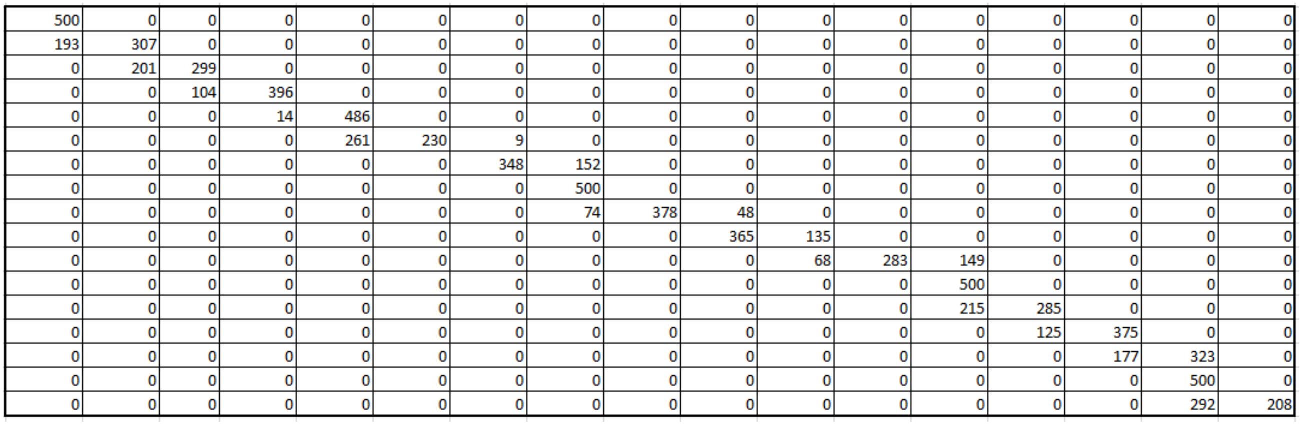
(b) Confusion Matrix

Figure 30: Wide network: Testing on Augmented Dataset



1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0.386	0.614	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0.402	0.598	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0.208	0.792	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0.028	0.972	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0.522	0.46	0.018	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0.696	0.304	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0.148	0.756	0.096	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0.73	0.27	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0.136	0.566	0.298	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0.43	0.57	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0.25	0.75	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.354	0.646	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.584	0.416

(a) Classification Matrix



500	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
193	307	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	201	299	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	104	396	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	14	486	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	261	230	9	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	348	152	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	500	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	74	378	48	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	365	135	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	68	283	149	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	500	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	215	285	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	125	375	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	177	323	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	500	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	292	208

(b) Confusion Matrix

Figure 31: Wide network: Validation on Augmented Dataset

The error plot is as shown in Fig 32

The command window recording of mini batch accuracy and the time taken for the network to completely train and run is given by fig 33. Please note that only the final record of each epoch has been shown in the report.

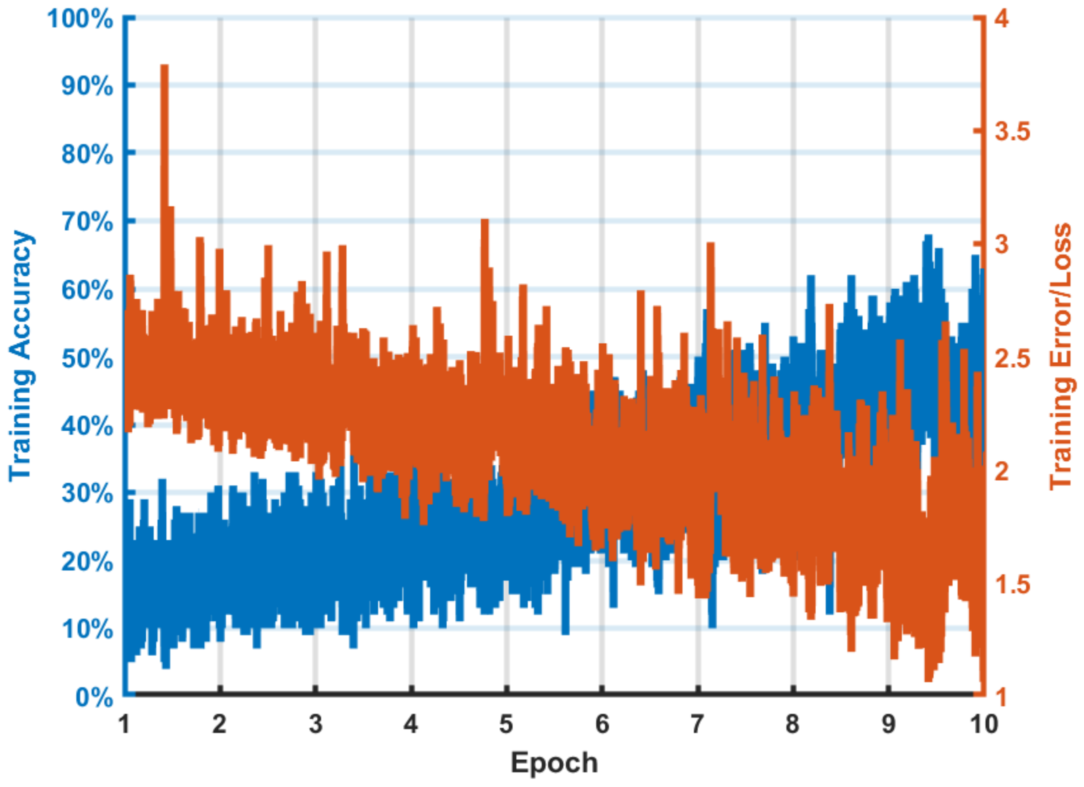
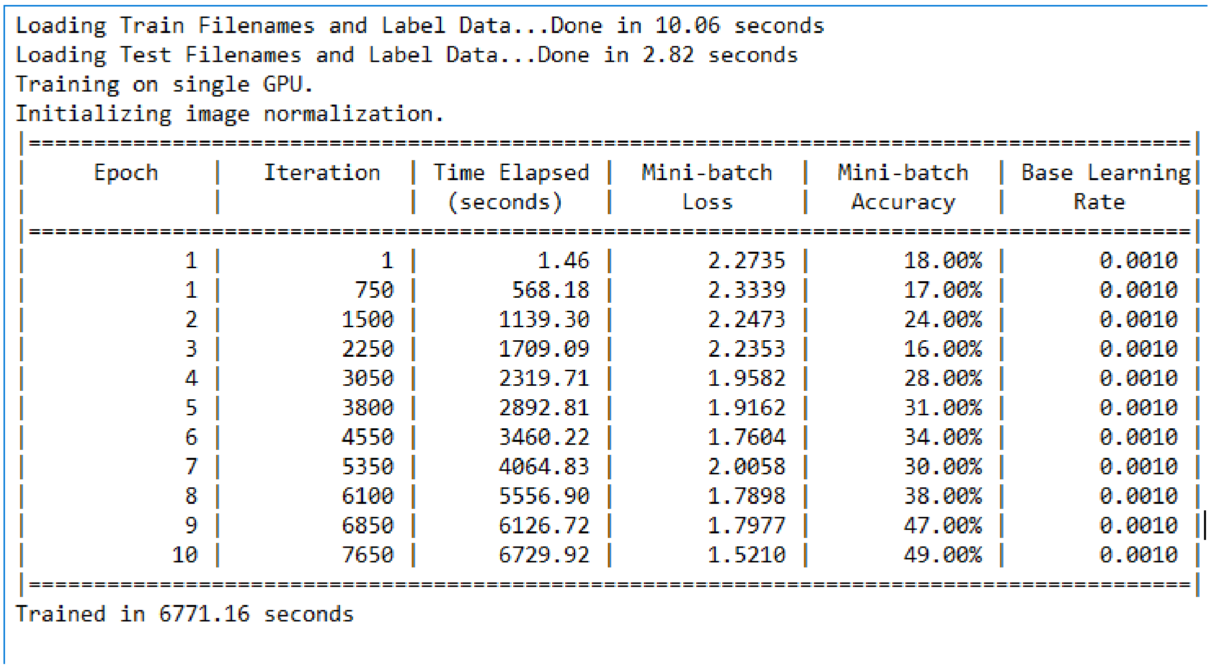


Figure 32: Error plot of Wide network on Augmented Dataset



```
Loading Train Filenames and Label Data...Done in 10.06 seconds
Loading Test Filenames and Label Data...Done in 2.82 seconds
Training on single GPU.
Initializing image normalization.
|=======================================================================================|
|     Epoch    |   Iteration  | Time Elapsed |  Mini-batch  |  Mini-batch  | Base Learning|
|              |              |  (seconds)   |     Loss     |   Accuracy   |     Rate     |
|=======================================================================================|
|            1 |            1 |         1.46 |       2.2735 |       18.00% |       0.0010 |
|            1 |          750 |       568.18 |       2.3339 |       17.00% |       0.0010 |
|            2 |         1500 |      1139.30 |       2.2473 |       24.00% |       0.0010 |
|            3 |         2250 |      1709.09 |       2.2353 |       16.00% |       0.0010 |
|            4 |         3050 |      2319.71 |       1.9582 |       28.00% |       0.0010 |
|            5 |         3800 |      2892.81 |       1.9162 |       31.00% |       0.0010 |
|            6 |         4550 |      3460.22 |       1.7604 |       34.00% |       0.0010 |
|            7 |         5350 |      4064.83 |       2.0058 |       30.00% |       0.0010 |
|            8 |         6100 |      5556.90 |       1.7898 |       38.00% |       0.0010 |
|            9 |         6850 |      6126.72 |       1.7977 |       47.00% |       0.0010 |
|           10 |         7650 |      6729.92 |       1.5210 |       49.00% |       0.0010 |
|=======================================================================================|
Trained in 6771.16 seconds
```

Figure 33: Mini batch accuracy and time results of Wide network

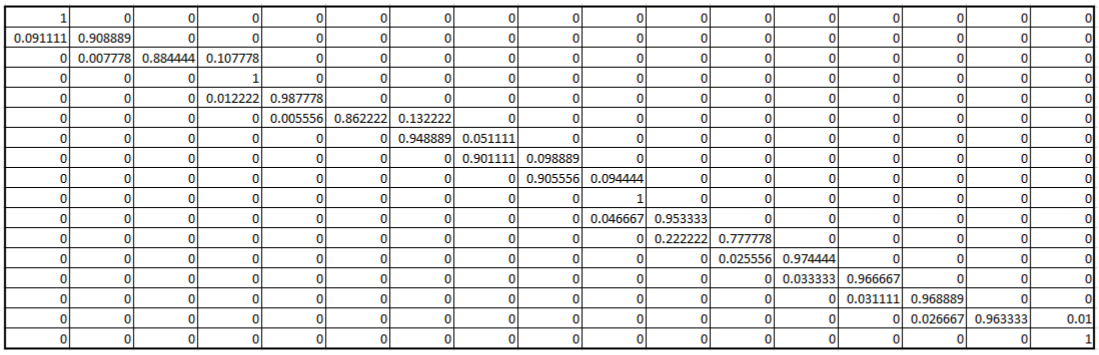
3.3.2 Wide Network on Original Dataset

After having trained the wide network, the layers were loaded onto the workspace by using the last checkpoint and thereby re training the network. After having done this, we obtain the training,testing and validation accuracies and they are observed to be much higher than when compared to the augmented dataset. This can be observed from Table 2 and 3. This is due to the fact that having trained the network with large amount of data, the network would have learnt the basic images from the original dataset. That is the reason we achieve high accuracy.

	On Augmented dataset	On Original (17,000) dataset
Training	30.99	86.93
Testing	13.55	84.20
Validation	13.61	84.65

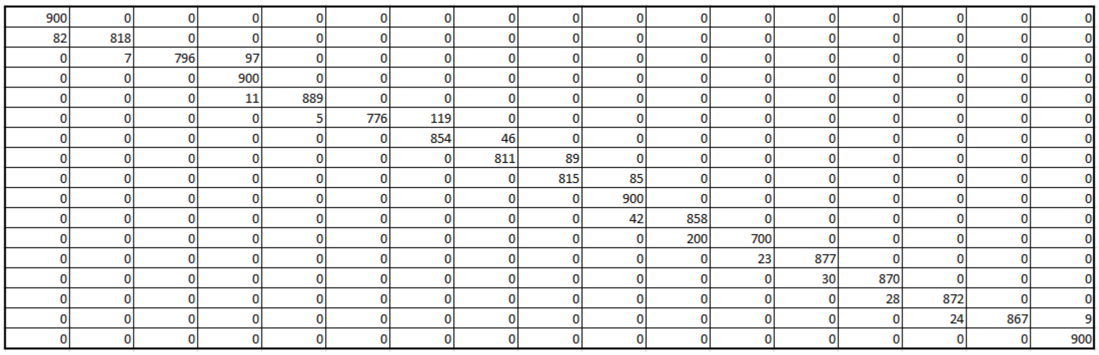
Table 3: Accuracy for wide Network with $\alpha = 1 \times 10^{-3}$ and number of Epochs = 5

The classification and confusion matrices for training, testing and validation are as shown in 34, 35 and 36 respectively.



1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0.091111	0.908889	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0.007778	0.884444	0.107778	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0.012222	0.987778	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0.005556	0.862222	0.132222	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0.948889	0.051111	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0.901111	0.098889	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0.905556	0.094444	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0.046667	0.953333	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0.222222	0.777778	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0.025556	0.974444	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0.033333	0.966667	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0.031111	0.968889	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.026667	0.963333	0.01
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

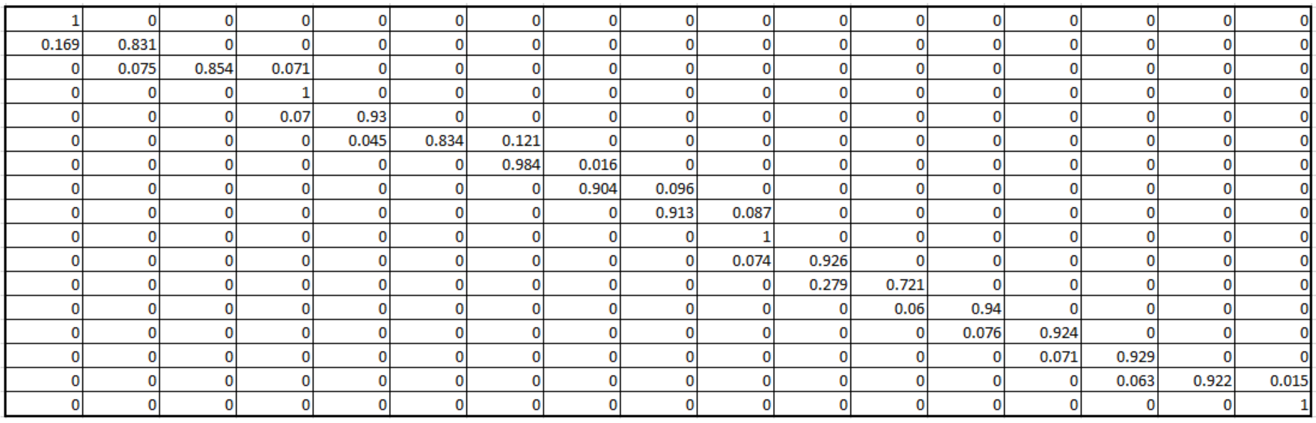
(a) Classification Matrix



900	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
82	818	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	7	796	97	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	900	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	11	889	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	5	776	119	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	854	46	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	811	89	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	815	85	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	900	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	42	858	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	200	700	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	23	877	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	30	870	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	28	872	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	24	867	9
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	900

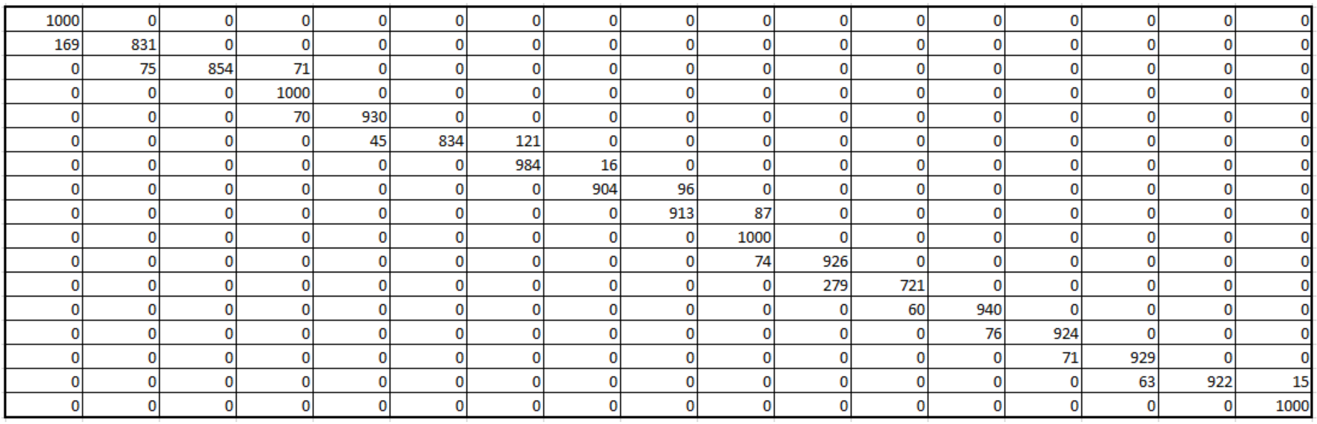
(b) Confusion Matrix

Figure 34: Wide network: Training on Original Dataset



1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0.169	0.831	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0.075	0.854	0.071	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0.07	0.93	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0.045	0.834	0.121	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0.984	0.016	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0.904	0.096	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0.913	0.087	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0.074	0.926	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0.279	0.721	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0.06	0.94	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0.076	0.924	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0.071	0.929	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.063	0.922	0.015
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

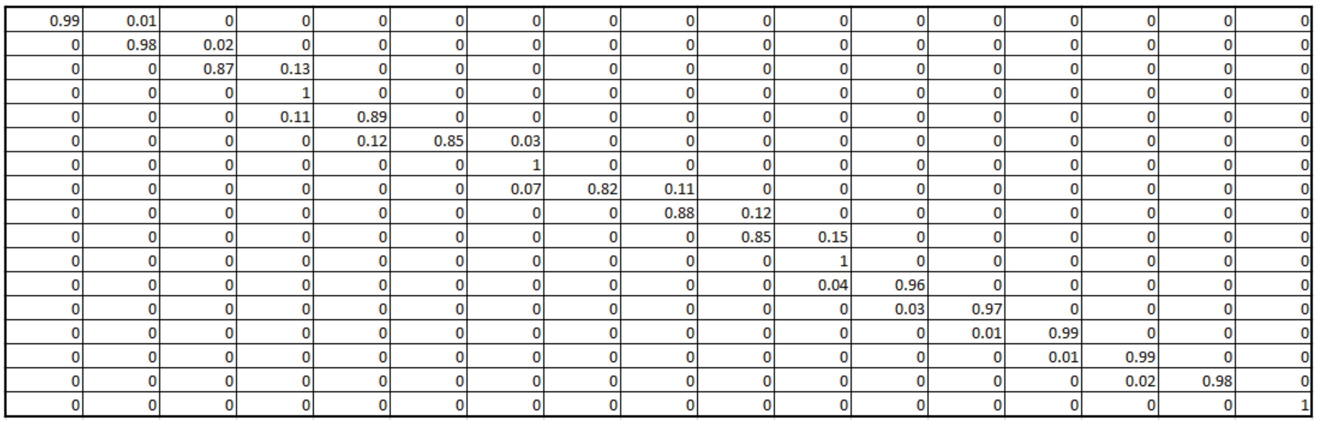
(a) Classification Matrix



1000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
169	831	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	75	854	71	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1000	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	70	930	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	45	834	121	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	984	16	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	904	96	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	913	87	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1000	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	74	926	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	279	721	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	60	940	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	76	924	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	71	929	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	63	922	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1000

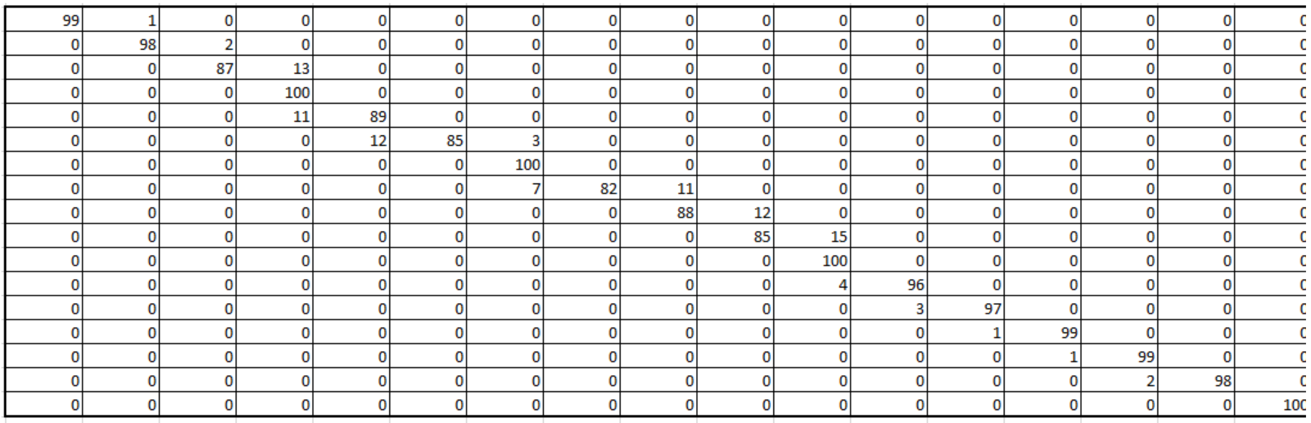
(b) Confusion Matrix

Figure 35: Wide network: Testing on Original Dataset



0.99	0.01	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0.98	0.02	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0.87	0.13	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0.11	0.89	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0.12	0.85	0.03	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0.07	0.82	0.11	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0.88	0.12	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0.85	0.15	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0.04	0.96	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0.03	0.97	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0.01	0.99	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0.01	0.99	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.02	0.98	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

(a) Classification Matrix



99	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	98	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	87	13	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	11	89	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	12	85	3	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	100	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	7	82	11	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	88	12	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	85	15	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	100	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	4	96	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	3	97	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	99	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	99	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	98	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100

(b) Confusion Matrix

Figure 36: Wide network: Validation on Original Dataset

The error plot is as shown in Fig 37

The command window recording of mini batch accuracy and the time taken for the network to completely train and run is given by fig 38.

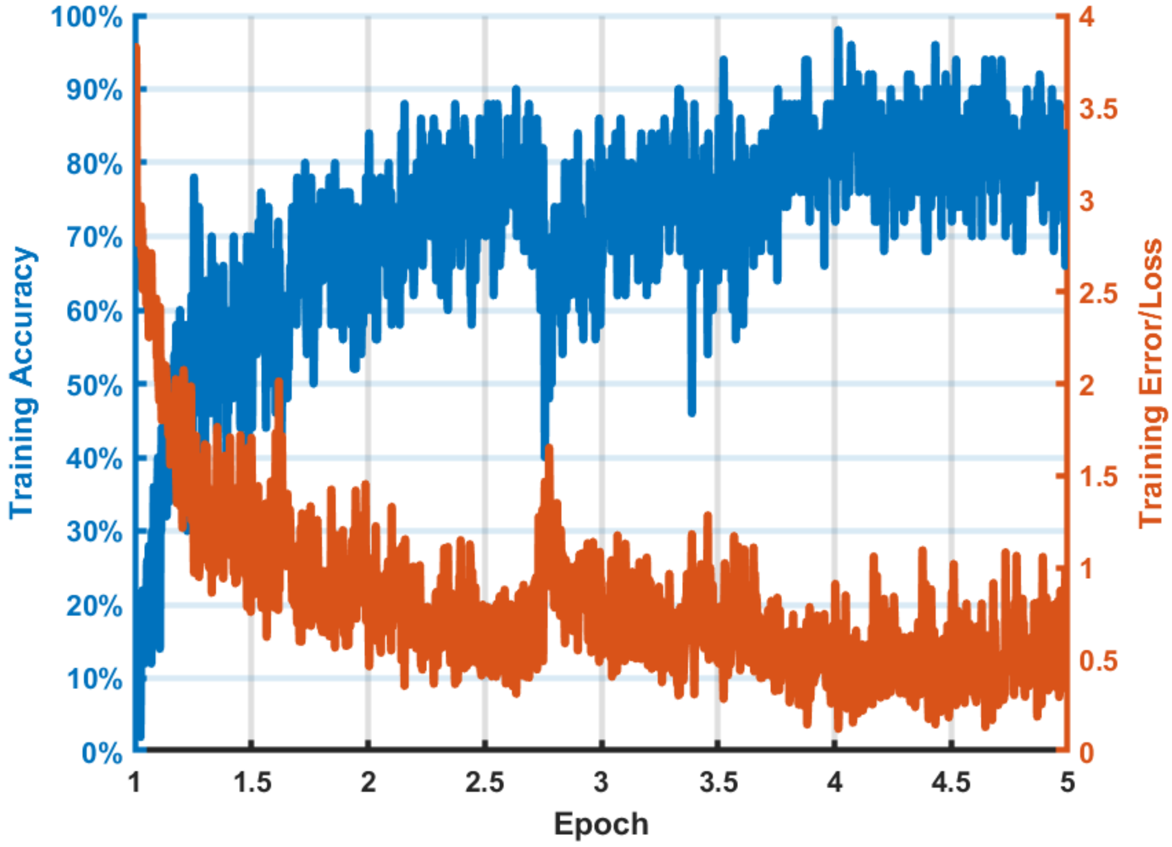
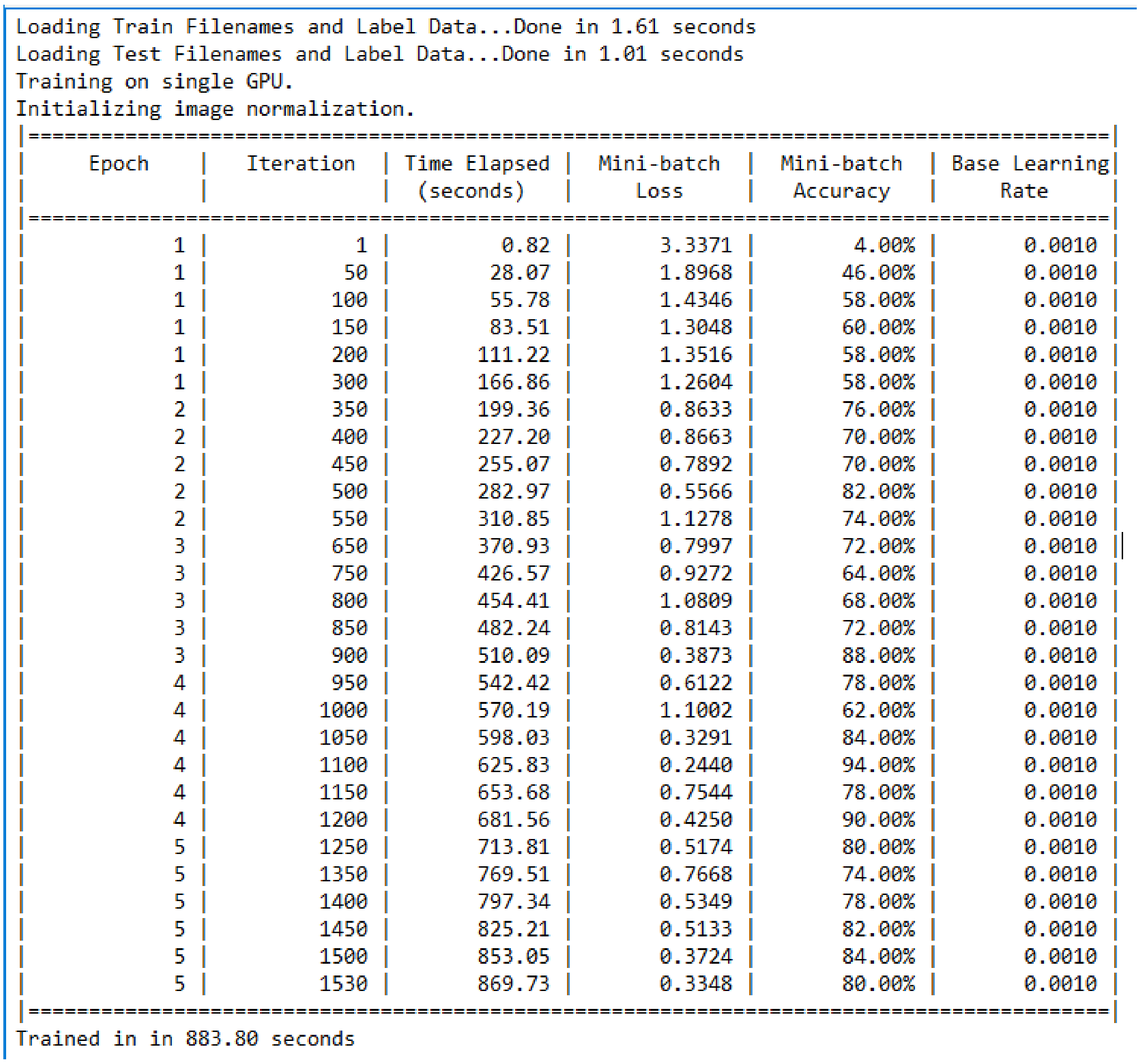


Figure 37: Error plot of Wide network on Original Dataset



```
Loading Train Filenames and Label Data...Done in 1.61 seconds
Loading Test Filenames and Label Data...Done in 1.01 seconds
Training on single GPU.
Initializing image normalization.
|=======================================================================================|
|     Epoch    |   Iteration  | Time Elapsed |  Mini-batch  |  Mini-batch  | Base Learning|
|              |              |  (seconds)   |     Loss     |   Accuracy   |     Rate     |
|=======================================================================================|
|            1 |            1 |         0.82 |       3.3371 |        4.00% |       0.0010 |
|            1 |           50 |        28.07 |       1.8968 |       46.00% |       0.0010 |
|            1 |          100 |        55.78 |       1.4346 |       58.00% |       0.0010 |
|            1 |          150 |        83.51 |       1.3048 |       60.00% |       0.0010 |
|            1 |          200 |       111.22 |       1.3516 |       58.00% |       0.0010 |
|            1 |          300 |       166.86 |       1.2604 |       58.00% |       0.0010 |
|            2 |          350 |       199.36 |       0.8633 |       76.00% |       0.0010 |
|            2 |          400 |       227.20 |       0.8663 |       70.00% |       0.0010 |
|            2 |          450 |       255.07 |       0.7892 |       70.00% |       0.0010 |
|            2 |          500 |       282.97 |       0.5566 |       82.00% |       0.0010 |
|            2 |          550 |       310.85 |       1.1278 |       74.00% |       0.0010 |
|            3 |          650 |       370.93 |       0.7997 |       72.00% |       0.0010 ||
|            3 |          750 |       426.57 |       0.9272 |       64.00% |       0.0010 |
|            3 |          800 |       454.41 |       1.0809 |       68.00% |       0.0010 |
|            3 |          850 |       482.24 |       0.8143 |       72.00% |       0.0010 |
|            3 |          900 |       510.09 |       0.3873 |       88.00% |       0.0010 |
|            4 |          950 |       542.42 |       0.6122 |       78.00% |       0.0010 |
|            4 |         1000 |       570.19 |       1.1002 |       62.00% |       0.0010 |
|            4 |         1050 |       598.03 |       0.3291 |       84.00% |       0.0010 |
|            4 |         1100 |       625.83 |       0.2440 |       94.00% |       0.0010 |
|            4 |         1150 |       653.68 |       0.7544 |       78.00% |       0.0010 |
|            4 |         1200 |       681.56 |       0.4250 |       90.00% |       0.0010 |
|            5 |         1250 |       713.81 |       0.5174 |       80.00% |       0.0010 |
|            5 |         1350 |       769.51 |       0.7668 |       74.00% |       0.0010 |
|            5 |         1400 |       797.34 |       0.5349 |       78.00% |       0.0010 |
|            5 |         1450 |       825.21 |       0.5133 |       82.00% |       0.0010 |
|            5 |         1500 |       853.05 |       0.3724 |       84.00% |       0.0010 |
|            5 |         1530 |       869.73 |       0.3348 |       80.00% |       0.0010 |
|=======================================================================================|
Trained in in 883.80 seconds
```

Figure 38: Mini batch accuracy and time results of Wide network on Original Dataset

3.4 Skinny network

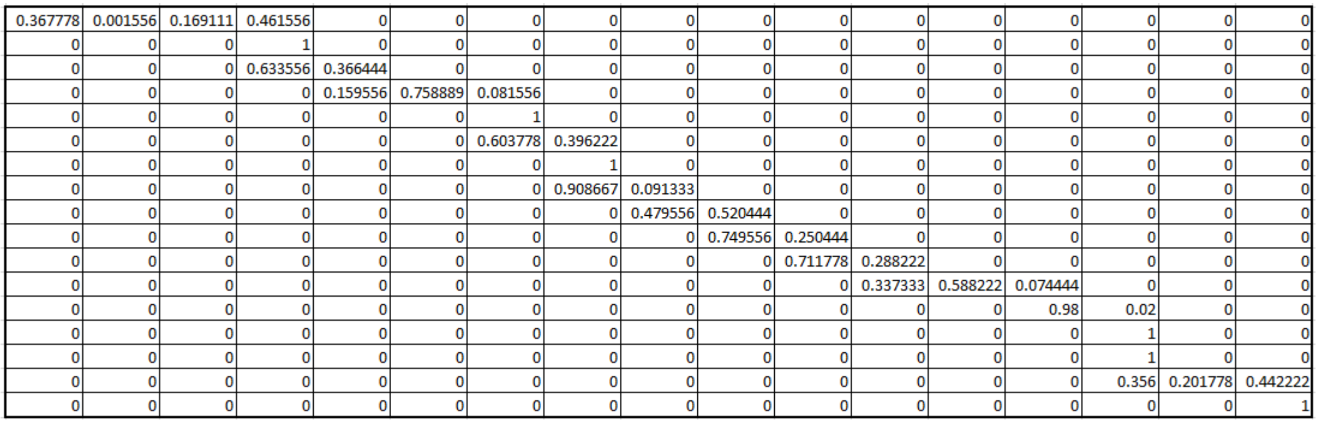
3.4.1 Skinny network on Augmented Dataset

Unlike the wide network, the skinny network was run only on a single configuration. The configuration is as shown in 7. Table 4 shows the accuracy obtained on training, testing and validation. Here, the learning parameter α was equal to 1×10^{-3} and 10 number of Epochs were used to run on a batch size of 200.

	Configuration 7
Training	23.68
Testing	23.77
Validation	23.82

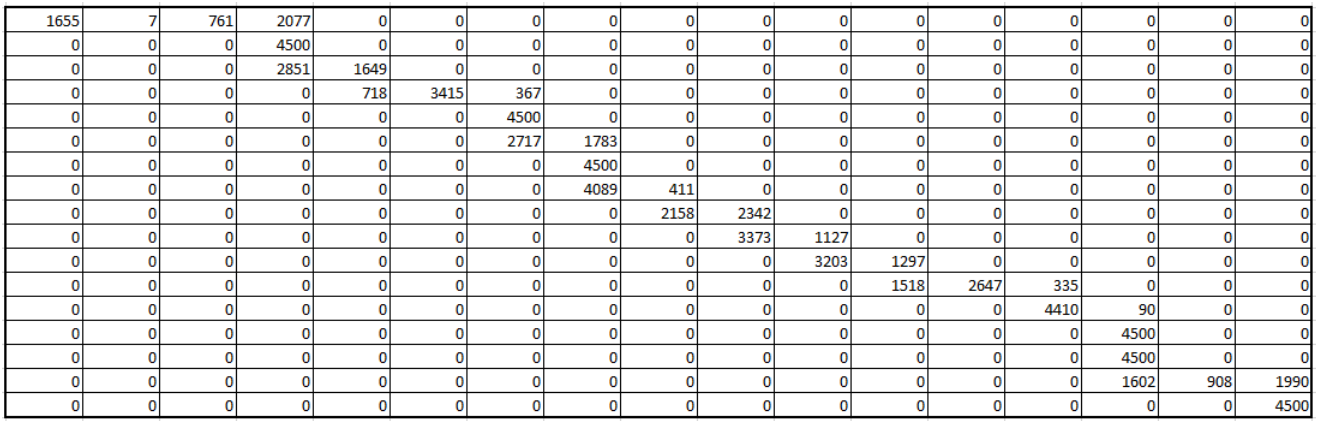
Table 4: Accuracy for Skinny Network

The classification and confusion matrices for training, testing and validation are as shown in 39, 40 and 41 respectively.



0.367778	0.001556	0.169111	0.461556	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0.633556	0.366444	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0.159556	0.758889	0.081556	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0.603778	0.396222	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0.908667	0.091333	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0.479556	0.520444	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0.749556	0.250444	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0.711778	0.288222	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0.337333	0.588222	0.074444	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0.98	0.02	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.356	0.201778	0.442222
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

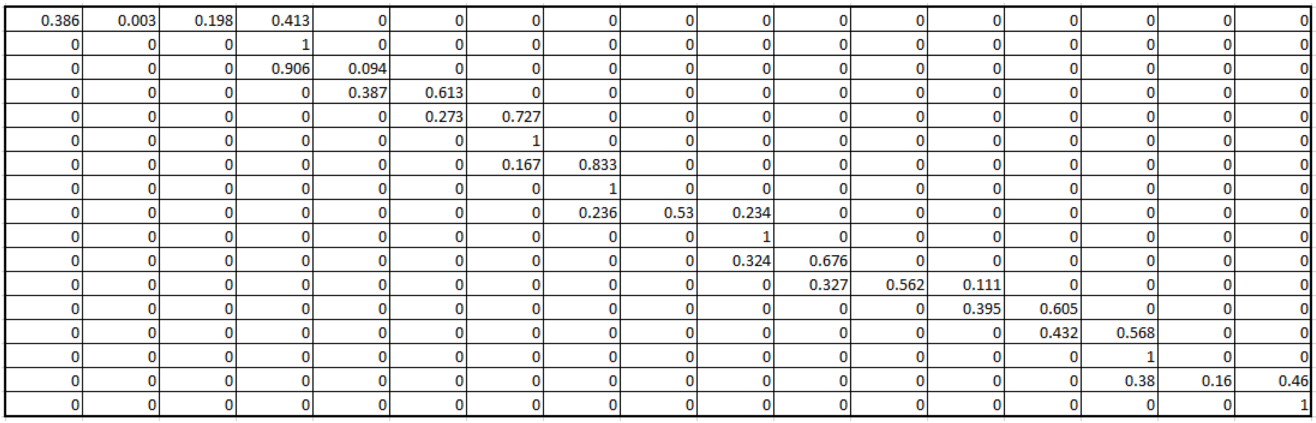
(a) Classification Matrix



1655	7	761	2077	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	4500	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	2851	1649	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	718	3415	367	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	4500	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	2717	1783	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	4500	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	4089	411	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	2158	2342	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	3373	1127	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	3203	1297	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1518	2647	335	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	4410	90	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	4500	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	4500	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1602	908	1990
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4500

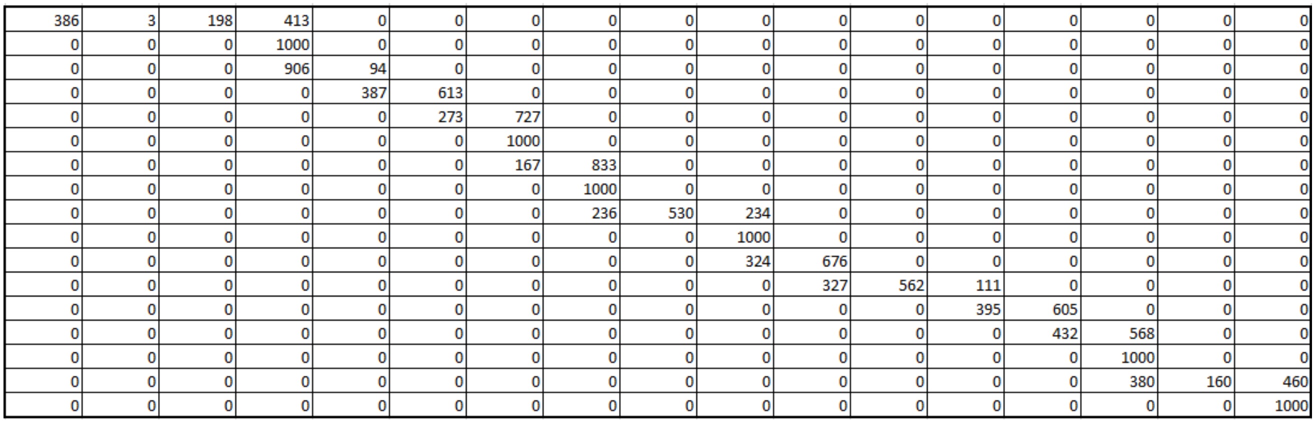
(b) Confusion Matrix

Figure 39: Skinny network: Training on Augmented Dataset



0.386	0.003	0.198	0.413	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0.906	0.094	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0.387	0.613	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0.273	0.727	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0.167	0.833	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0.236	0.53	0.234	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0.324	0.676	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0.327	0.562	0.111	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0.395	0.605	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0.432	0.568	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.38	0.16	0.46
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

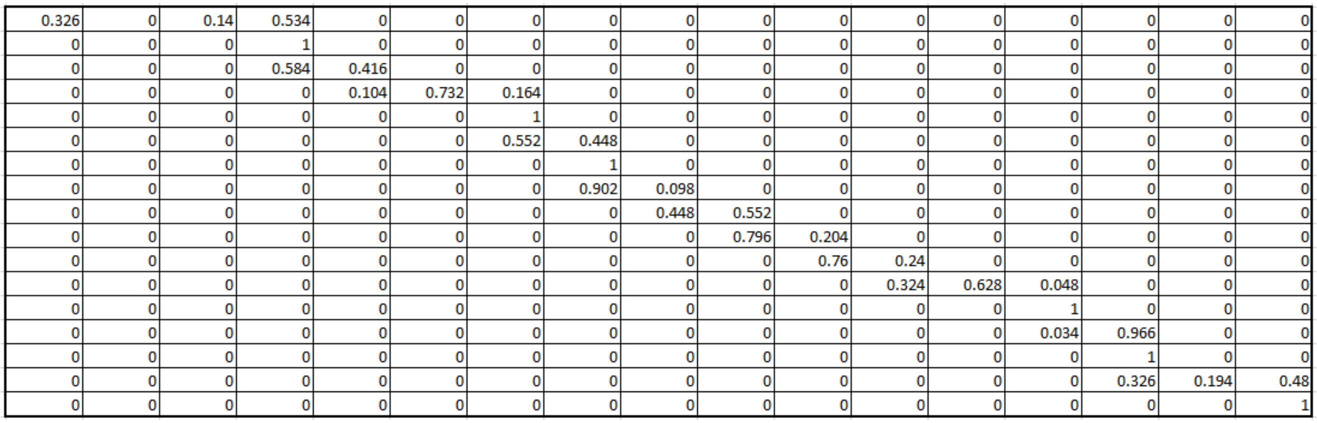
(a) Classification Matrix



386	3	198	413	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1000	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	906	94	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	387	613	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	273	727	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1000	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	167	833	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1000	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	236	530	234	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1000	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	324	676	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	327	562	111	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	395	605	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	432	568	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1000	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	380	160	460
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1000

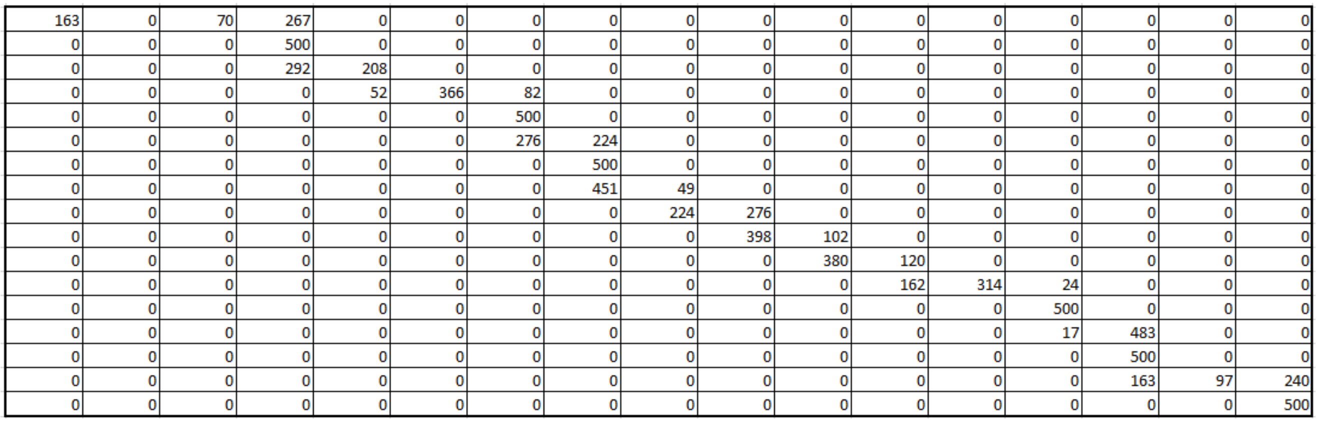
(b) Confusion Matrix

Figure 40: Skinny network: Testing on Augmented Dataset



0.326	0	0.14	0.534	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0.584	0.416	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0.104	0.732	0.164	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0.552	0.448	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0.902	0.098	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0.448	0.552	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0.796	0.204	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0.76	0.24	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0.324	0.628	0.048	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0.034	0.966	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.326	0.194	0.48
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

(a) Classification Matrix



163	0	70	267	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	500	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	292	208	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	52	366	82	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	500	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	276	224	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	500	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	451	49	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	224	276	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	398	102	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	380	120	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	162	314	24	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	500	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	17	483	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	500	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	163	97	240
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	500

(b) Confusion Matrix

Figure 41: Skinny network: Validation on Augmented Dataset

The error plot is as shown in Fig 42. It can be observed that the loss decreases while the accuracy is increasing at a slow rate. This is as expected. However, there are spikes in between which peaks the error loss to very high values. These are anomalies which indicates that the network is not good and a better set of configuration is to be chosen.

The command window recording of mini batch accuracy and the time taken for the network to completely train and run is given by fig 43. Please note that only the final record of each epoch has been shown in the report.

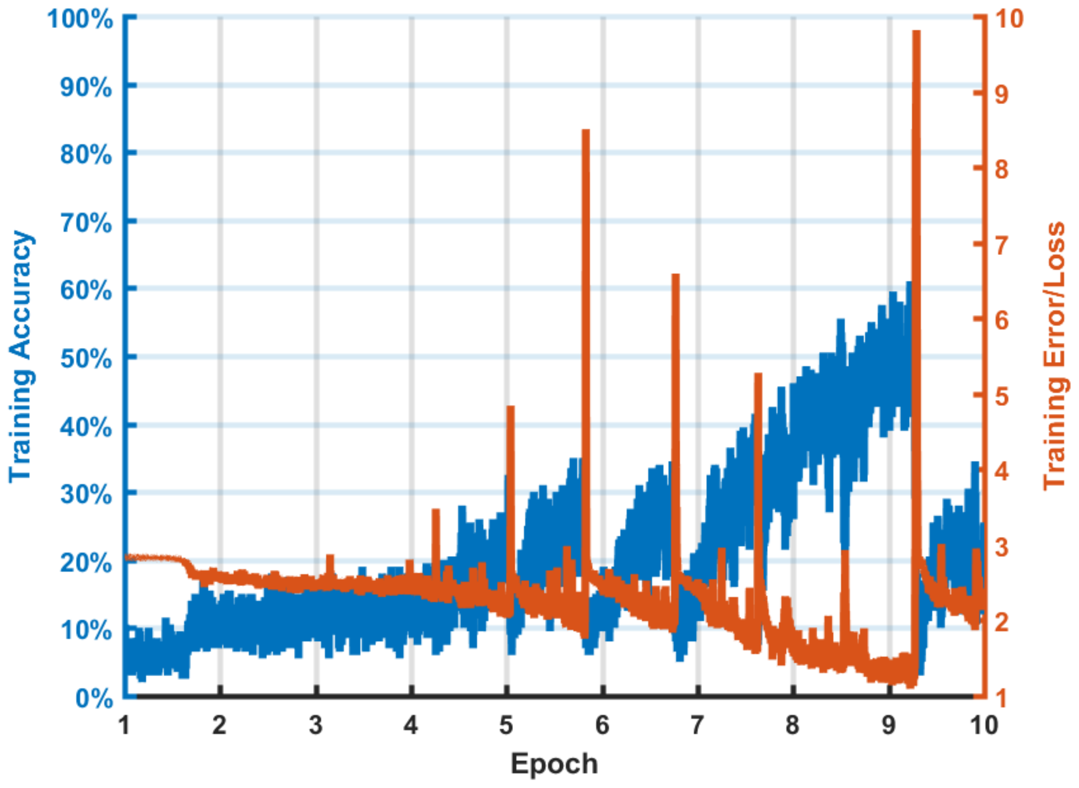
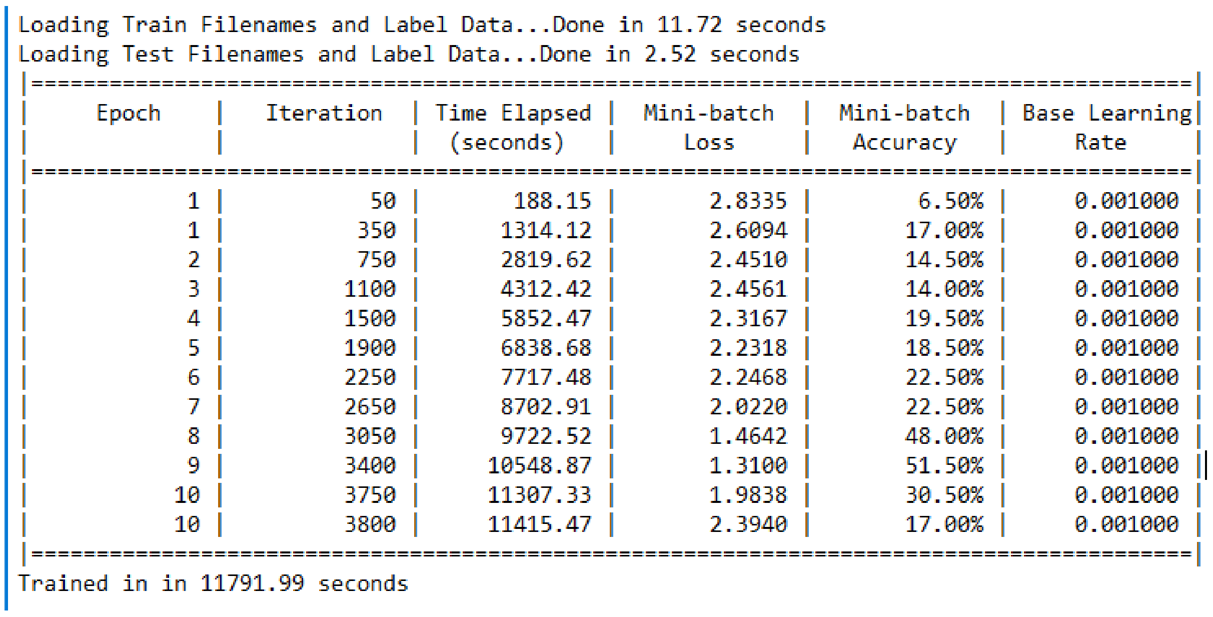


Figure 42: Error plot of skinny network on Augmented Dataset



```
Loading Train Filenames and Label Data...Done in 11.72 seconds
Loading Test Filenames and Label Data...Done in 2.52 seconds
|=========================================================================================|
|     Epoch     |   Iteration   | Time Elapsed  |  Mini-batch   |  Mini-batch   | Base Learning|
|               |               |  (seconds)    |     Loss      |   Accuracy    |     Rate     |
|=========================================================================================|
|             1 |            50 |        188.15 |        2.8335 |         6.50% |     0.001000 |
|             1 |           350 |       1314.12 |        2.6094 |        17.00% |     0.001000 |
|             2 |           750 |       2819.62 |        2.4510 |        14.50% |     0.001000 |
|             3 |          1100 |       4312.42 |        2.4561 |        14.00% |     0.001000 |
|             4 |          1500 |       5852.47 |        2.3167 |        19.50% |     0.001000 |
|             5 |          1900 |       6838.68 |        2.2318 |        18.50% |     0.001000 |
|             6 |          2250 |       7717.48 |        2.2468 |        22.50% |     0.001000 |
|             7 |          2650 |       8702.91 |        2.0220 |        22.50% |     0.001000 |
|             8 |          3050 |       9722.52 |        1.4642 |        48.00% |     0.001000 |
|             9 |          3400 |      10548.87 |        1.3100 |        51.50% |     0.001000 |
|            10 |          3750 |      11307.33 |        1.9838 |        30.50% |     0.001000 |
|            10 |          3800 |      11415.47 |        2.3940 |        17.00% |     0.001000 |
|=========================================================================================|
Trained in in 11791.99 seconds
```

Figure 43: Mini batch accuracy and time results of skinny network

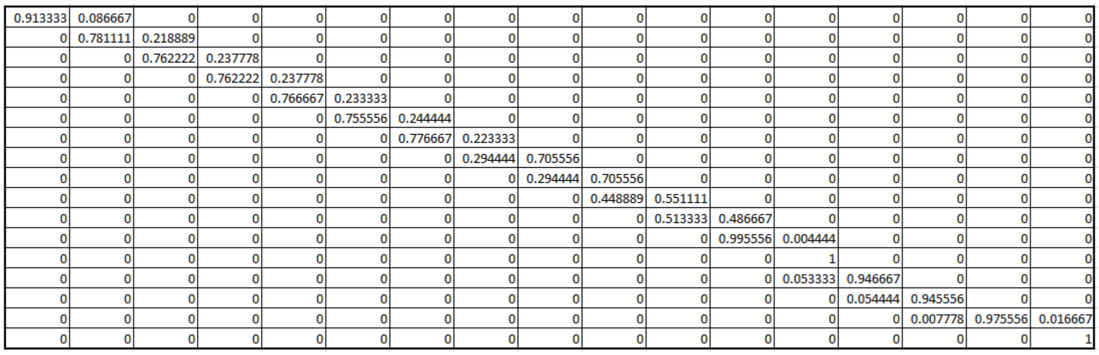
3.4.2 Skinny network on Original Dataset

After having trained the skinny network, the layers were loaded onto the workspace by using the last checkpoint and thereby re training the network. After having done this, we obtain the training,testing and validation accuracies and they are observed to be much higher than when compared to the augmented dataset. This can be observed from Table 4 and 5. This is due to the fact that having trained the network with large amount of data, the network would have learnt the basic images from the original dataset. That is the reason we achieve high accuracy.

	On Augmented dataset	On Original (17,000) dataset
Training	23.68	90.84
Testing	23.77	90.01
Validation	23.82	89.82

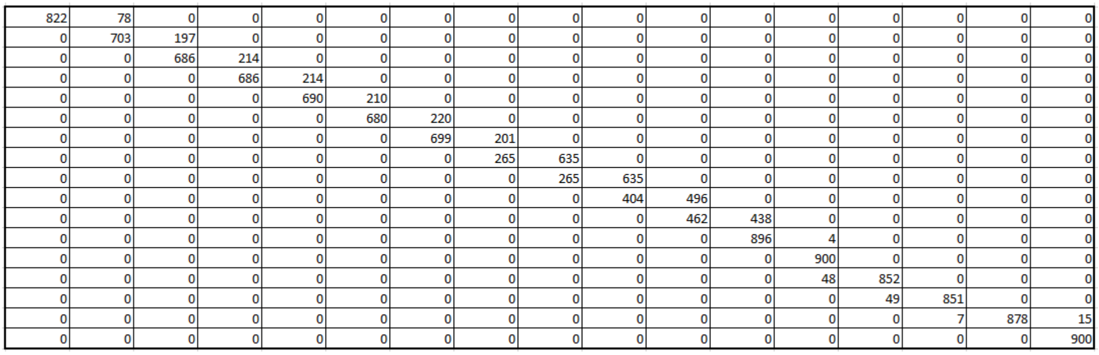
Table 5: Accuracy for skinny Network with $\alpha = 1 \times 10^{-3}$ and number of Epochs = 5

The classification and confusion matrices for training, testing and validation are as shown in 44, 45 and 46 respectively.



0.913333	0.086667	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0.781111	0.218889	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0.762222	0.237778	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0.762222	0.237778	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0.766667	0.233333	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0.755556	0.244444	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0.776667	0.223333	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0.294444	0.705556	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0.294444	0.705556	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0.448889	0.551111	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0.513333	0.486667	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0.995556	0.004444	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0.053333	0.946667	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0.054444	0.945556	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.007778	0.975556	0.016667
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

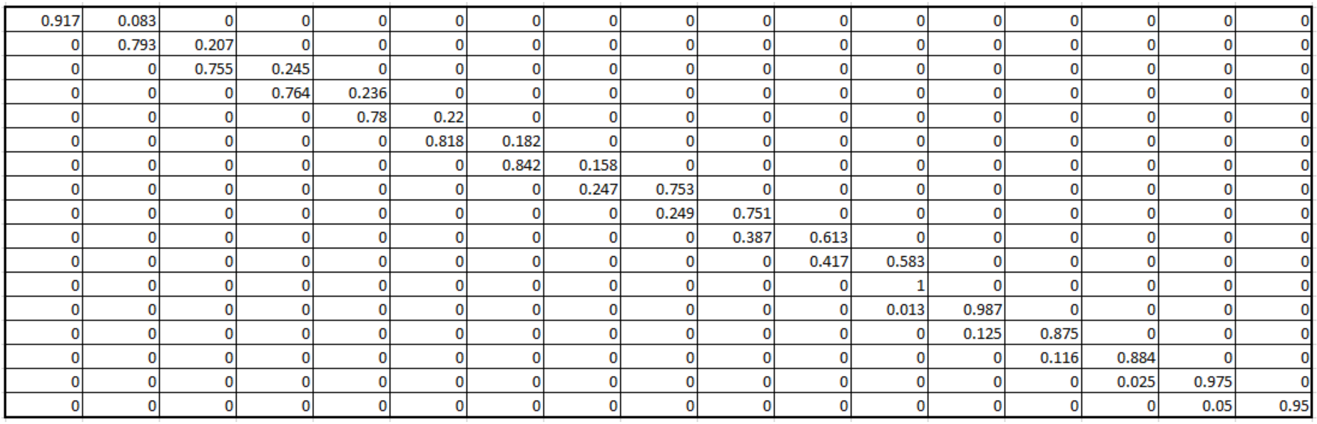
(a) Classification Matrix



822	78	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	703	197	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	686	214	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	686	214	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	690	210	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	680	220	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	699	201	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	265	635	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	265	635	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	404	496	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	462	438	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	896	4	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	900	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	48	852	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	49	851	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	878	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	900

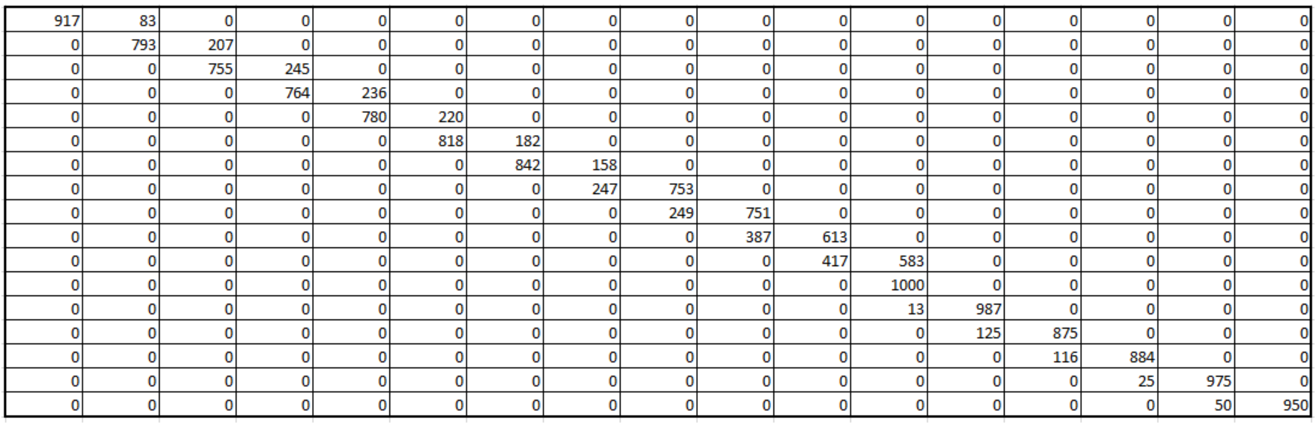
(b) Confusion Matrix

Figure 44: Skinny network: Training on Original Dataset



0.917	0.083	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0.793	0.207	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0.755	0.245	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0.764	0.236	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0.78	0.22	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0.818	0.182	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0.842	0.158	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0.247	0.753	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0.249	0.751	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0.387	0.613	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0.417	0.583	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0.013	0.987	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0.125	0.875	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0.116	0.884	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.025	0.975	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.05	0.95

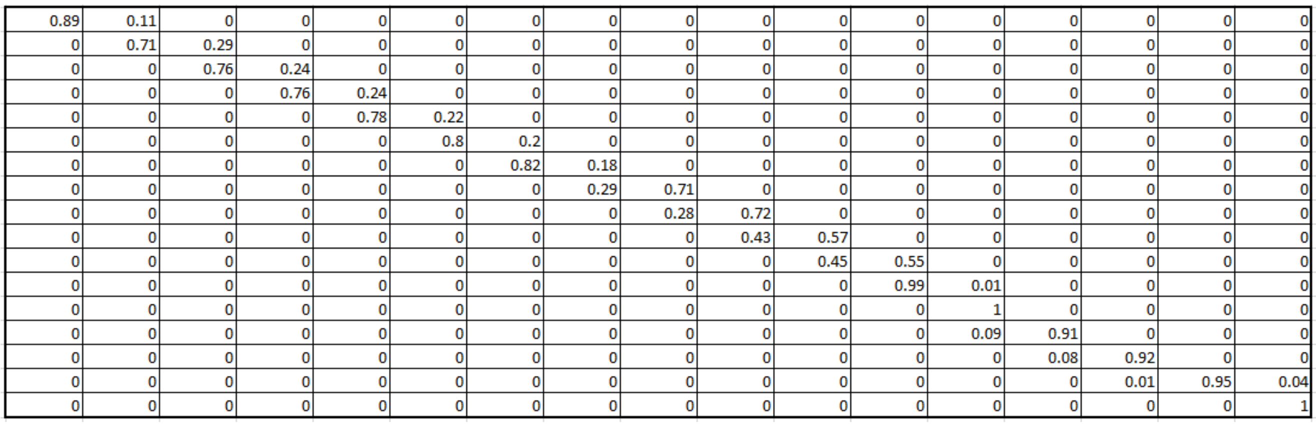
(a) Classification Matrix



917	83	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	793	207	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	755	245	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	764	236	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	780	220	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	818	182	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	842	158	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	247	753	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	249	751	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	387	613	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	417	583	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1000	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	13	987	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	125	875	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	116	884	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	25	975	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	50	950

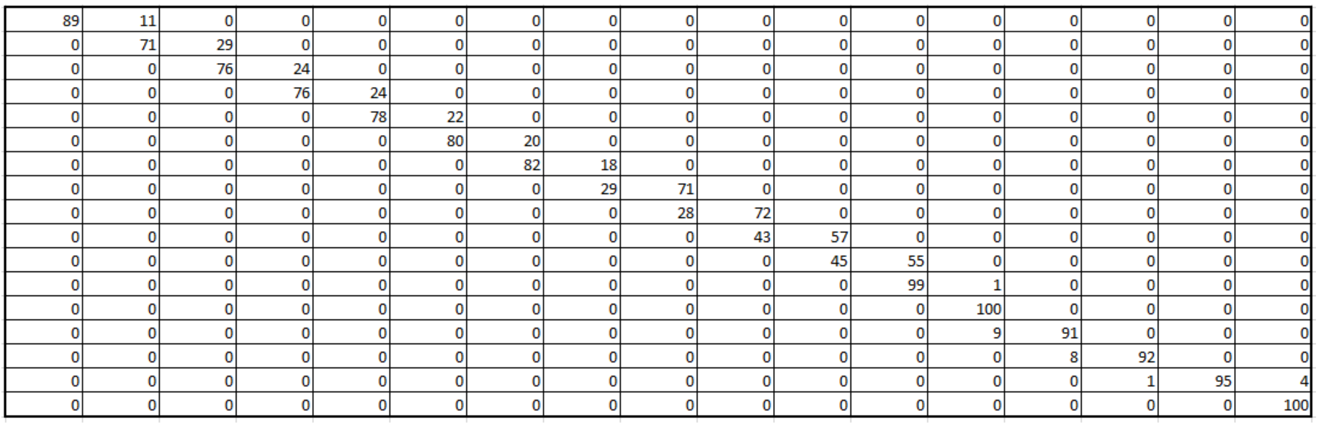
(b) Confusion Matrix

Figure 45: Skinny network: Testing on Original Dataset



0.89	0.11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0.71	0.29	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0.76	0.24	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0.76	0.24	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0.78	0.22	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0.8	0.2	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0.82	0.18	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0.29	0.71	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0.28	0.72	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0.43	0.57	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0.45	0.55	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0.99	0.01	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0.09	0.91	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0.08	0.92	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.01	0.95	0.04
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

(a) Classification Matrix



89	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	71	29	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	76	24	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	76	24	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	78	22	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	80	20	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	82	18	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	29	71	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	28	72	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	43	57	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	45	55	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	99	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	100	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	9	91	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	8	92	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	95	4
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100

(b) Confusion Matrix

Figure 46: Skinny network: Validation on Original Dataset

The error plot is as shown in Fig 47

The command window recording of mini batch accuracy and the time taken for the network to completely train and run is given by fig 48.

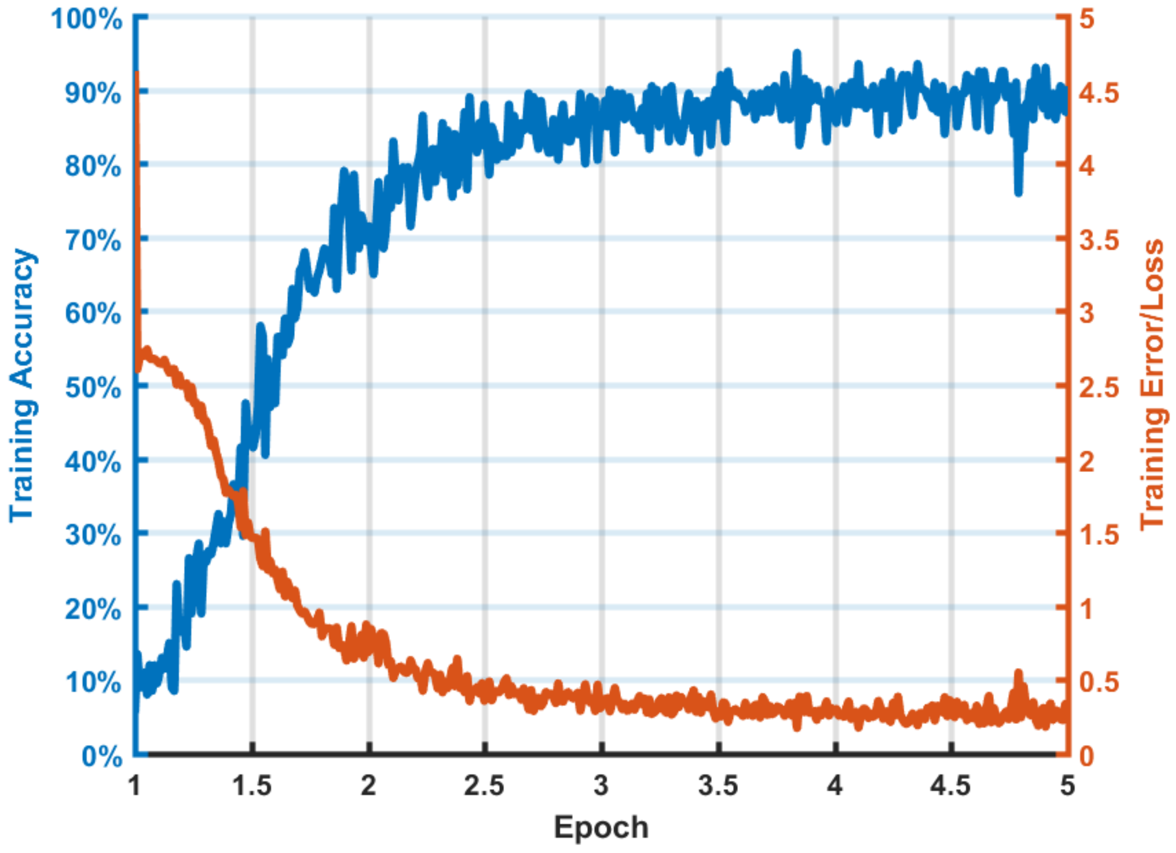
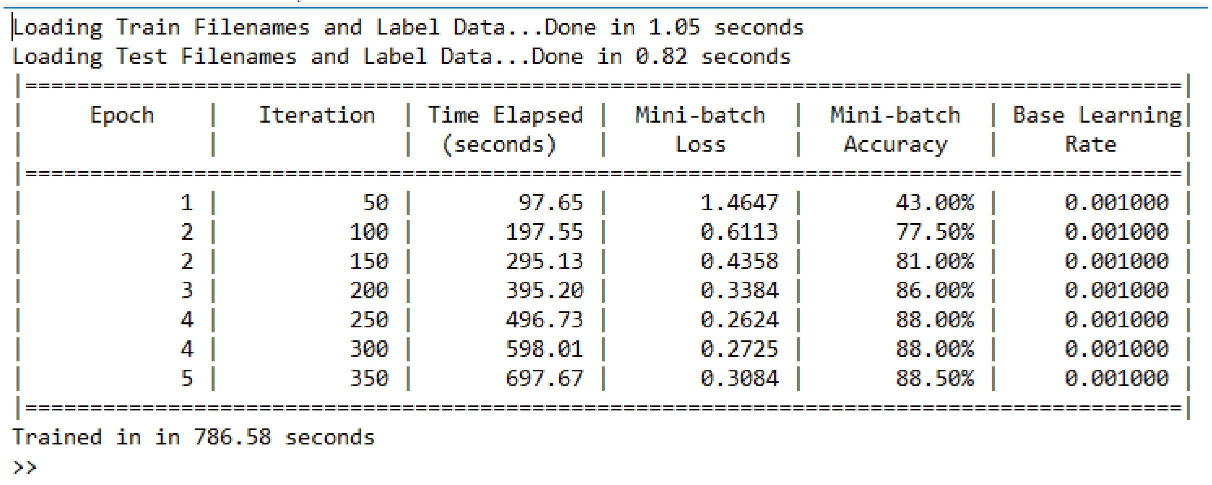


Figure 47: Error plot of Skinny network on Original Dataset



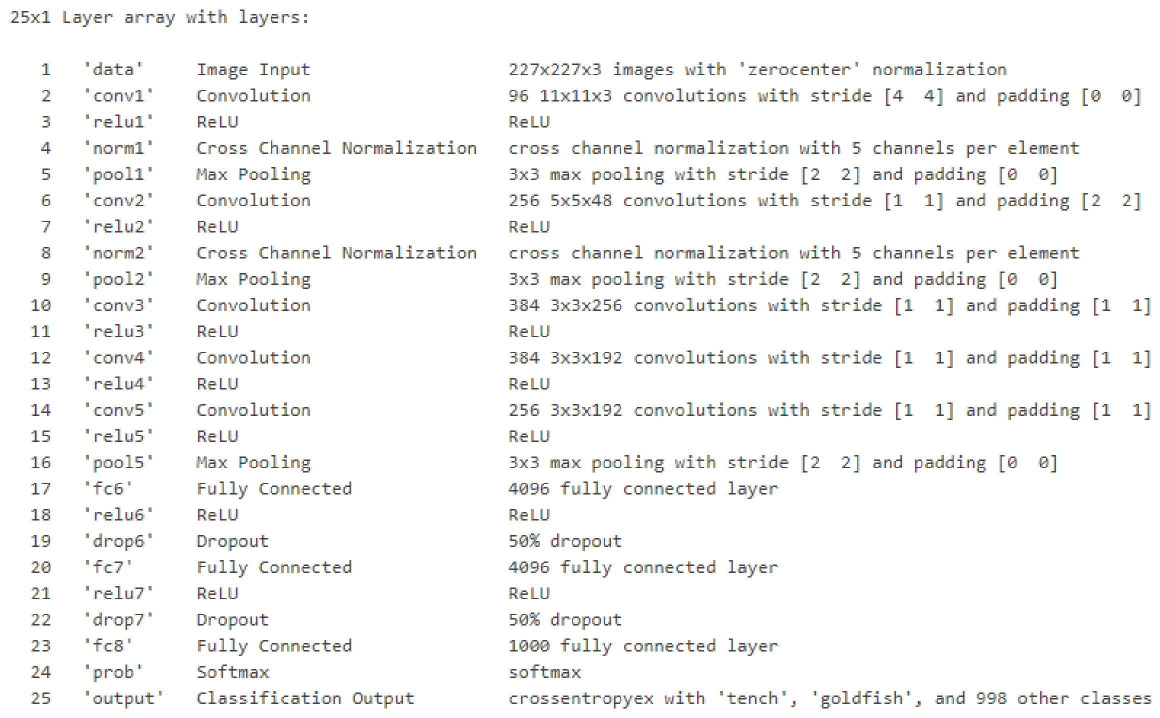
```
Loading Train Filenames and Label Data...Done in 1.05 seconds
Loading Test Filenames and Label Data...Done in 0.82 seconds
|=========================================================================================|
|     Epoch    |  Iteration  | Time Elapsed |  Mini-batch  |  Mini-batch  | Base Learning|
|              |             |  (seconds)   |     Loss     |   Accuracy   |     Rate     |
|=========================================================================================|
|            1 |          50 |        97.65 |       1.4647 |       43.00% |     0.001000 |
|            2 |         100 |       197.55 |       0.6113 |       77.50% |     0.001000 |
|            2 |         150 |       295.13 |       0.4358 |       81.00% |     0.001000 |
|            3 |         200 |       395.20 |       0.3384 |       86.00% |     0.001000 |
|            4 |         250 |       496.73 |       0.2624 |       88.00% |     0.001000 |
|            4 |         300 |       598.01 |       0.2725 |       88.00% |     0.001000 |
|            5 |         350 |       697.67 |       0.3084 |       88.50% |     0.001000 |
|=========================================================================================|
Trained in in 786.58 seconds
>>
```

Figure 48: Mini batch accuracy and time results of skinny network on Original Dataset

4 Extra Credits

4.1 Transfer Learning using AlexNet

AlexNet is one of the largest Convolutional Neural Networks. It consists of 5 convolutional layers and 3 Fully Connected Layers. The Network has been trained upon the ImageNet dataset that consists of about 15 Million labelled high resolution images and roughly 22,000 categories.
This part of the project deals in understanding transfer learning. Transfer Learning is when we keep the initial pretrained layers of the AlexNet as it is and plug in the FullyConnected(17), the softmax layer and the classification layer at the end. Thus we could expect to get a better accuracy as the pretrained networks weights are fine-tuned to identify a large and varied set of patterns. Figure 49 shows the 25 Layers of the AlexNet.



```
  25x1 Layer array with layers:

     1   'data'     Image Input                   227x227x3 images with 'zerocenter' normalization
     2   'conv1'    Convolution                   96 11x11x3 convolutions with stride [4  4] and padding [0  0]
     3   'relu1'    ReLU                          ReLU
     4   'norm1'    Cross Channel Normalization   cross channel normalization with 5 channels per element
     5   'pool1'    Max Pooling                   3x3 max pooling with stride [2  2] and padding [0  0]
     6   'conv2'    Convolution                   256 5x5x48 convolutions with stride [1  1] and padding [2  2]
     7   'relu2'    ReLU                          ReLU
     8   'norm2'    Cross Channel Normalization   cross channel normalization with 5 channels per element
     9   'pool2'    Max Pooling                   3x3 max pooling with stride [2  2] and padding [0  0]
    10   'conv3'    Convolution                   384 3x3x256 convolutions with stride [1  1] and padding [1  1]
    11   'relu3'    ReLU                          ReLU
    12   'conv4'    Convolution                   384 3x3x192 convolutions with stride [1  1] and padding [1  1]
    13   'relu4'    ReLU                          ReLU
    14   'conv5'    Convolution                   256 3x3x192 convolutions with stride [1  1] and padding [1  1]
    15   'relu5'    ReLU                          ReLU
    16   'pool5'    Max Pooling                   3x3 max pooling with stride [2  2] and padding [0  0]
    17   'fc6'      Fully Connected               4096 fully connected layer
    18   'relu6'    ReLU                          ReLU
    19   'drop6'    Dropout                       50% dropout
    20   'fc7'      Fully Connected               4096 fully connected layer
    21   'relu7'    ReLU                          ReLU
    22   'drop7'    Dropout                       50% dropout
    23   'fc8'      Fully Connected               1000 fully connected layer
    24   'prob'     Softmax                       softmax
    25   'output'   Classification Output         crossentropyex with 'tench', 'goldfish', and 998 other classes
```

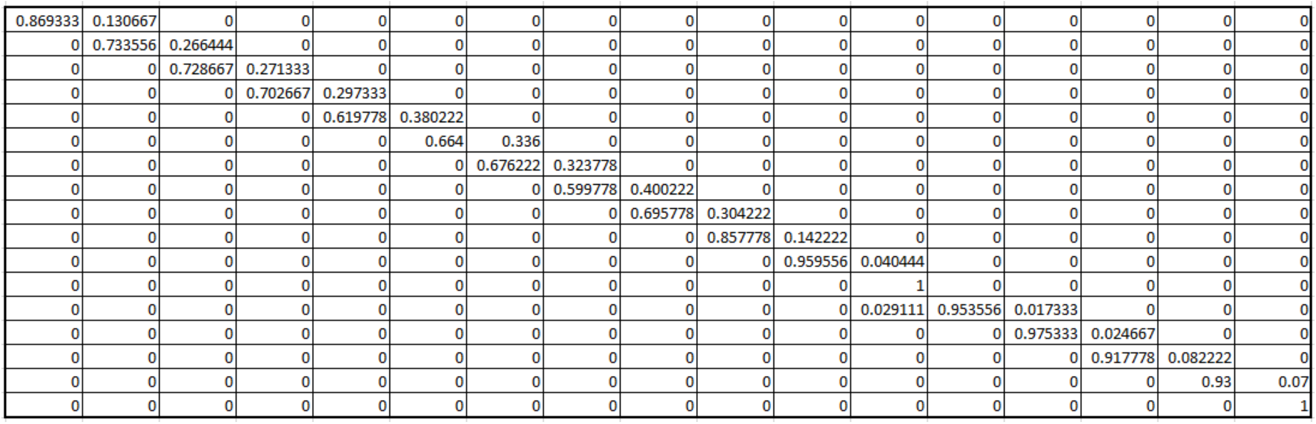
Figure 49: AlexNet network architecture

The input layer for this architecture is $227 \times 227 \times 3$. Therefore we create the AlexNet dataset as described in 2.1.3. Here, we resize the augmented dataset and replicate the channels to obtain the desired size. It can be seen from the error plot shown in fig **??** that the transfer learning boosts the accuracy by a large margin. The loss function continuously decreases and converges. The accuracies are as shown in table 6.

We obtain the following classification and confusion matrices of training, testing and validation as shown in Fig 50, 51 and 52.

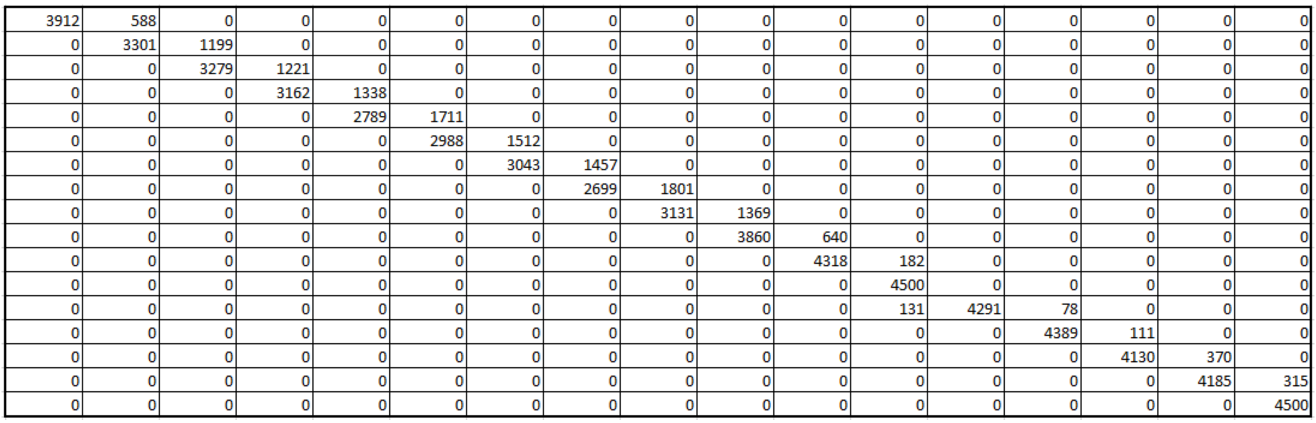
	AlexNet
Training	91.54
Testing	84.68
Validation	87.85

Table 6: Accuracy for AlexNet with $\alpha = 5 \times 10^{-4}$ and number of Epochs = 10



0.869333	0.130667	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0.733556	0.266444	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0.728667	0.271333	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0.702667	0.297333	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0.619778	0.380222	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0.664	0.336	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0.676222	0.323778	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0.599778	0.400222	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0.695778	0.304222	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0.857778	0.142222	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0.959556	0.040444	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0.029111	0.953556	0.017333	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0.975333	0.024667	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.917778	0.082222	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.93	0.07
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

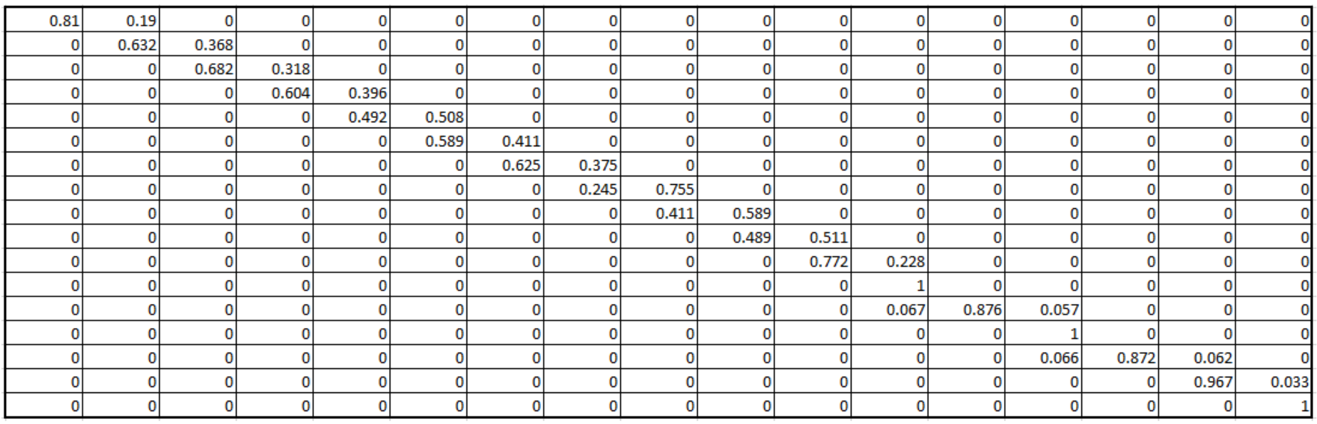
(a) Classification Matrix



3912	588	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	3301	1199	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	3279	1221	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	3162	1338	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	2789	1711	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	2988	1512	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	3043	1457	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	2699	1801	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	3131	1369	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	3860	640	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	4318	182	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	4500	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	131	4291	78	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	4389	111	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	4130	370	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4185	315
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4500

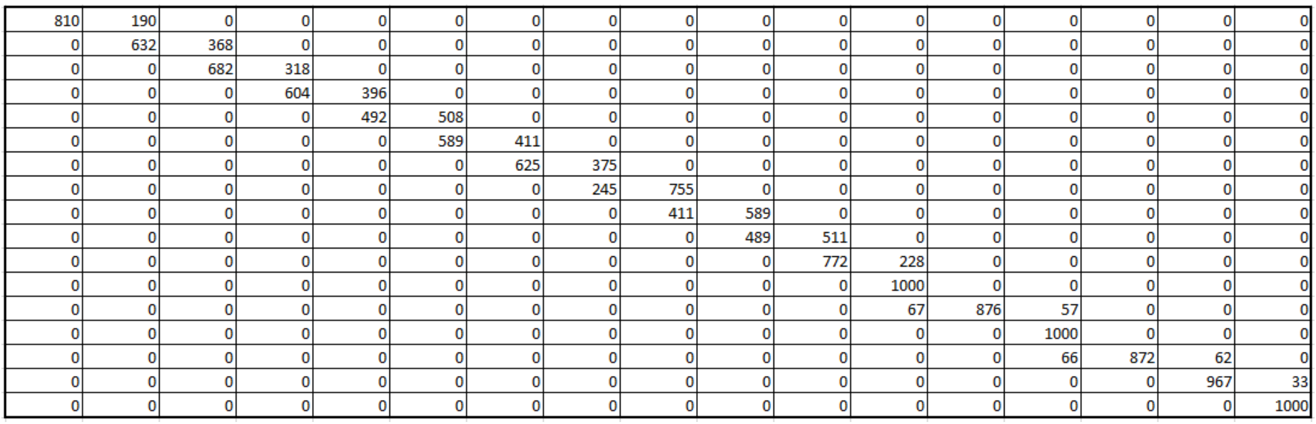
(b) Confusion Matrix

Figure 50: Alexnet: Training



0.81	0.19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0.632	0.368	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0.682	0.318	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0.604	0.396	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0.492	0.508	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0.589	0.411	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0.625	0.375	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0.245	0.755	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0.411	0.589	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0.489	0.511	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0.772	0.228	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0.067	0.876	0.057	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0.066	0.872	0.062	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.967	0.033
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

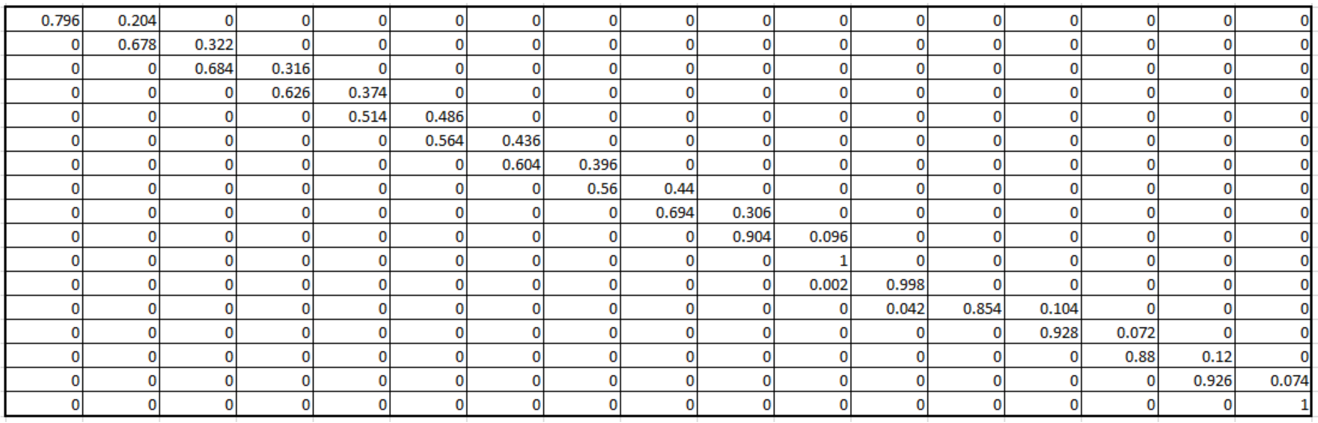
(a) Classification Matrix



810	190	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	632	368	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	682	318	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	604	396	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	492	508	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	589	411	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	625	375	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	245	755	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	411	589	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	489	511	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	772	228	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1000	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	67	876	57	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1000	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	66	872	62	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	967	33
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1000

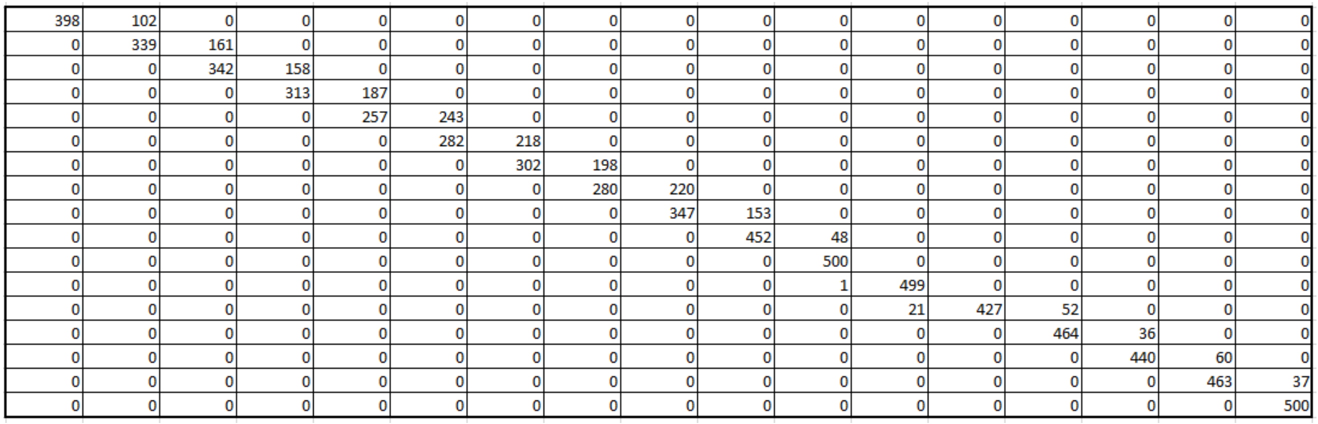
(b) Confusion Matrix

Figure 51: alexnet: Testing



0.796	0.204	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0.678	0.322	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0.684	0.316	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0.626	0.374	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0.514	0.486	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0.564	0.436	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0.604	0.396	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0.56	0.44	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0.694	0.306	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0.904	0.096	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0.002	0.998	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0.042	0.854	0.104	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0.928	0.072	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.88	0.12	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.926	0.074
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

(a) Classification Matrix



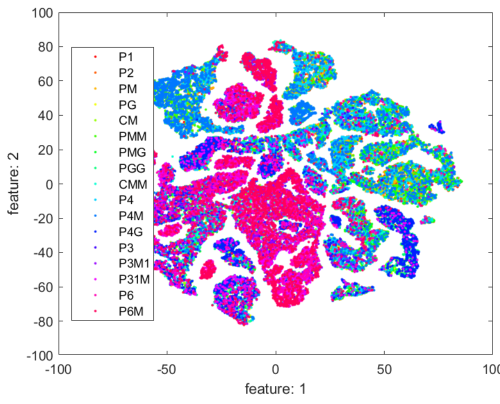
398	102	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	339	161	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	342	158	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	313	187	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	257	243	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	282	218	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	302	198	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	280	220	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	347	153	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	452	48	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	500	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	499	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	21	427	52	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	464	36	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	440	60	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	463	37
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	500

(b) Confusion Matrix

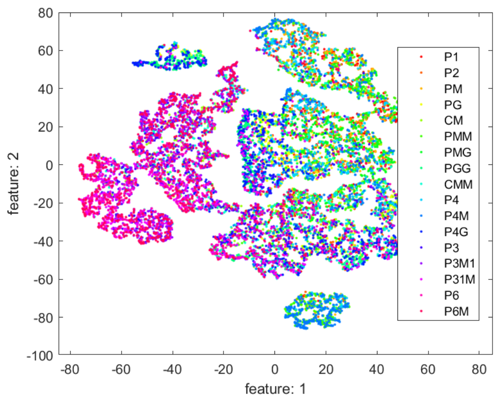
Figure 52: alexnet: Validation

4.2 t-SNE multidimensional reduction on fully connected layer activations of network trained on the augmentations

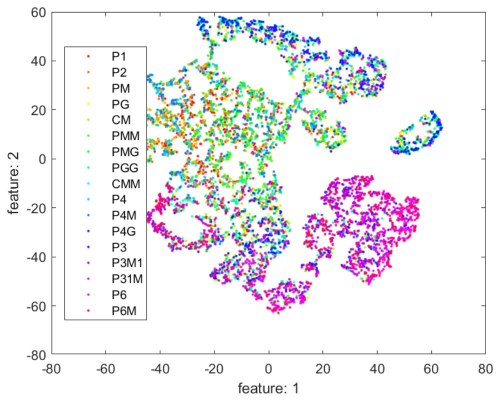
As the Section name specifies, we perform t-SNE multidimensional reduction on fully connected layer activations of wide and skinny network. t-SNE is an algorithm for dimensionality reduction that is suited for visualizing high dimensional data. It works on the principle of getting the high dimensional points to nearby low dimensional points. t-SNE is similar to Fisher LDA as both are dimensionality reduction algorithms. In order to implement this, we input the activations of the train, test and validation set to the t-SNE function on MATLAB. This outputs the projected 2-D data, that is, the dimensions are reduced to 2.
From the visualization shown in Fig 53 and Fig 54 that the t-SNE plot for the Skinny network looks better than the one for the wide network. This can be attributed to the better accuracy of the skinny network over the wide network. We can clearly see that the classes are separable and more distinct for the skinny network. The tSNE visualization of Wide network looks like a random cloud of points due to its poor ability to be classified.



(a) Train

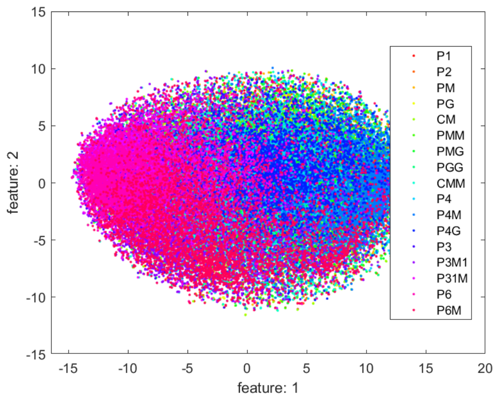


(b) Test

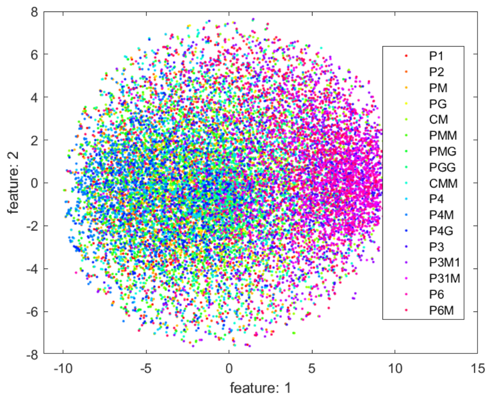


(c) Validation

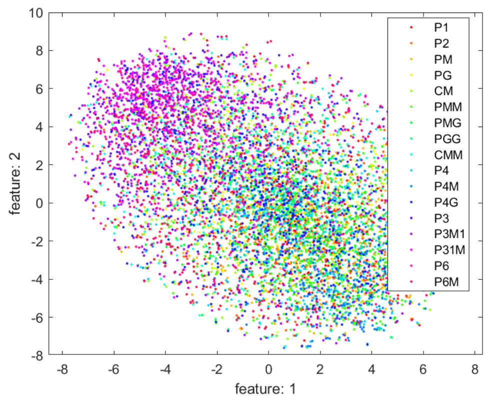
Figure 53: tSNE visualization on Skinny network



(a) Train



(b) Test



(c) Validation

Figure 54: tSNE visualization on Wide network

4.3 Visualization of first layer of filters

Using the MATLAB command cv_layer = deepDreamImage(net,layer,channels), it returns an array of images that activates the channels within the network net of the layer 'layer'. These images highlight the features learned by a network. While I have written the code for accessing the first layer using the deepDreamImage, there is some issue when I load my saved workspace. While it contains the accuracy and other details such as the variable net1 and the layers inside them, however there are no information available inside the layers for some of them. This could probably be an issue of having saved the workspace from the command window. However I am certain that if you run the code, then I will obtain the necessary visualisations. But for the case of the starter code *i.e.* the step 1, I was able to realise it from loading the worked space and running the same code. They are as follows:

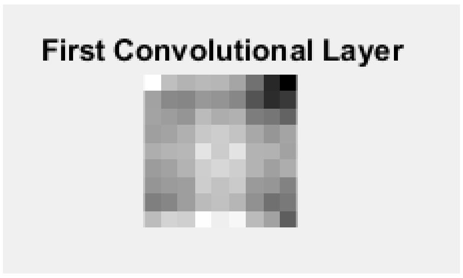


Figure 55: $\alpha = 1 \times 10^{-5}$

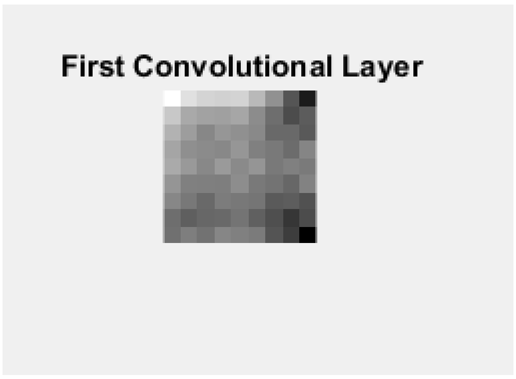


Figure 56: $\alpha = 5 \times 10^{-4}$

5 Conclusion

Based on the objectives performed in this project, the following conclusions can be drawn:

- For any CNN the input layer is always an image and the output is one of the k classes of the image that it is classified into.
- CNN is a complex network, however it is built by a set of simple building blocks.
- In order to obtain maximum accuracy, the network should consist of a large number of data. However, the issue with large data comes with increased computational expense. As the data is increased, the time taken for the network to learn the parameters increases.
- One epoch is one forward propagation plus a backward propagation. As the number of epochs are increased, the network fine tunes the features that it learns until a global minima is achieved. Although, the larger the number of epochs, the longer the network takes to be trained completely.
- To fasten or slow down the process of training, the use of the learning parameter can be made. However, there is no fixed method of determining the optimal learning rate. For each network, the learning rate differs and thereby a programmer needs to tinker with the parameter to obtain an optimal output. If its too small, then the network takes a lot of time and iterations to achieve a decent accuracy while if it is too high, then the gradient descent could overshoot the minima and not achieve the minima, thereby providing a bad accuracy.
- Using the method of transfer learning, the time taken to train the network and the accuracy with which the network is trained can be significantly improved. 3
- Loss is a summation of errors made for each input in the training and validation set. Loss is calculated on training and validation and its interpretation is how well the model is doing for these two sets. We could intuitively say that if the loss is lesser, then the accuracy is higher. However, in machine learning, lower loss could be a result of the network overfitting the data which could be a bad thing for the network and thus act very poorly on the testing dataset. Therefore lesser loss does not necessarily mean higher accuracy.

References

[1] Bishop, Christopher M. *Pattern Recognition and Machine Learning.* 2006.

[2] Prince, Simon JD *Computer vision: models, learning, and inference.* 2012

[3] http://cs231n.github.io/convolutional-networks/comp