Project 4

Purushartha Singh

April 13, 2021

Abstract

The project looks at using different types Convolutional Neural Networks (CNN) to perform a classification task and familiarizing with the different visualization tools that can be used with the model architecture.

Contents

1 Introduction

1.1 Background

Deep learning is a subset of machine learning which is based on artificial neural networks capable of unsupervised learning from unstructured or unlabeled data. The learning can be supervised, unsupervised and semi supervised. The term 'deep' comes from the number of hidden layers in a neural network.

The most popular neural network architecture is Convolutional Neural network."A CNN convolves learned features with input data, and uses 2D convolutional layers, making this architecture well suited to processing 2D data, such as images."[1]The advantage of this type of architecture is that the performance is improved by increasing the data. It requires less preprossessing as compared to other image classification algorithms. Thus, the reduced need of human intervention in feature extraction is another advantage of CNNs.

1.2 Objective

The following were the objectives of the project

1. Running the provided pipeline to get baseline results with the wallpaper dataset.
2. Preprossessing image data to create augmented dataset
3. Building a wide and a skinny network to perform classification on the augmented dataset
4. Compare the results and visualize the network using T-SNE and visualization of the first layer.
5. Use transfer learning from a pre-existing model (Alexnet) and compare with the other models.

1.3 Theory

A convolutional neural network consists of various layers which are all connected starting with the input layer which is the batch of images. This is followed by the following types layers in the Neural network used in the project [2]:

1. **convolution2dLayer:** Defines 2D convolution layer for Convolutional Neural Networks. filterSize specifies the height and width of the filters. It can be a scalar, in which case the filters will have the same height and width, or a vector [h w] where h specifies the height for the filters, and w specifies the width. numFilters specifies the number of filters. The other parameter is the 2 * 2 buffer around the image.
2. **BatchNormalizationLayer:** Creates a batch normalization layer. This type of layer normalizes each channel across a mini-batch. This can be useful in reducing sensitivity to variations within the data.

3. **ReLULayer:** Creates a rectified linear unit layer. This type of layer performs a simple threshold operation, where any input value less than zero will be set to zero.

4. **MaxPooling2DLayer:** A max pooling layer divides the input into rectangular pooling regions, and outputs the maximum of each region. poolSize specifies the width and height of a pooling region. It can be a scalar, in which case the pooling regions will have the same width and height, or a vector [h w] where h specifies the height and w specifies the width. Note that if the 'Stride' dimensions are less than the respective pool dimensions, then the pooling regions will overlap.

5. **fullyConnectedLayer:** Creates a fully connected layer. outputSize specifies the size of the output for the layer. A fully connected layer will multiply the input by a matrix and then add a bias vector.

6. **SoftmaxLayer:** Creates a softmax layer. This layer is useful for classification problems as it normalizes the distribution so that all values add up to 1, making it a probability distribution.

2 Data

2.1 Wallpaper Dataset

The original dataset provided is the Wallpaper dataset which was previously used in the Project 2. This dataset has a set of 17 different types of wallpapers. Each set contains 1000 training and testing images of 256 * 256 pixels. The images are monocolor. This acts as the base dataset which is preprocessed to get the augmented dataset. An example of images from this set is shown below in fig 1.

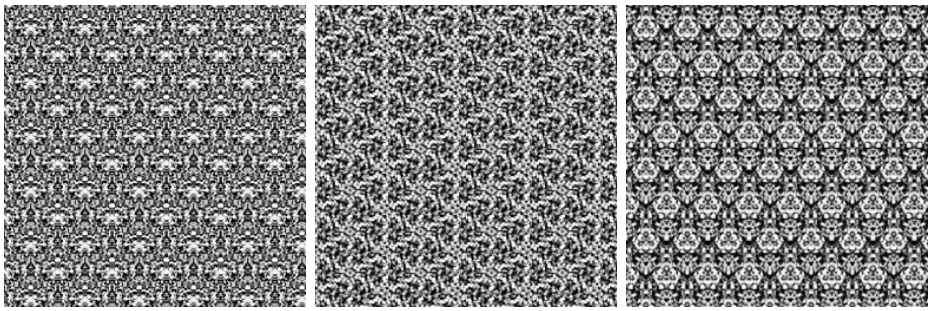


Figure 1: Examples of the raw images of class CM (left), P6 (center), and P3M1 (right) from the wallpaper data set

2.2 Augmented Dataset

From the original wallpaper data, each image had a list of random transformations performed as per the instructions:

1. The original image is scaled randomly by a factor of 100%-200% using the command **imresize**. The histogram of the scaling for training and testing set images is shown in fig 2

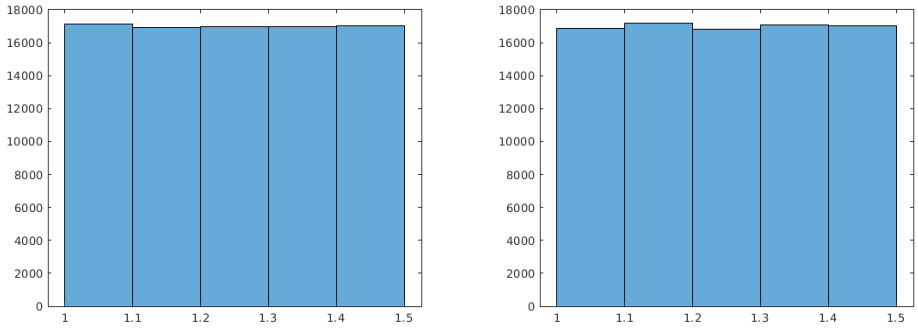


Figure 2: The histogram of frequency of magnification for the training set (left) and testing set (right)

2. The scaled image is rotated randomly by an angle between 0^o and 360^o using the command **imrotate**. The histogram of the frequency of rotation for training and testing set images is shown in fig 3

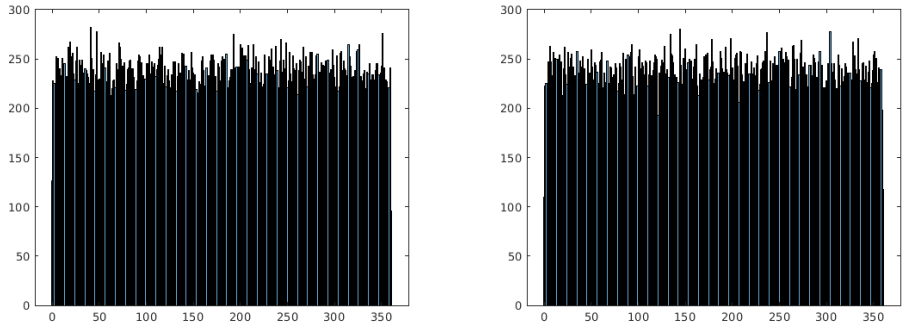


Figure 3: The histogram of frequency of rotation for the training set (left) and testing set (right)

3. The image is shifted in both x and y coordinates randomly by 0-40 pixels using the **imtranslate** function. The histogram of the translation for training and testing set images is shown in fig 4

4. To ensure that all the changes do not lead to empty pixels, the resulting image is cropped to the center 128 * 128 pixels using the **imcrop** function. This ensured that the pattern was present throughout the image.

For each image in the original dataset, 5 new images were generated by using the aforementioned transformations. This gave a final dataset of 85000 images in both training

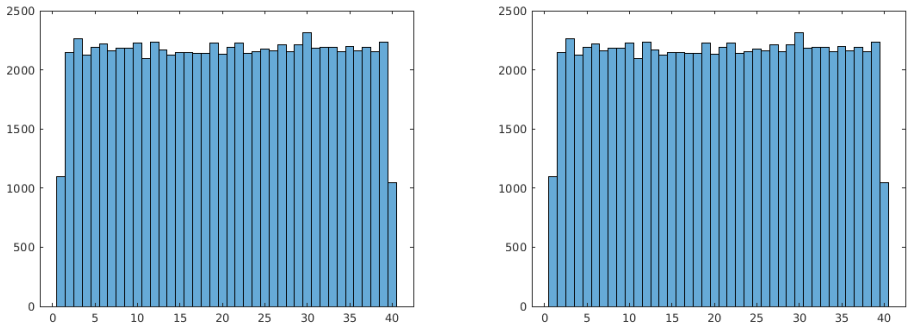


Figure 4: The histogram of frequency of magnification for the training set (left) and testing set (right)

and testing data with an even split into the 17 classes of wallpapers. The example of the transformations can be seen in the fig 5

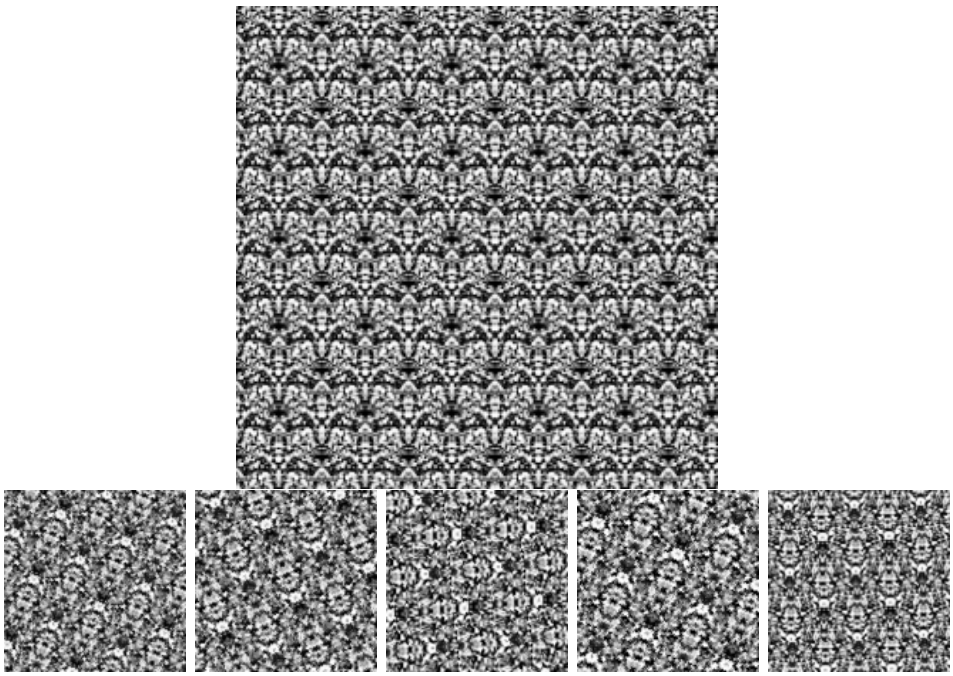


Figure 5: The CM image (above) from the raw data has been augmented into 5 new data points (below) using the transformations mentioned in the augmented dataset

2.3 Alex Transformation

For the transfer learning section of the project, the AlexNet requires an input of 227 * 227 colored image. The augmented dataset is taken for this part and is transformed using the **imresize** and **ind2rgb** functions which give us 85000 227 * 227 color images in both training and testing data which correspond to each image in the augmented dataset. An

example of the transformation can be seen in fig 6

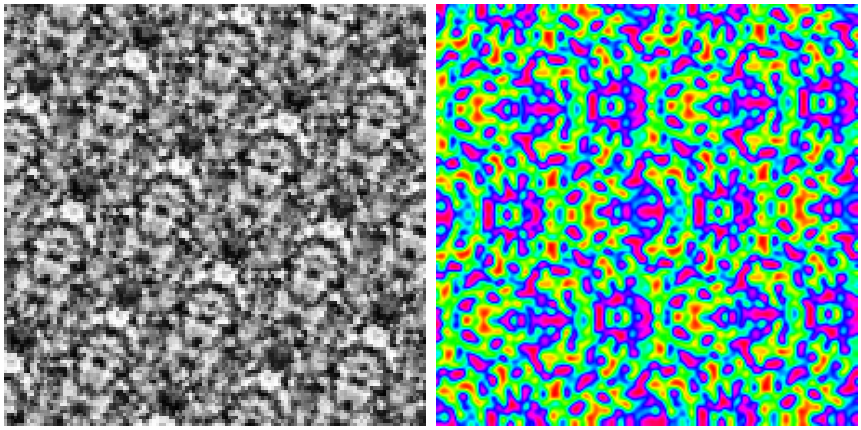


Figure 6: An example of the Alex input transformation for a CM image with the input image from augmented data set (left) and the new modified image (right)

3 Approach

Overall, 4 different Convolutional Neural Networks were used throughout the project. For most of them, the majority of variables were attempted to be kept constant (except certain cases due to limitations of the hardware). The structure and overall description of each of these networks is given below and the results corresponding to each experiment run on them can be found in the next section.

3.1 Default Network

This network was provided with the starting code and was considered the baseline. Experiments were run using batch size of 400 on the ROAR server. The learning rate was kept at 5e-5 after testing and tuning. The model was run for a total of 10 epochs. There are total of 9 layers for the network and a description of each individual layer can be seen in the figure 7

3.2 Skinny Network

This network was built on top of the baseline code and adds many extra layers (a total of 20) to the code while keeping the size of each layer relatively the same. Experiments were run using batch size of 100 on the ROAR server. The model was run for a total of 20 epochs. The learning rate was kept at 1e-4 for the first 10 epochs and was increased to 5e-4 for the next 10 to improve rate of convergence. This was the slowest network to run out of all the networks. The layers for the network can be seen in the figure 8

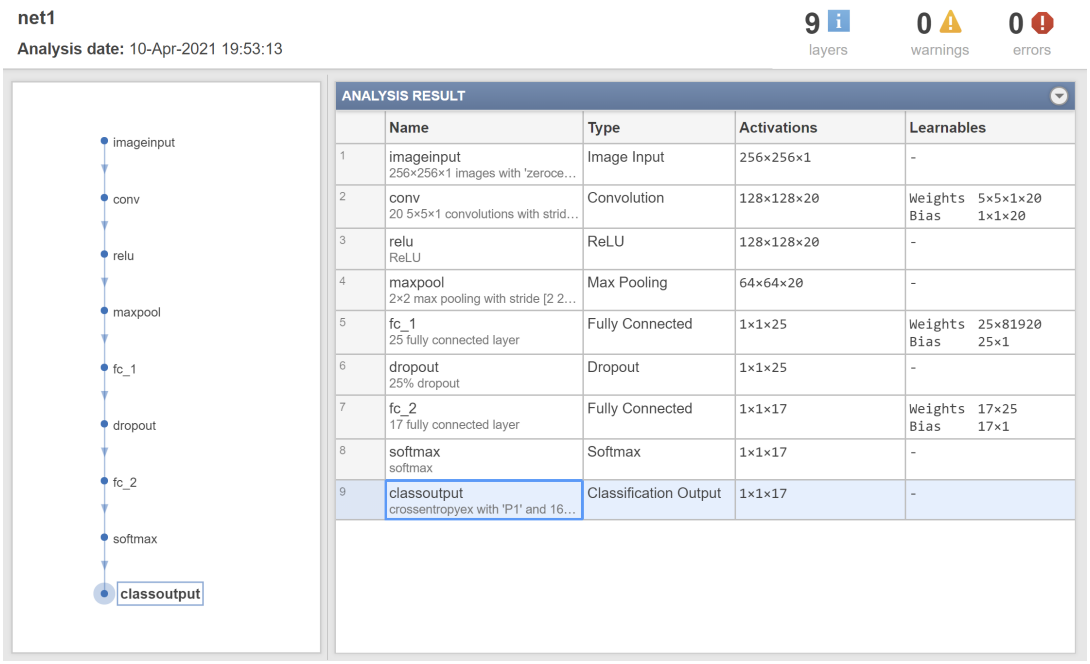


Figure 7: A diagram of all the layers of the default network

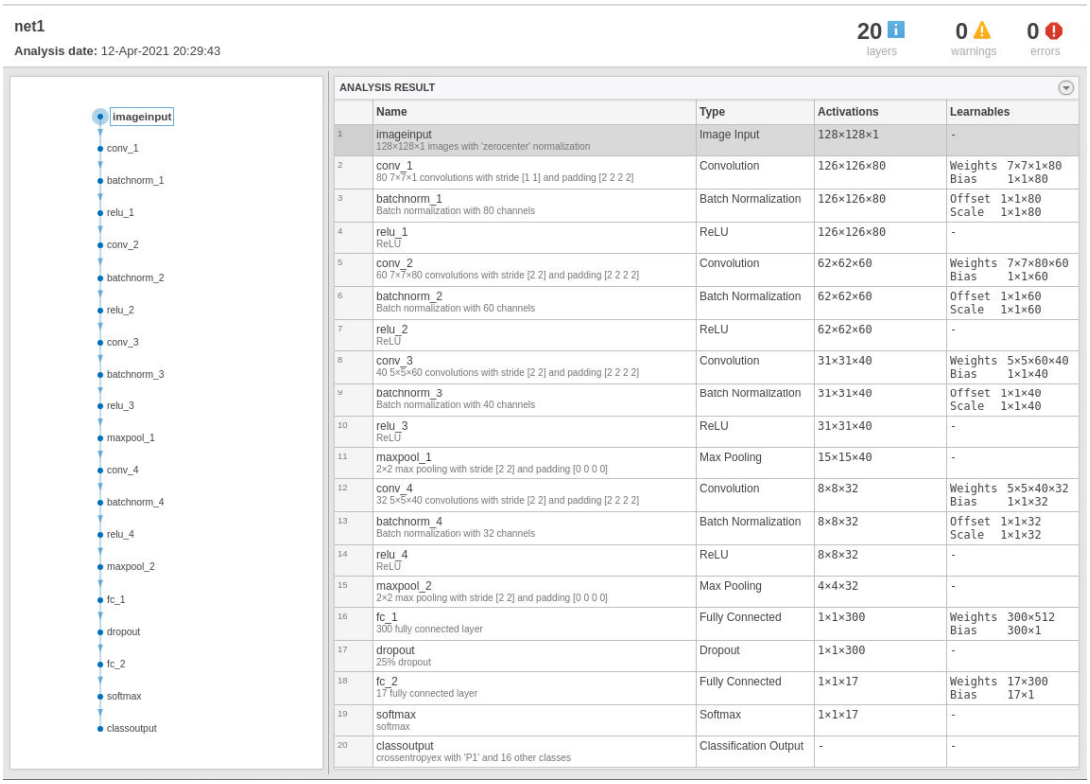
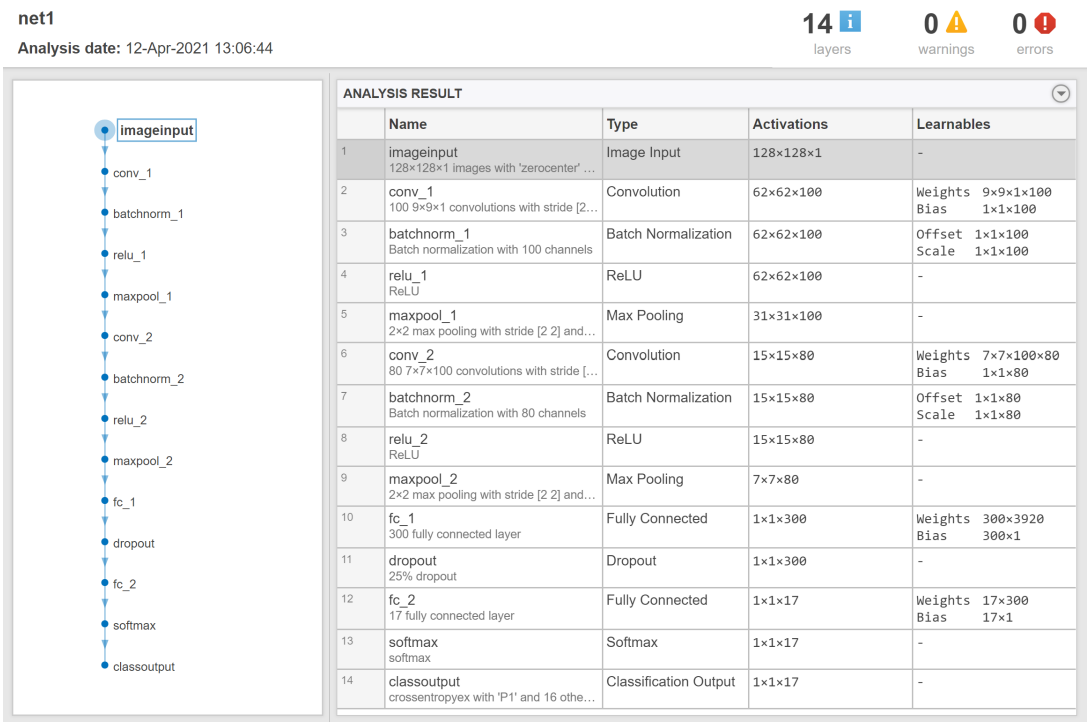


Figure 8: A diagram of all the layers of the skinny network

3.3 Wide Network

This network was implemented on top of the baseline code and increases the size of each layer substantially compared to the original code while keeping to the minimum number of layers expected from the code at 9. Experiments were done on the local machine with a batch size of 25. The layers for the network can be seen in the figure 9



	Name	Type	Activations	Learnables
1	imageinput 128×128×1 images with 'zerocenter' ...	Image Input	128×128×1	-
2	conv_1 100 9×9×1 convolutions with stride [2...	Convolution	62×62×100	Weights 9×9×1×100 Bias 1×1×100
3	batchnorm_1 Batch normalization with 100 channels	Batch Normalization	62×62×100	Offset 1×1×100 Scale 1×1×100
4	relu_1 ReLU	ReLU	62×62×100	-
5	maxpool_1 2×2 max pooling with stride [2 2] and...	Max Pooling	31×31×100	-
6	conv_2 80 7×7×100 convolutions with stride [...	Convolution	15×15×80	Weights 7×7×100×80 Bias 1×1×80
7	batchnorm_2 Batch normalization with 80 channels	Batch Normalization	15×15×80	Offset 1×1×80 Scale 1×1×80
8	relu_2 ReLU	ReLU	15×15×80	-
9	maxpool_2 2×2 max pooling with stride [2 2] and...	Max Pooling	7×7×80	-
10	fc_1 300 fully connected layer	Fully Connected	1×1×300	Weights 300×3920 Bias 300×1
11	dropout 25% dropout	Dropout	1×1×300	-
12	fc_2 17 fully connected layer	Fully Connected	1×1×17	Weights 17×300 Bias 17×1
13	softmax softmax	Softmax	1×1×17	-
14	classoutput crossentropyex with 'P1' and 16 othe...	Classification Output	1×1×17	-

Figure 9: A diagram of all the layers of the Wide network

3.4 Alex Network

This network was taken directly from the pre trained AlexNet network. The input was modified to be compatible with the input layers of the original network and the last 3 layers were changed to get the output for the Wallpaper dataset problem. Due to inability to run or install AlexNet to the server machine, this was run on the local machine with a 4GB VRAM graphic card. Due to the limitation of the machine, the batch size was kept small at 25 as a higher number lead to GPU memory shortage. This resulted in a high number of passes and so, the learning rate was kept at 1e-5 and number of epochs was kept at 10. There are total of 25 layers for the network and a description of each individual layer can be seen in the figure 10

4 Results

Multiple experiments were run on the aforementioned Networks and the results and corresponding diagrams are shown below:

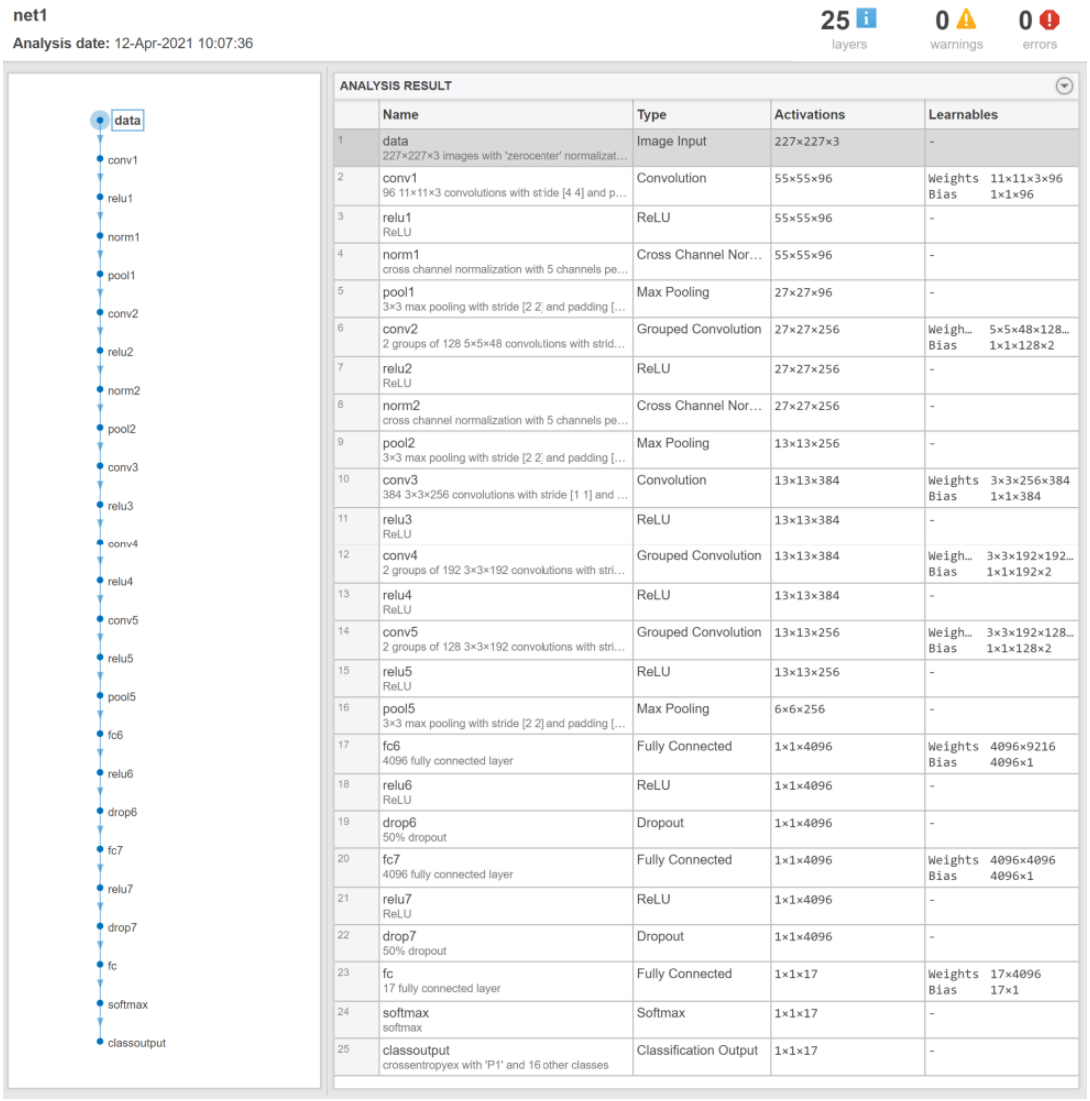


Figure 10: A diagram of all the layers of the Alex Transfer network

4.1 Default Network - Raw Data

This run had 10 epochs with batch size 400 and learning rate of 5e-5. It took 454.8 seconds on the server allocated machine getting testing accuracy of **0.7041**, validation set accuracy **0.7106**, and training set accuracy of **0.7722**. The accuracies show a mild over-fitting but the overall results match the expectations from the network. The confusion matrices (11), accuracy and loss graph over the runs (12), first layer activation visualization, and T-SNE visualization (13) are shown in the corresponding figures.

4.2 Default Network - Augmented Data

This run had 10 epochs with batch size 400 and learning rate of 5e-5. It took 675.68 seconds on the server allocated machine getting testing accuracy of **0.1104**, validation set accuracy **0.1109**, and training set accuracy of **0.1193**. Given that the augmented dataset

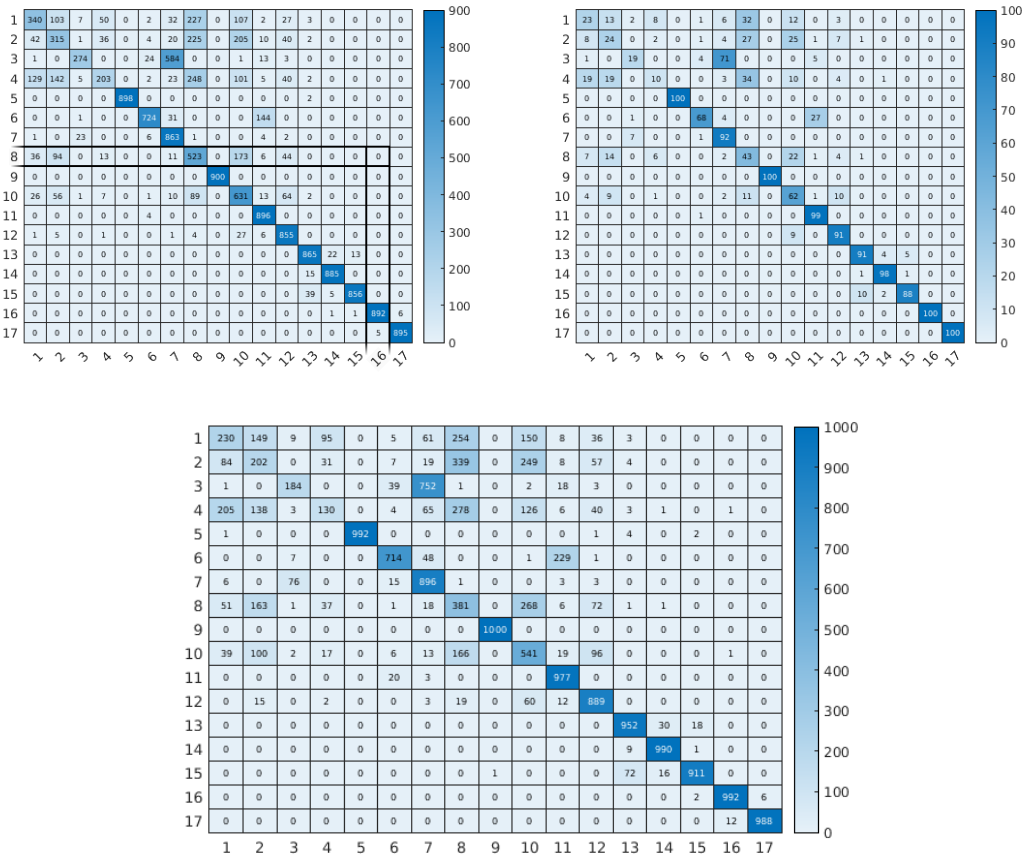


Figure 11: The confusion matrices of classification by the default Network on the original data with the training data (left), validation set data(right), and the test set data (bottom)

is a lot tougher than the original dataset, the result is expected. The confusion matrices (14), accuracy and loss graph over the runs (15), first layer activation visualization, and T-SNE visualization (16) are shown in the corresponding figures.

4.3 Skinny Network - Augmented Data

The network ran with a batch size of 100 on the server for 15 epochs. The learning rate for the first 10 epochs was 1e-4 and the last 5 was 5e-4. This entire process took 31832 seconds or approximately 9 hours and the network still did not seem like completely converging. This was the slowest of all the networks and was probably the reason why more optimization could not be done for it. The final testing accuracy was **0.583**, the training set accuracy was **0.6519** and validation set accuracy of **0.6015** was achieved. The confusion matrices (17), accuracy and loss graph over the runs (18), first layer activation visualization, and T-SNE visualization (19) are shown in the corresponding figures.

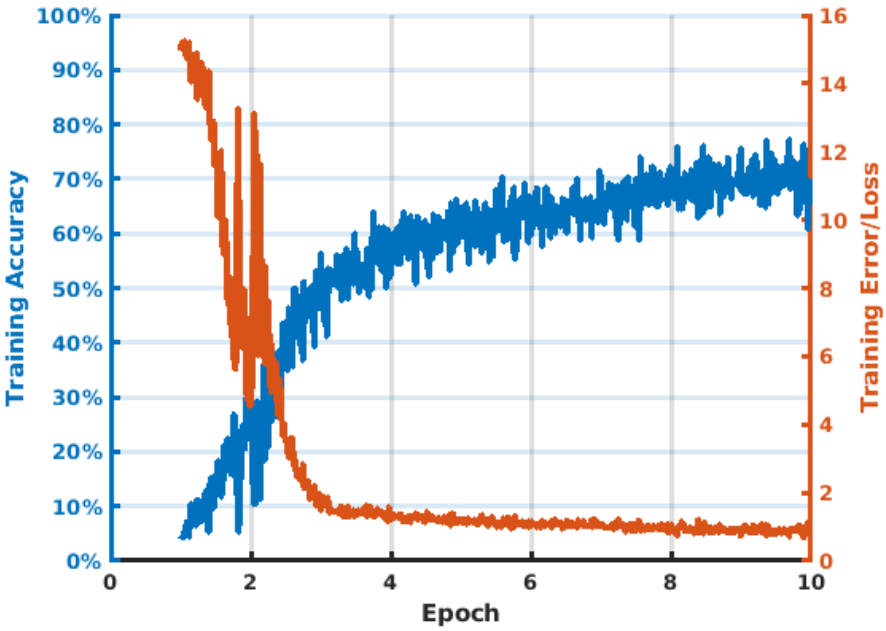


Figure 12: The accuracy and loss graph over the passes for the original data using the default network

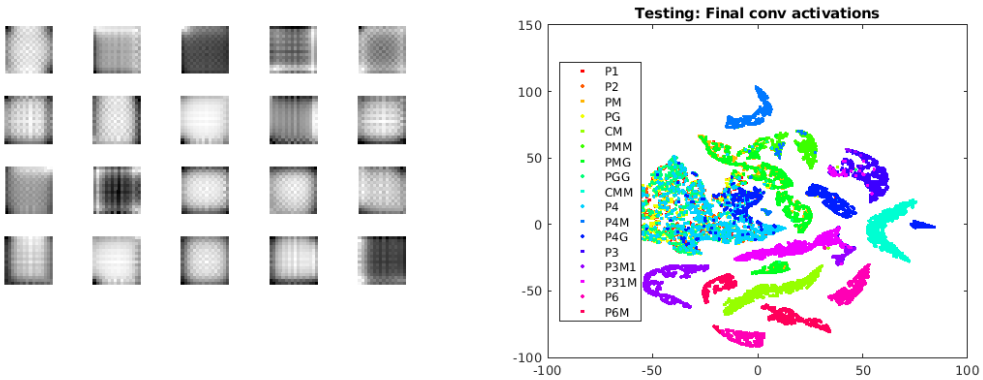


Figure 13: The first layer activation (left) and T-SNE visualization of the final layer (right) for the original data using the default network

4.4 Skinny Network - Raw Data

The network was kept the same as the previous experiment (batch size = 100, 15 epochs, learning rate for the first 10 epochs was 1e-4 and the last 5 was 5e-4). The process took 10586 seconds approximately 3.5 hours and the network converged completely. The final resulting classification had a training accuracy of **1.00**, testing accuracy of **0.9241**, and validation set accuracy of **0.9559**. The confusion matrices (20), accuracy and loss graph over the runs (21), first layer activation visualization, and T-SNE visualization (22) are shown in the corresponding figures.

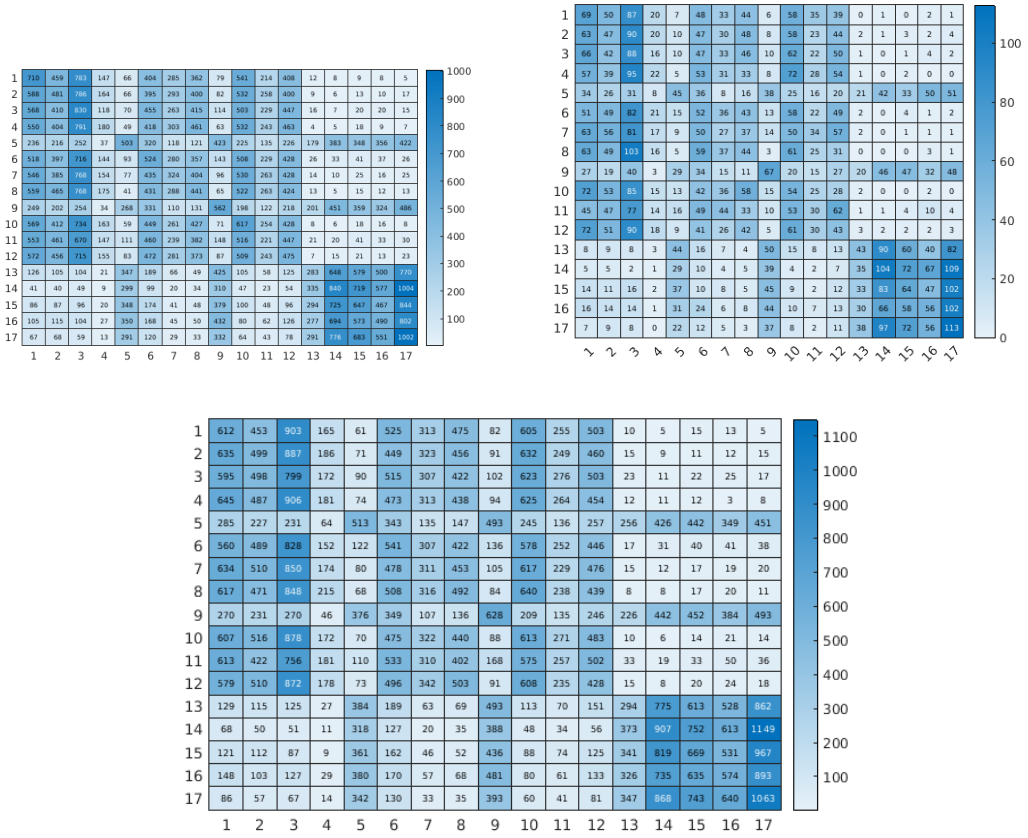


Figure 14: The confusion matrices of classification by the default Network on the augmented data with the training data (left), validation set data(right), and the test set data (bottom)

4.5 Wide Network - Augmented Data

The Network was run with a batch size of 25 on the local machine for 20 epochs. The learning rate for the first 10 epochs was 5e-6 and the next 10 epochs was 5e-5. The rate was kept low to compensate the low batch size. The code took 7374 seconds to finish execution. The testing accuracy achieved was **0.5045**, the training accuracy was **0.6630**, and the validation accuracy was **0.534**. There was a considerable amount of variance in the error and accuracy due to the small size of the batch and the increased size of the layers. The confusion matrices (23), accuracy and loss graph over the runs (24), first layer activation visualization, and T-SNE visualization (25) are shown in the corresponding figures.

4.6 Wide Network - Raw Data

The same experiment was repeated with the same parameters (batch size = 25, 20 epochs, learning rate 5e-6 for first 10, 5e-5 for next 10). The network took significantly less time 2294 seconds compared to the augmented data run and ended up reaching 100% training set accuracy by epoch 15. The final resulting classification had a training accuracy

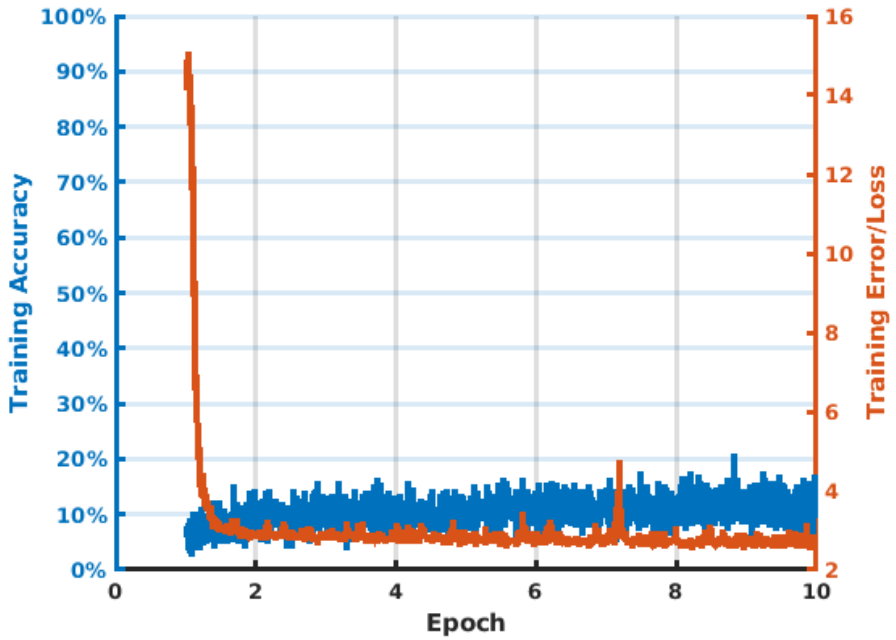


Figure 15: The accuracy and loss graph over the passes for the augmented data using the default network

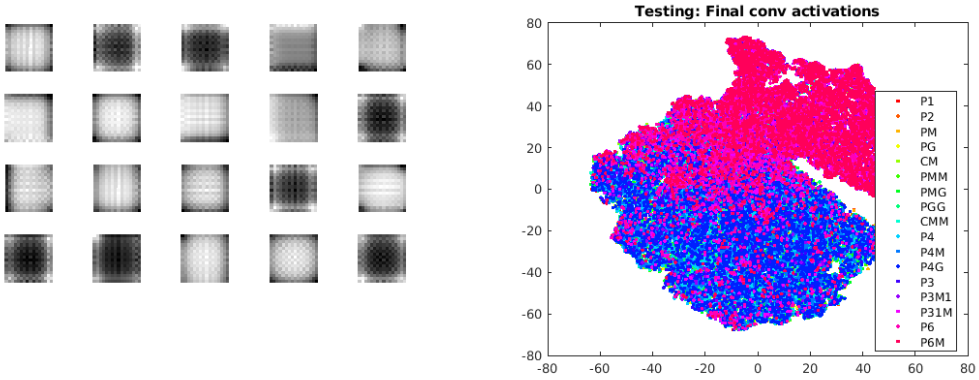


Figure 16: The first layer activation (left) and T-SNE visualization of the final layer (right) for the augmented data using the default network

of **1.00**, testing accuracy of **0.9401**, and validation set accuracy of **0.9359**. The improvement in result was expected given the significantly simpler dataset.The confusion matrices, accuracy and loss graph over the runs, first layer activation visualization, and T-SNE visualization are shown in fig INSERT¿

4.7 Alex Network - Alex Data

Due to inability to run Alex net on the server machine, this was conducted on the local machine with a batch size of 50, alpha value of 5e-5 for 10 epochs. As mentioned

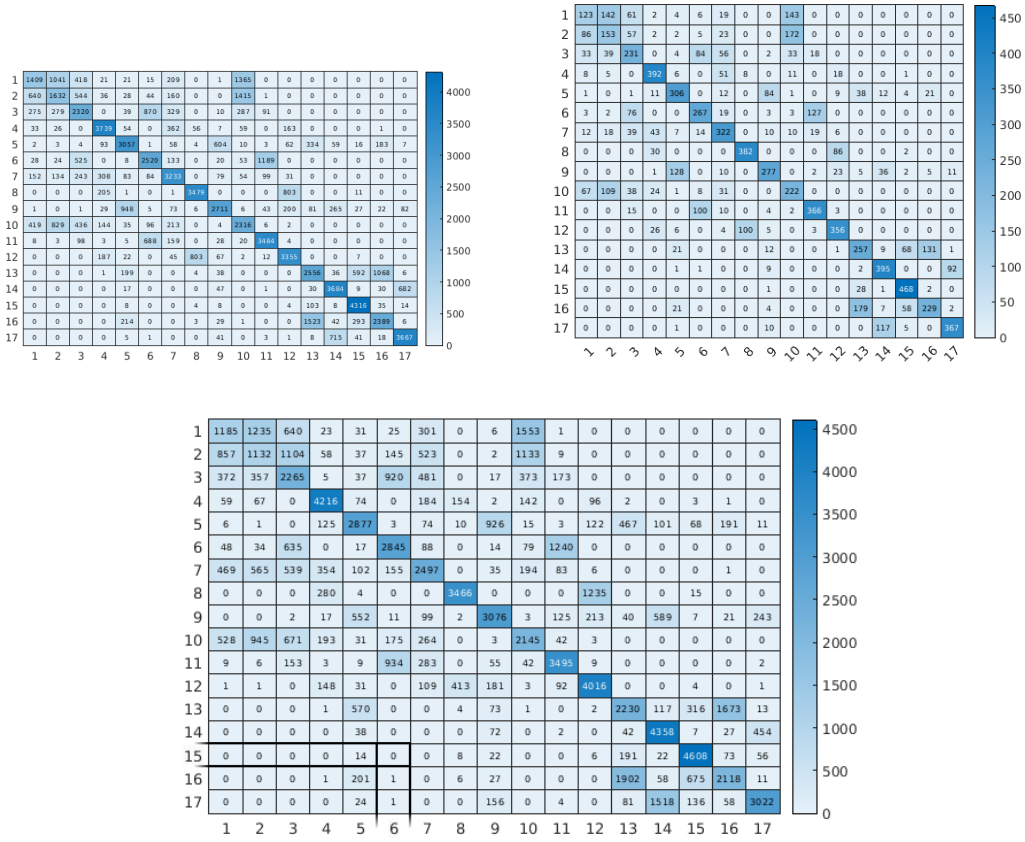


Figure 17: The confusion matrices of classification by the Skinny Network on the augmented data with the training data (left), validation set data(right), and the test set data (bottom)

previously, the net needed a modified input. The total run time of this was 7369.96 seconds. Given the pre trained network, this method had the best results for the data, getting a really high testing accuracy of **0.8281**, validation accuracy of **0.8291**, and training accuracy of **0.9358**. The long tun time can be attributed to the high number of passes alongside the large number of layers in the network. The confusion matrices (29), accuracy and loss graph over the runs (30), first layer activation visualization, and T-SNE visualization (31) are shown in corresponding figures.

5 Discussion

There are several take aways from the project. The primary thing was creating my own Neural Network and the hands on approach really helped with understanding different aspects of Neural Networks such as parameters, layers, and their implications on the final model alongside the use of various tools that help with the long process such as check points and general tips.

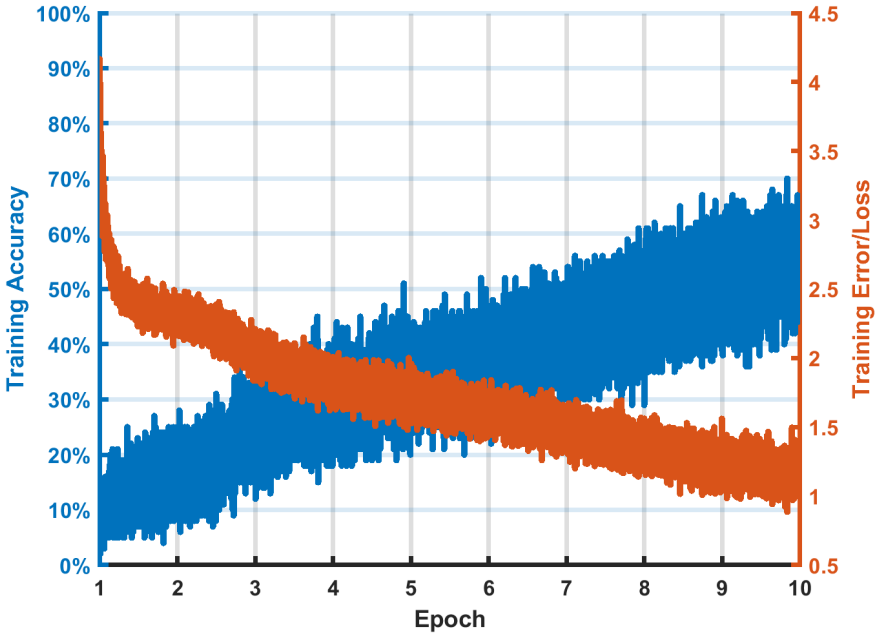


Figure 18: The accuracy and loss graph over the passes for the augmented data using the Skinny network

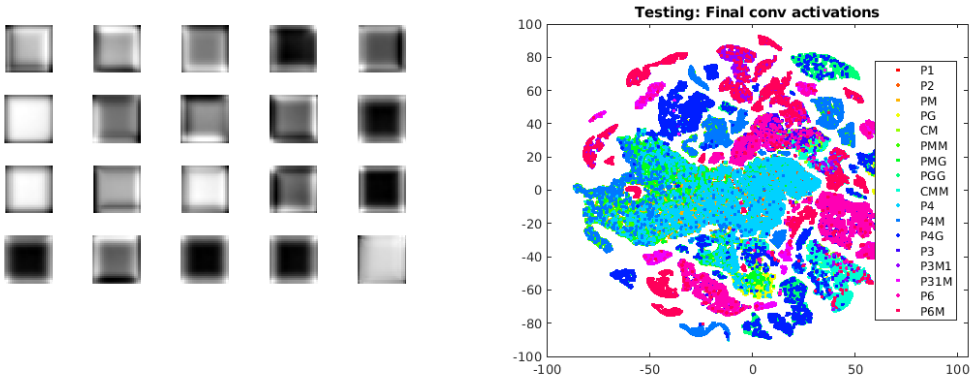


Figure 19: The first layer activation (left) and T-SNE visualization of the final layer (right) for the augmented data using the Skinny network

5.1 Relation between learning rate and batch size

A key thing noted was the relation between the learning rate and batch size. Increasing the batch size increased the speed of the algorithm, but also reduced the accuracy due to less number of passes. This in turn needs the learning rate to be increased to keep up with the model otherwise the rate converges prematurely. This was remedied several times by employing variable learning rate which is low for initial epochs but is then increased to a higher value to increase the speed of convergence as shown in the skinny network and wide network.

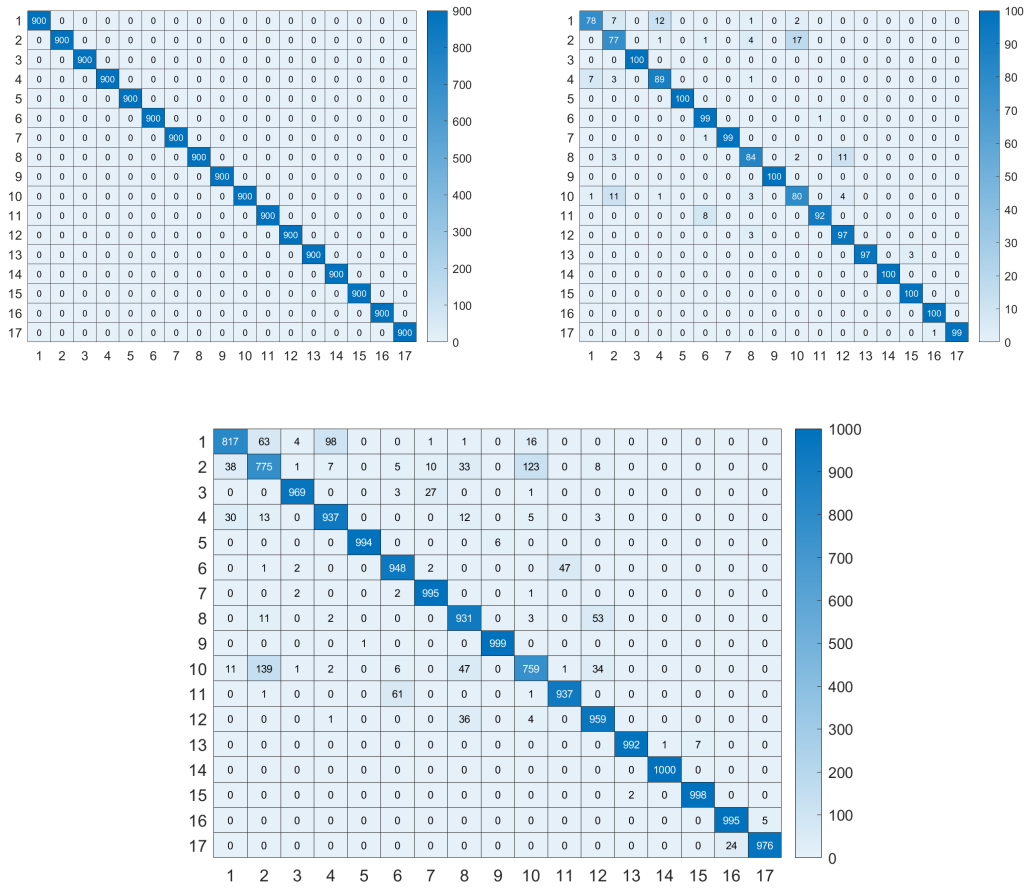


Figure 20: The confusion matrices of classification by the Skinny Network on the raw data with the training data (left), validation set data(right), and the test set data (bottom)

5.2 Difference between Skinny and Wide Networks

Difference between skinny and wide networks was also very apparent. While both can be used to improve the effectiveness of a given Neural networks, both have their own set of restrictions.

Skinny networks considerably increase the run time and size of the model. They were the slowest out of all the models. However, they are very good at learning the information, and do not converge easily. Given enough iterations, they do a good job of learning the model accurately.

Wide networks are a lot faster than skinny networks. However, they are very sensitive to the learning rate given the amount of information each layer contains and converge very quickly if not checked. They also have a lower ceiling than skinny networks. A good model such as AlexNet utilizes both wide layers and more number of layers to get the best out of both these ways to improve your network.

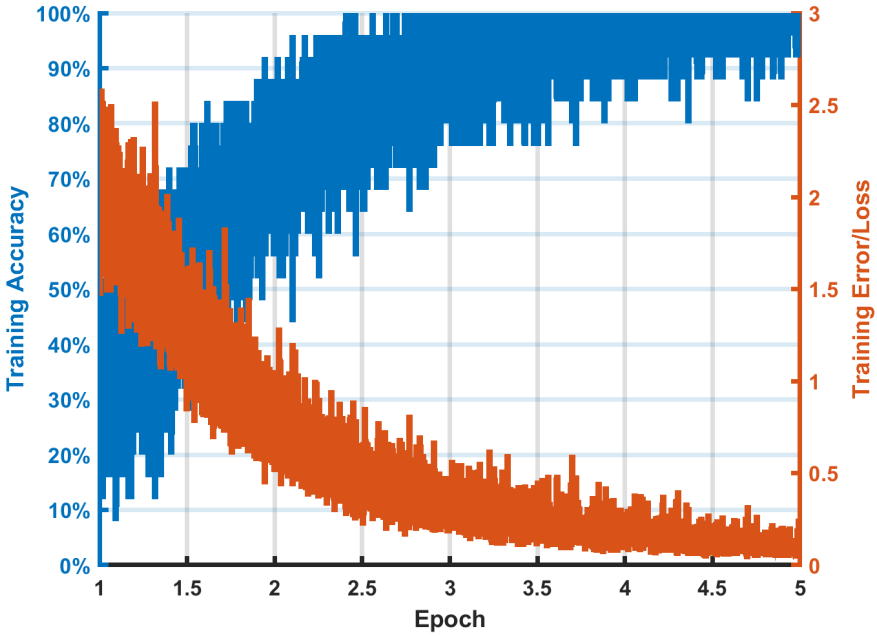


Figure 21: The accuracy and loss graph over the passes for the raw data using the Skinny network

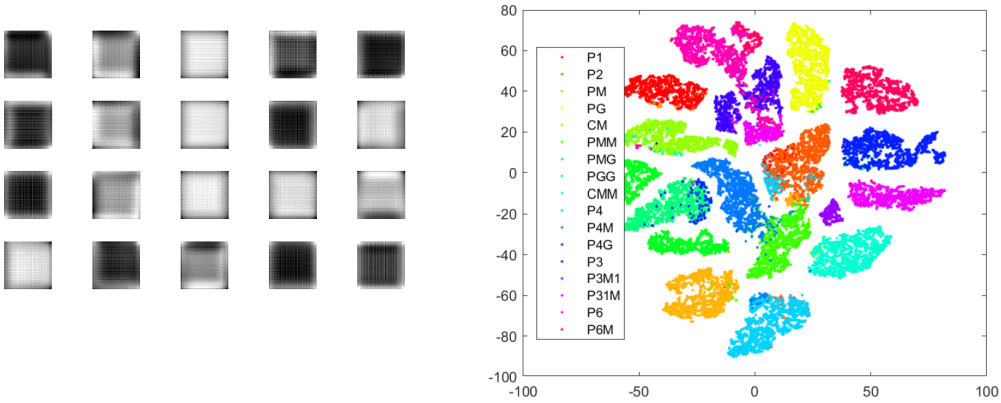


Figure 22: The first layer activation (left) and T-SNE visualization of the final layer (right) for the raw data using the Skinny network

5.3 Accuracy and Loss Function

Another key thing to note, specifically from the default network on the augmented dataset was that the relationship between loss and accuracy is not strictly proportional. In that case and in many other intermediary networks, the loss reduced but the accuracy did not increase over many epochs. This needs to be carefully observed for any network so that the final result converges to a high accuracy.

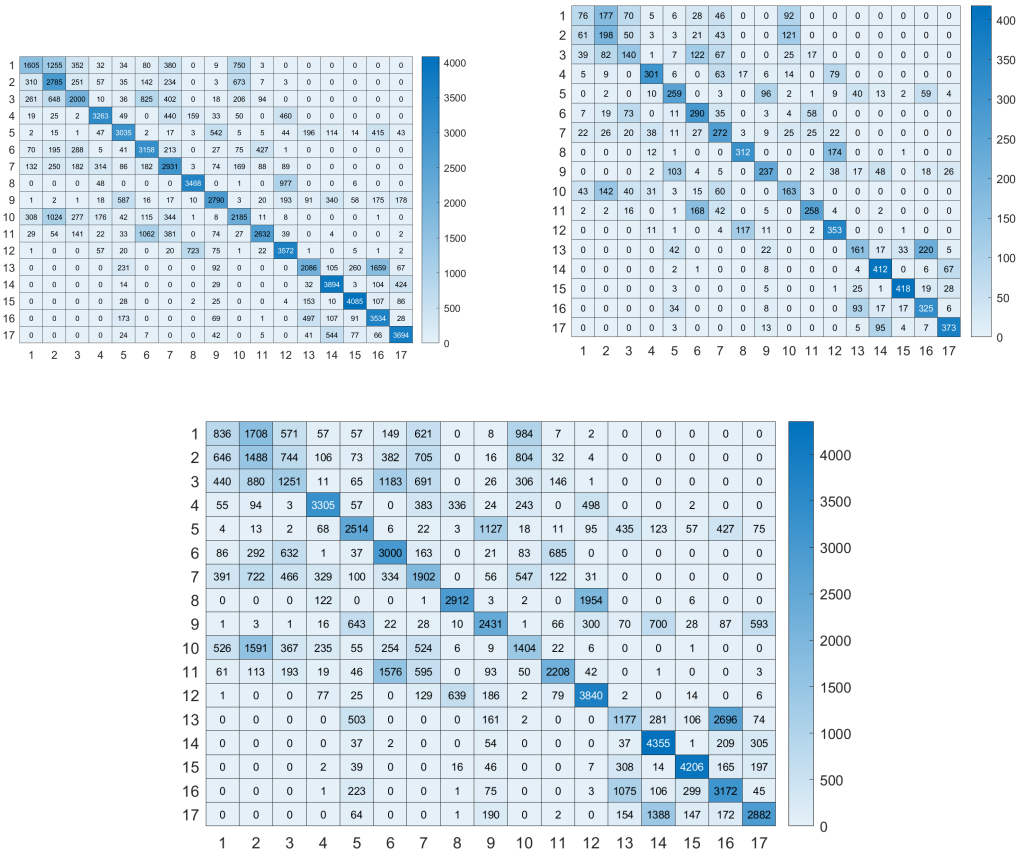


Figure 23: The confusion matrices of classification by the Wide Network on the augmented data with the training data (left), validation set data(right), and the test set data (bottom)

5.4 Data modification

In addition, it was interesting to see how a simple data set can be augmented to increase the difficulty of the problem and similarly, a dataset can be modified to facilitate better classification by making more information available than the original data. This was very apparent with the Alexnet as it got incredibly good results in less time and part of it was the ability to utilize the extra dimension of color for the image data.

The final thing to take out of the project was the effectiveness of transfer learning. Instead of making a new Neural network for each problem, a pre trained model can be modified to fit into the pipeline to get a much better performance in terms of accuracy as well as time. This could help in the final class project as well.

6 Conclusion

Neural Networks are a very powerful tools which can perform many very difficult tasks. While a simple model is sufficient for a small task, more difficult tasks require much more

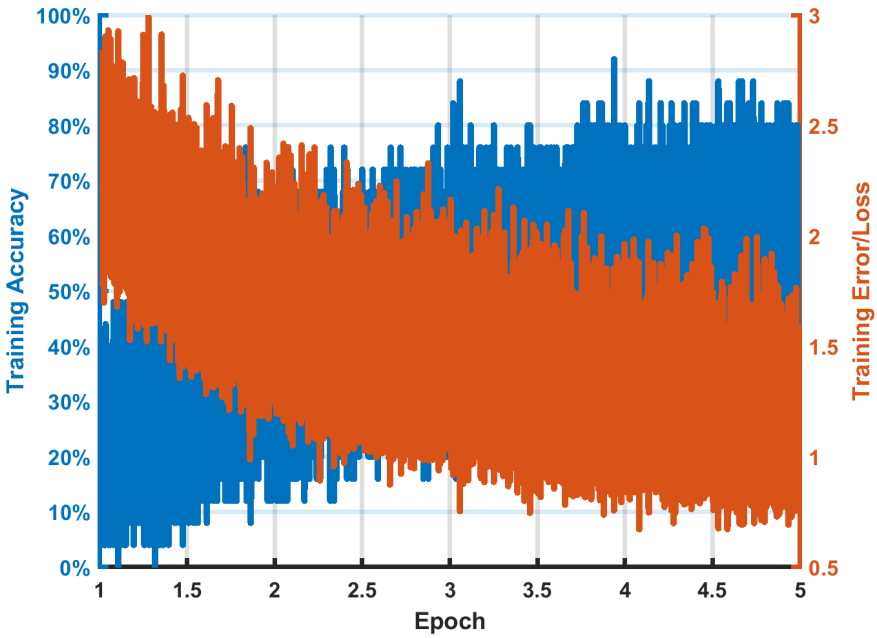


Figure 24: The accuracy and loss graph over the passes for the augmented data using the Wide network

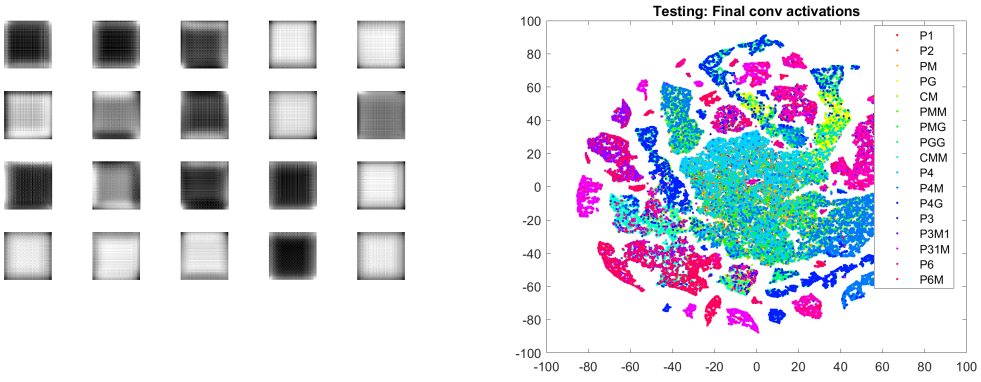


Figure 25: The first layer activation (left) and T-SNE visualization of the final layer (right) for the augmented data using the Wide network

complicated networks which are both resource intensive and time consuming. Understanding the basic fundamentals of Neural Networks is very important before starting to build them as that can save a lot of time during the development cycle and help with understanding the right tuning of different parameters. The use of trial-and-error method in the project has given vital information on how to build better neural networks and further work would certainly make the world of networks more intuitive.

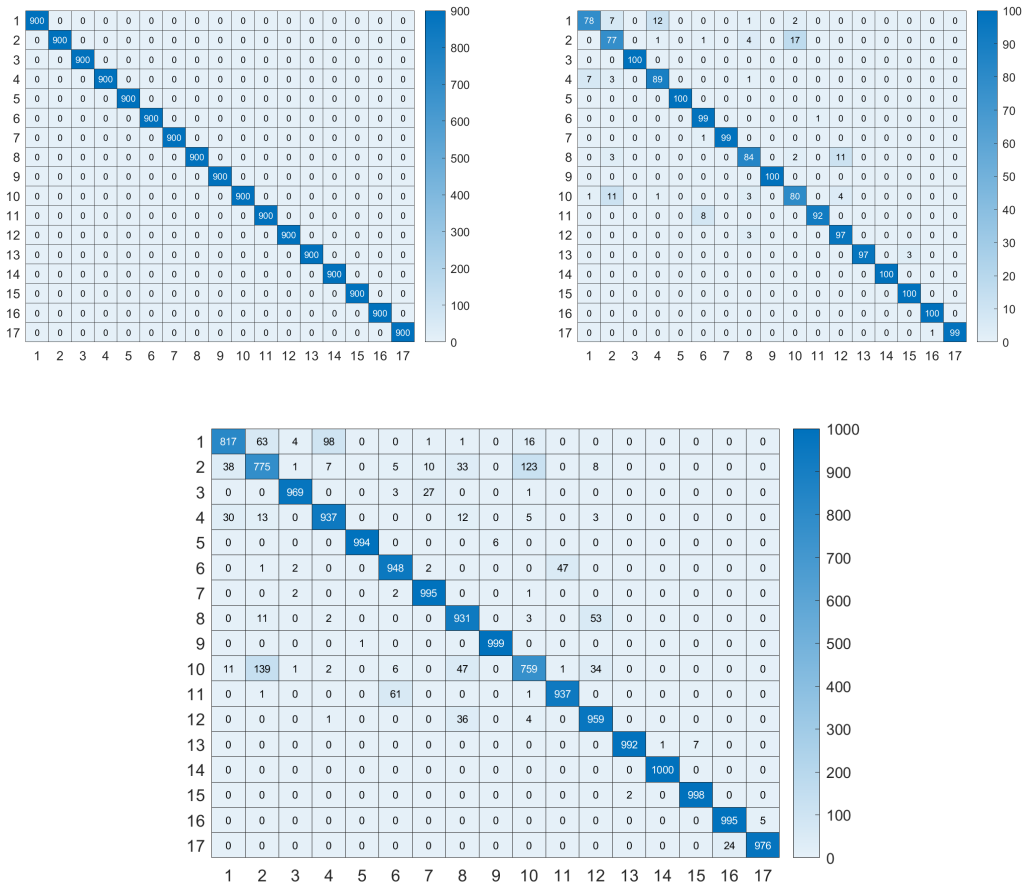


Figure 26: The confusion matrices of classification by the default wide on the raw data with the training data (left), validation set data(right), and the test set data (bottom)

References

[1] M. Works, "What is deep learning?" [Online]. Available: https://www.mathworks.com/discovery/deep-learning.html 3

[2] "Mathworks - list of deep learning layers." [Online]. Available: https://www.mathworks.com/help/deeplearning/ug/list-of-deep-learning-layers.html 3

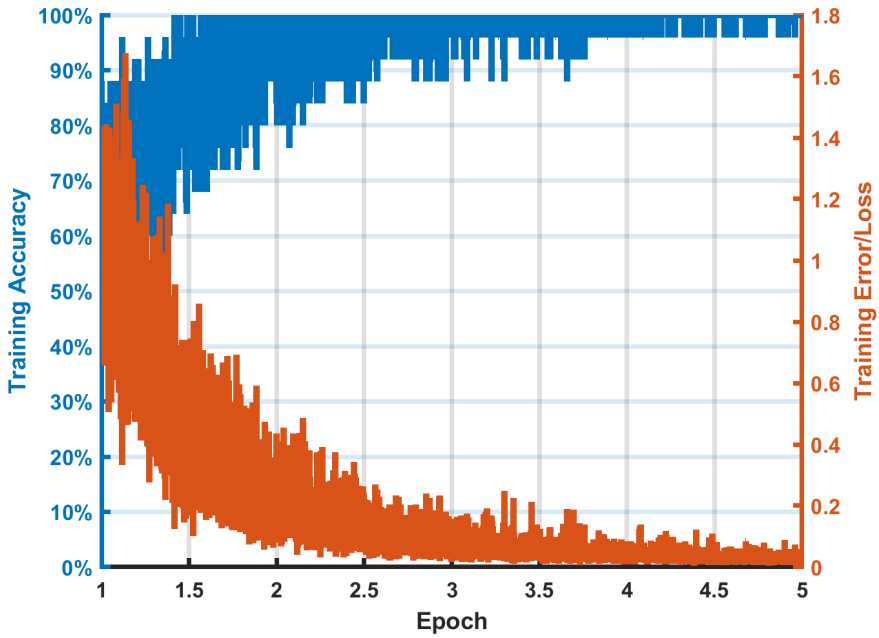


Figure 27: The accuracy and loss graph over the passes for the raw data using the wide network

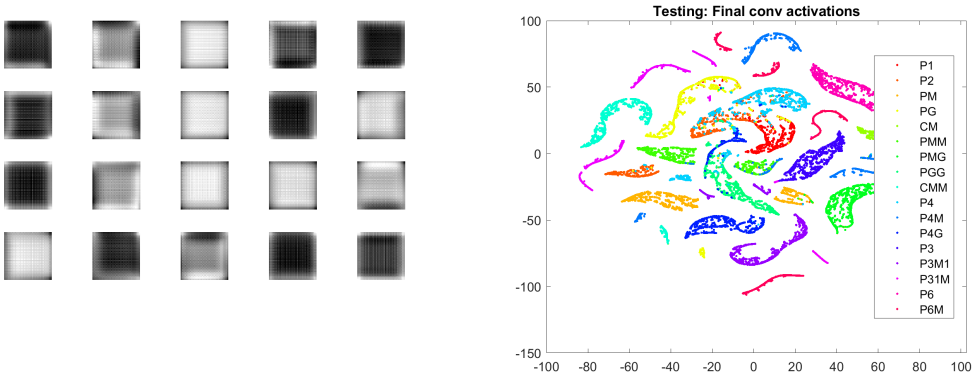


Figure 28: The first layer activation (left) and T-SNE visualization of the final layer (right) for the raw data using the wide network

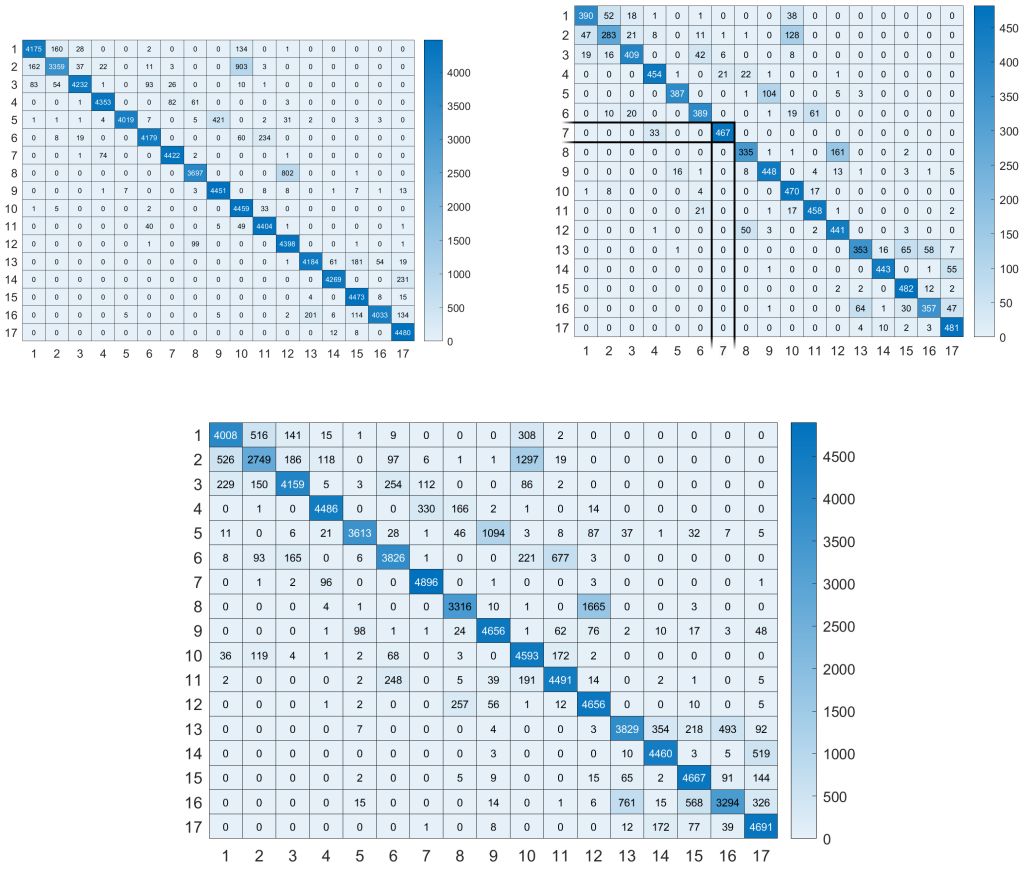


Figure 29: The confusion matrices of classification by the Alexnet Network on the modified augmented data with the training data (left), validation set data(right), and the test set data (bottom)

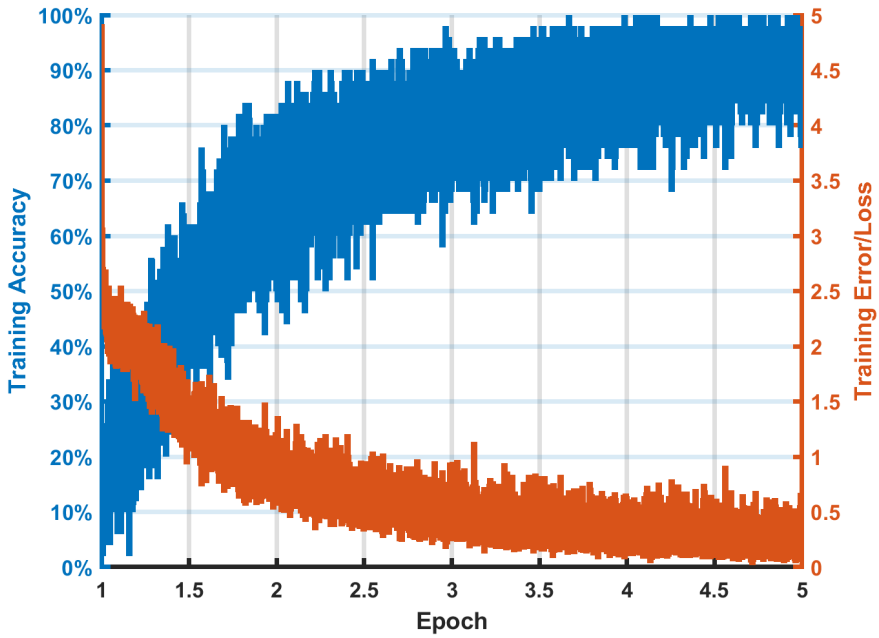


Figure 30: The accuracy and loss graph over the passes for the augmented data using the AlexNet transfer learning network

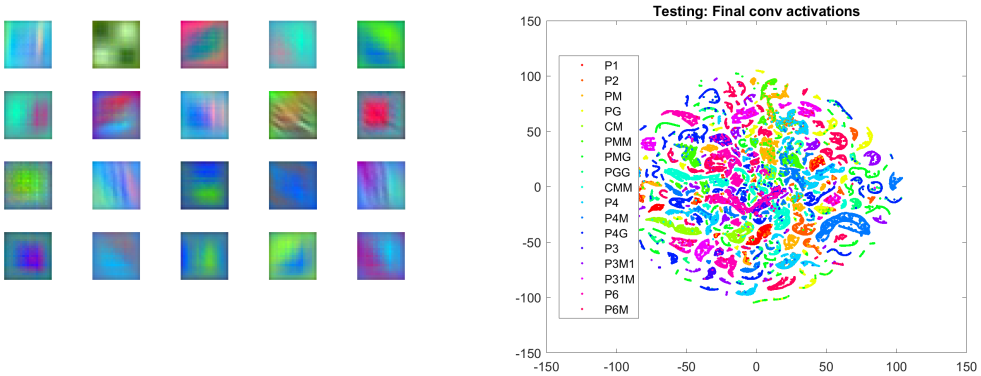


Figure 31: The first layer activation (left) and T-SNE visualization of the final layer (right) for the augmented data using the AlexNet network