RBE/CS549 Computer Vision

Homework 0 - Alohomora

Anagha R. Dangle
Email: ardangle@wpi.edu

***Abstract*—The assignment is divided into two phases:**
Phase 1: Shake my Boundary- In this phase of assignment, Pb-Lite boundary detection algorithm is to be implemented. The algorithm detects boundaries with per-pixel probability, by combining the local and global image information i.e. Texture, Color and Brightness.
Phase 2: Deep dive into Deep learning- In this phase of assignment, different neural network architectures are been implemented on CIFAR dataset and the results are analysed.

***Index Terms*—Gaussian filter, Sobel and Canny edge detection, BSDS500, CIFAR10, ResNet.**

I. Phase 1: Shake my Boundary

In this assignment, a simplified version of Pb probability of boundary was to be implemented. This algorithm finds boundaries by examining brightness, color, and texture information in multiple scales.The implementation of this algorithm was intended to outperform the standard Sobel and Canny filter edge detection algorithms. The algorithm has five main basic steps:
1. Filters- Three different filters: Oriented Derivative of gaussian Filter, Leung-Malik Filter and Gabor filter are implemented.
2. Create Half-Disc masks
3. Generate Texton Map and Texton Gradient.
4. Generate Brightness Map and Brightness gradient.
5. Generate Color Map and Color Gradient.
6. Combine information from the features with a Sobel and Canny methods (Average).
Each step of the algorithm is detailed in following section.

A. Filters

Three filter banks are used in this assignment, namely, DoG filterbank, Leung-Malik filter bank, and Gabor filter. These filter capture the information from the image, later used for edge detection. The filterbanks explained further in detail:

1) Oriented Derivative of Gaussian filter: The first order derivative is computed by convolving the Gaussian with Sobel in respect to horizontal and vertical axes. The resultant derivative of Gaussian is then oriented by given number of orientations. This oriented Derivatives of Gaussian are appended to form the DoG filterbank.

For the code implementation, 2 scales and 16 orientations were used resulting in 2×16 filters.

The Illustration of OrientedDoG Filterbank is given by Fig(1).

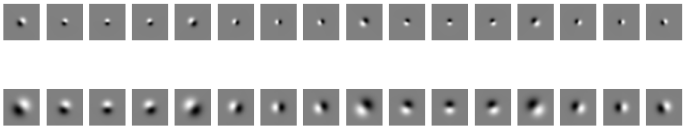


Fig. 1: OrientedDoG Filterbank

2) Leung-Malik Filter: The implemented Leung-malik filterbank has 48 filters in total. Two versions of Filterbanks are implemented for the scope of the assignment: LM Large and LM Small. The scales $\sigma = [1, 2\text{–}, 2, 22\text{–}]$ are used for LM Small and $\sigma = [2\text{–}, 2, 22\text{–}, 4]$ for LM Large.

The basic flow for generating a LM filter is to start with generating a gaussian filter. Then the computation of Laplacian of Gaussian is done. While calculating the LoG, The Gaussian Filter is convolved with the Laplacian Negative matrix to get the LoG filter matrix. The other Derivative of Gaussian and Second derivative of Gaussian are added to the filterbank in the next step.

The illustration of both the filters is given by Fig(2) and Fig(3).

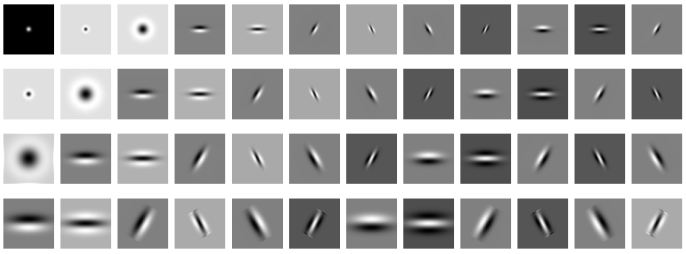


Fig. 2: Leung-Malik Large Filterbank

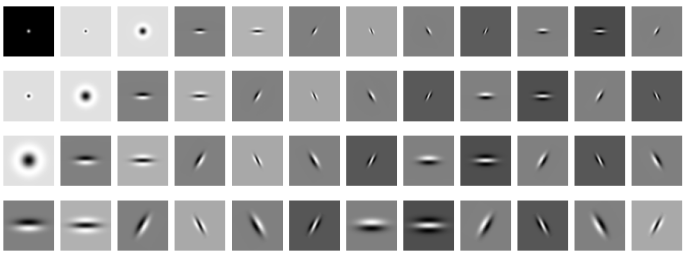


Fig. 3: Leung-Malik Small Filterbank

3) Gabor filter: A 2-D Gabor filter is a Gaussian kernel function modulated by a sinusoidal plane wave. A Gabor filter

can be used to automatically extract features using a filter bank.

The Gabor filter used in this assignment has 5 scales, [3,5,7,9,11], with 8 orientations of theta and lambda value of 5.

The Gabor filterbank representation can be seen in Fig(4).

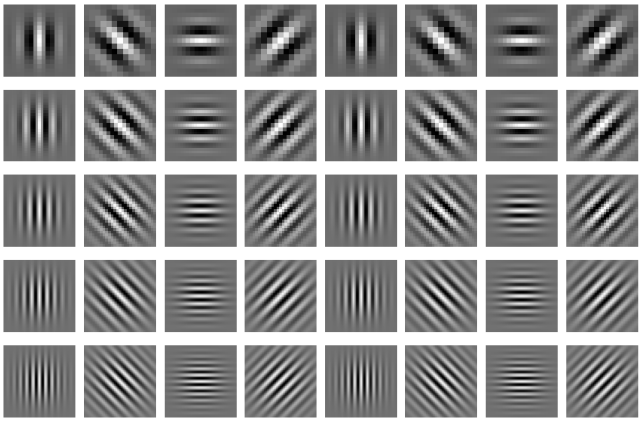


Fig. 4: Gabor Filterbank

B. Half discs

Half discs are just binary images of half discs with different orientation. These discs are used to calculate the gradient i.e. Chi-Square distance for the further filtering operations. For this implementation, 8 orientations and 3 scales, [5, 7, 10] are considered.

The Fig(5) shows the collection of half-discs.

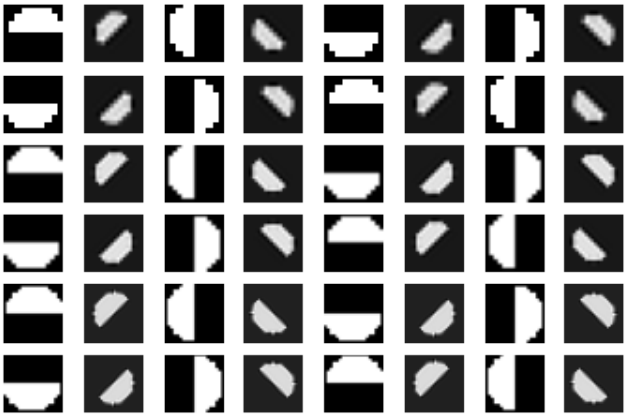


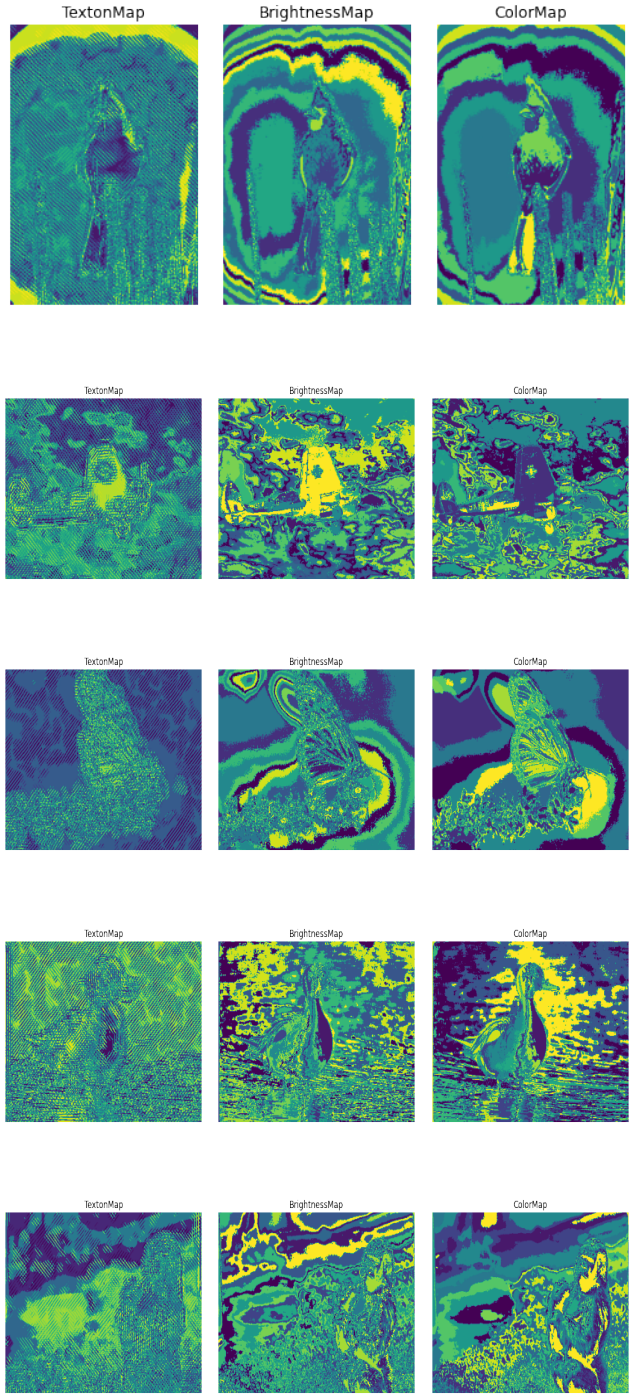
Fig. 5: Half Discs

C. Texton, brightness and color map

Each pixel recieves N filter responses from N filters in the filter bank. By substituting a discrete texton ID for each N-dimensional vector, we can simplify this representation. We achieve this by using kmeans to cluster the filter responses at all image pixels into K textons. This generates the Texton map.

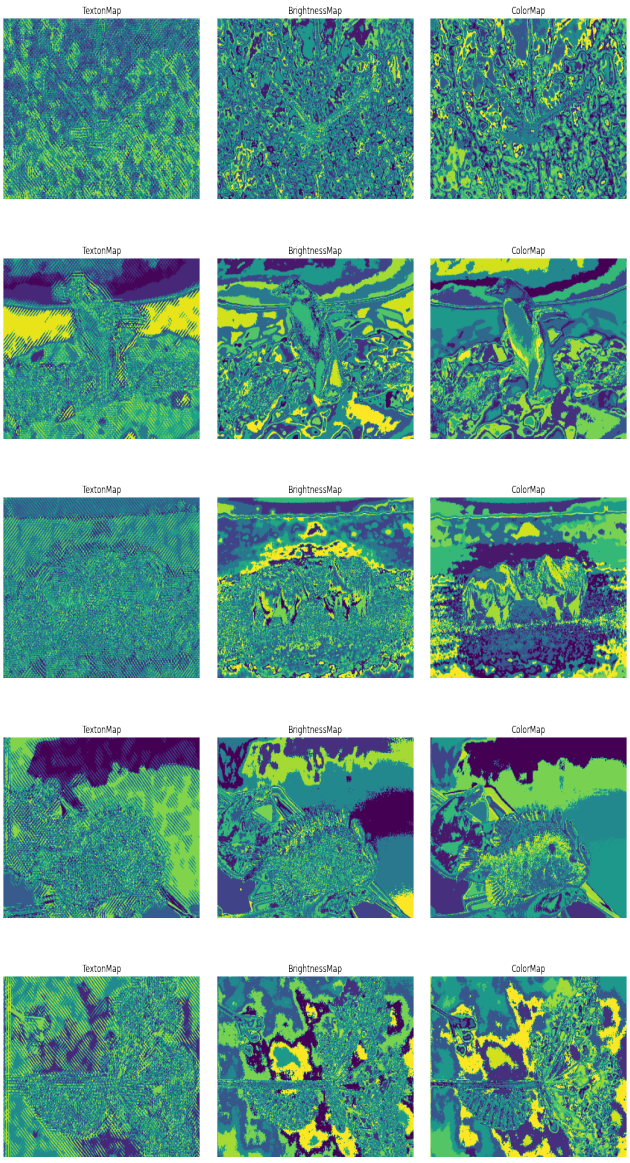
In the similar way a brightness map and a color map are also constructed using Kmeans clustering. The concept of the brightness map is to capture the brightness changes in the image and similarly for color map, the color changes are to be captured.

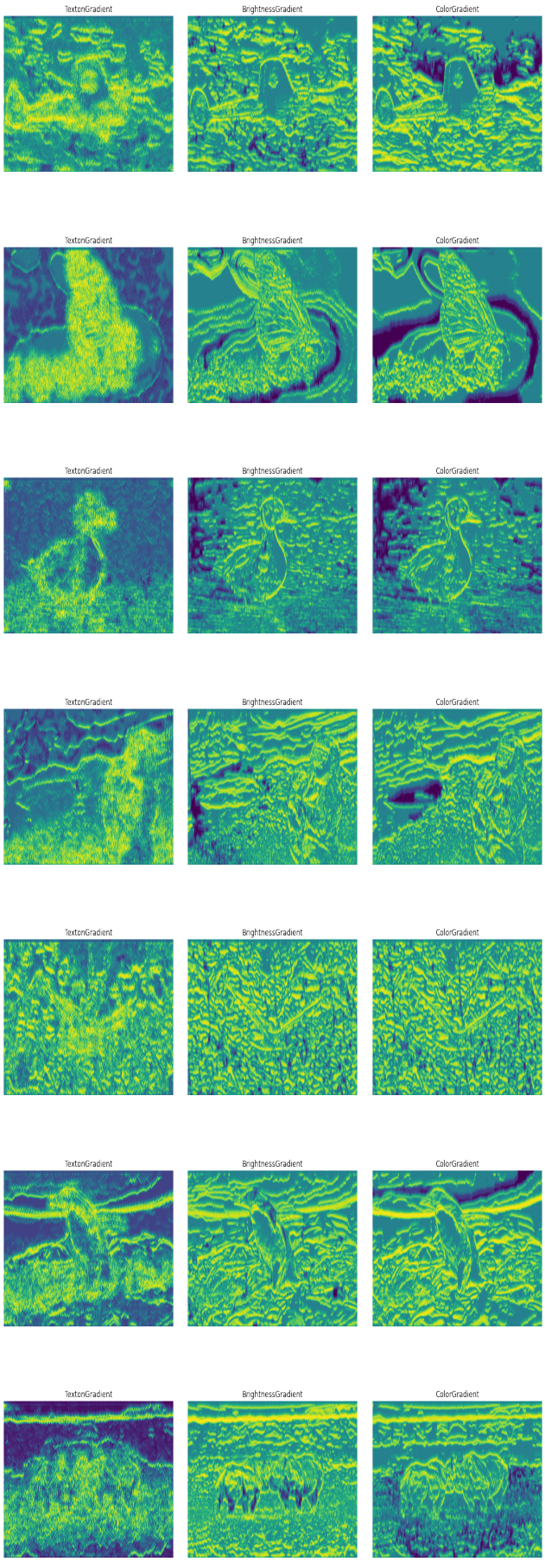
The output for all the images(1-10) is shown in the figures below.



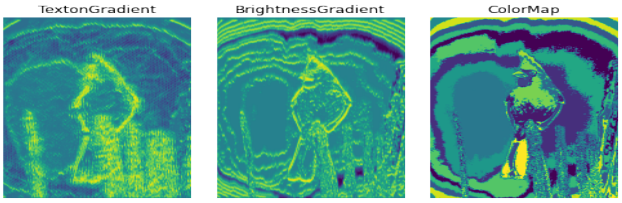
D. Texton, Brightness and color Gradient

The next step is to find the gradients of the texture, brightness and color at each pixel.



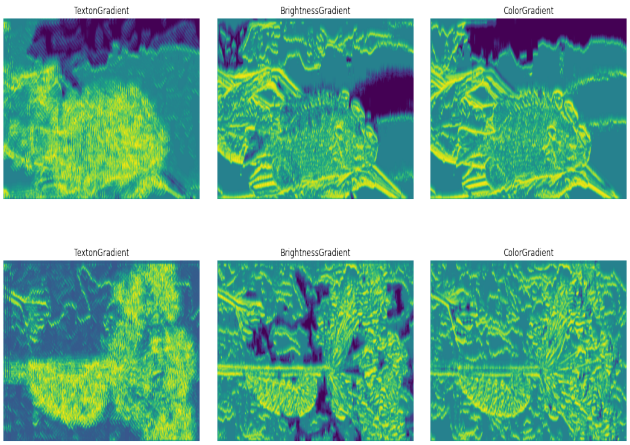


Using the generated half-disc masks along with the Chi-square distance, we can compute the gradient map for texture, brightness, and color of the input image. We obtain the gradient of the pixel by applying pairs of half-disk masks to each pixel of the maps and computing the Chi square distance.



E. Comparative output for Sobel, Canny and Pb-Lite

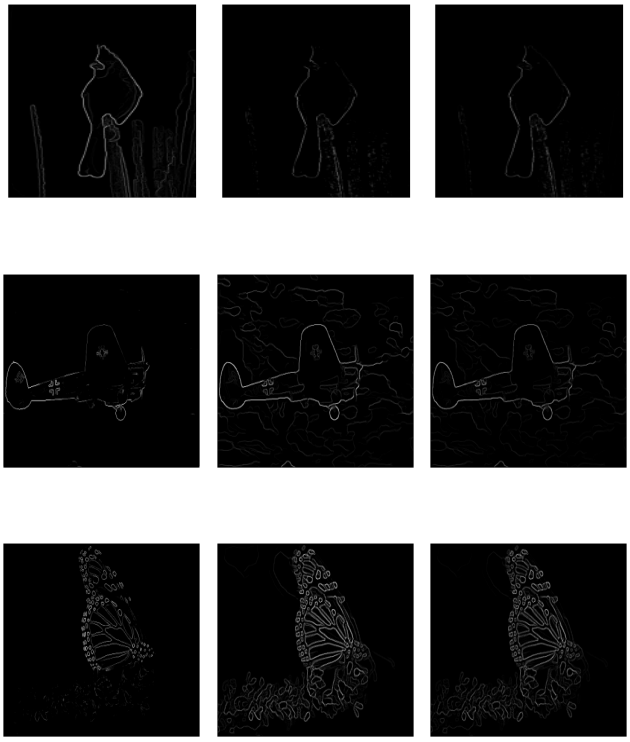
The final step is to combine information from the features with a Sobel or Canny edge detection or an average of both using a simple equation. Here, I have averaged the both using



the following simple formula. The values of w1 and w2 are kept as 0.1 and 0.9. By changing the values of w1 and w2, we can control the intensity of the dges and noise that can be filtered.

The output detected edges in the images are shown in the following figures.

As seen from the figures, though Canny edge detector seems to have better edges, the noise is also captured. The Pb-lite provides a stable edges with average advantages of both Sobel and canny edge detector. As an observation, by tuning the values of w1 and w2 the resultant Pb-lite output can be controlled.



F. Improvements that can be made and observations

As can be seen from the output comparative images, Canny edge detector seems to better detect edged. However, if

observed closely we notice noise present in the image. The Pb-lite output is much better resistant to noise.

Though, tested on very small dataset, the accuracy of the detector can be improved more with time. The Gaussian filter scales and sizes can be changed to get tuned outputs. Similarly in the case of other filters as well, small changes in the scales and sizes result in drastic effect on the output. The tuning of theses parameters can affect the overall results. Optimization functions can be written to automate the selection of parameters rather the trial-and-error method which is used in this assignment.

II. Phase 2: Deep dive in deep learning

Three different types of neural networks have been trained on CIFAR10-dataset for the scope of this assignment. With the given starter code further modifications were done to obtain training and testing checkpoints. The input size of the image is 3x32x32.

I tried to train the network with two different Loss functions and optimizers. The MAE loss function was not able to converge properly, hence I proceeded with using Cross Entropy for the final model. In comparison with SGD and ADAM optimizer, ADAM performed better giving higher accuracy. The learning rate for both was kept to be 0.0001. The default β_1 and β_2 parameters were used.

Due to hardware constraints, the time required for training was very high, so I decided to reduce the training set by the factor of 2. The BatchSize was also increased to 3.

The different variations tried, are explained below.

A. Simple network

As a starter neural network, I have implemented a simple 2 Convolutional + 3 Linear layer network. Kernel shape of 5x5 is used along with no padding and single stride. ReLU activation function is used. The architecture is shown in Fig.[41]

As mentioned earlier the optimizer is Adam and loss function is Cross Entropy. The training and testing accuracy is

This network is a very small, and mostly not preferable to train a large dataset as it can underfit the data. The size and complexity of the network can be increased to maximise the accuracy.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1            [-1, 6, 28, 28]             456
         MaxPool2d-2            [-1, 6, 14, 14]               0
            Conv2d-3           [-1, 16, 10, 10]           2,416
         MaxPool2d-4            [-1, 16, 5, 5]               0
            Linear-5                 [-1, 120]          48,120
            Linear-6                  [-1, 84]          10,164
            Linear-7                  [-1, 10]             850
================================================================
Total params: 62,006
Trainable params: 62,006
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 0.06
Params size (MB): 0.24
Estimated Total Size (MB): 0.31
----------------------------------------------------------------
```

Fig. 36: Simplified Neural Network architecture with parameters

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1            [-1, 6, 28, 28]             456
         MaxPool2d-2            [-1, 6, 14, 14]               0
       BatchNorm2d-3            [-1, 6, 14, 14]              12
            Conv2d-4           [-1, 16, 10, 10]           2,416
         MaxPool2d-5            [-1, 16, 5, 5]               0
       BatchNorm2d-6            [-1, 16, 5, 5]              32
            Linear-7                 [-1, 120]          48,120
            Linear-8                  [-1, 84]          10,164
            Linear-9                  [-1, 10]             850
================================================================
Total params: 62,050
Trainable params: 62,050
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 0.07
Params size (MB): 0.24
Estimated Total Size (MB): 0.32
----------------------------------------------------------------
```

Fig. 37: Modified neural network architecture with parameters

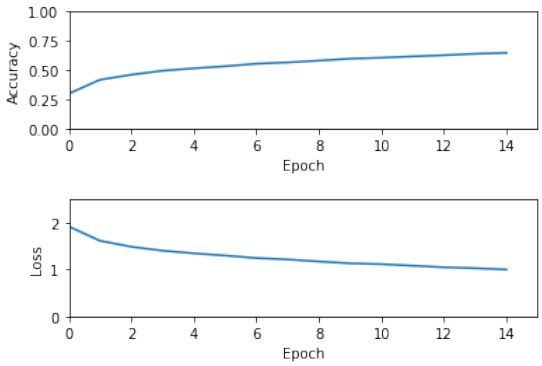


Fig. 38: Accuracy and loss for simple network

B. Modified network

The simple neural network architecture is modified further by adding two Batch normalization layer. The architecture is shown the Fig [46].

The accuracy increased to some extent in comparison to simple network by adding Batch normalization.

C. ResNet9

The ResNet9 is the simplest implementation of ResNet with 9 convolutional layers. The architecture is shown in Fig.[48]

To prevent over-fitting, we use a network with residual blocks instead of traditional neural networks, where each layer feeds into the next layer and directly into the layers about 2-3 next. The network is shown in the figure.

The network I have implemented has very low testing accuracy as the model was trained for less images and might have tend to overfit, observing at the training accuracy. With access to better hardware the training data can be increased which would in turn increase the testing accuracy.

D. DenseNet

Was not able to train DenseNet due to time constraint but did start with implementation. Doing it will provide a good learning opportunity.

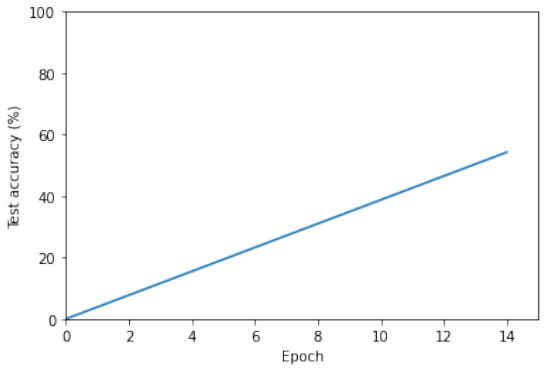


Fig. 39: Test Accuracy for modified network

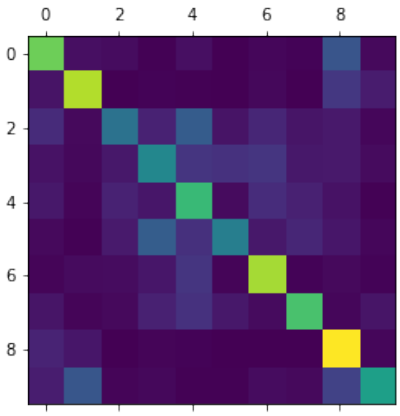


Fig. 40: Confusion matrix for testing accuracy of simple network

III. Conclusion

The comparative study of all the models is given in tabular form as follows,

Model	Train accuracy	Test accuracy
Simple network	57.49	45.36
Modified network	64.68	52.51
ResNet	72.5	10.0

The expected accuracy for Phase 2 of the assignment was not achieved. I was not able to tune the parameters more finely to get better accuracy. Also due to very less training, the testing accuracy also could not be increased.

I tried to implement the concepts however and it was on a very basic level. The models can be improved by adding more layers and by choosing proper parameters for activation, dropout, etc.

However, as seen from the table, Modified neural network has drastically increased the accuracy of the model.

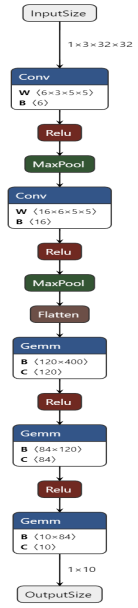


Fig. 41: Simple Network

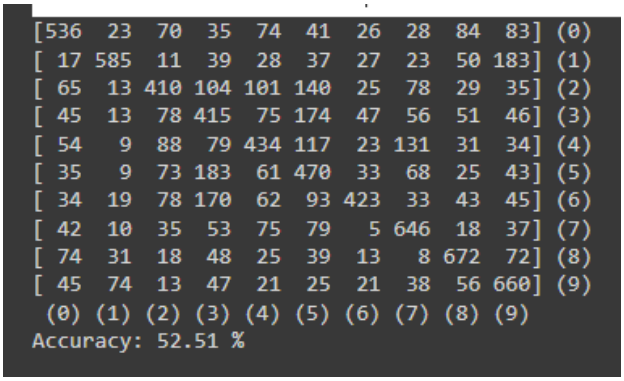


Fig. 42: Confusion matrix for testing accuracy of modified network

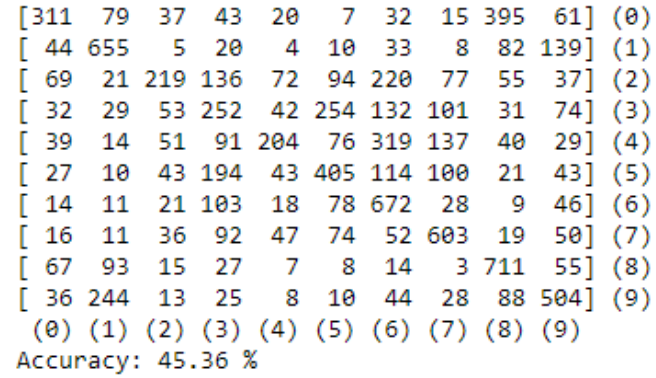


Fig. 43: Accuracy and loss for testing accuracy of simple network

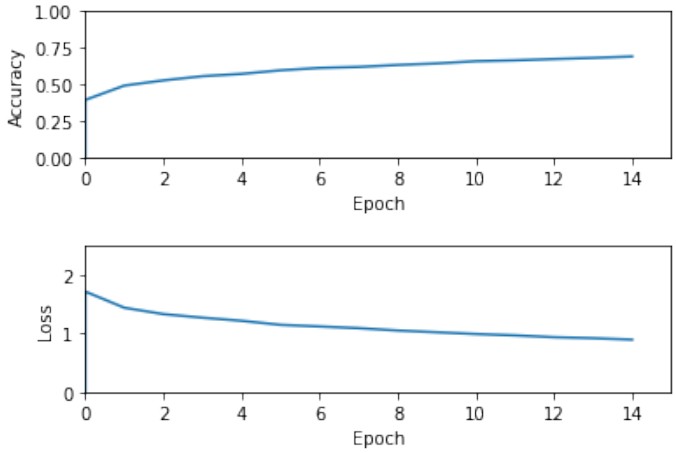


Fig. 44: Accuracy and loss for modified network

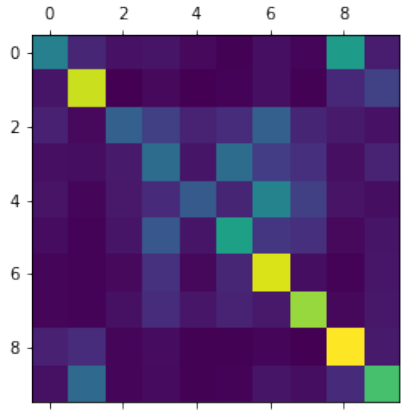


Fig. 45: Confusion matrix for modified network

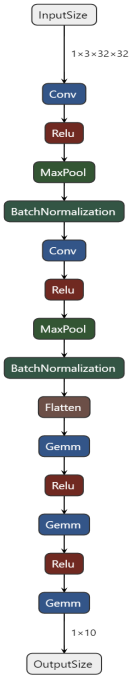


Fig. 46: Modified network

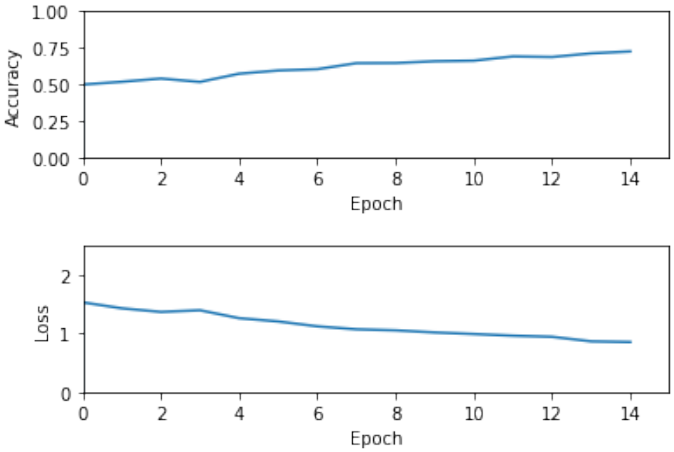


Fig. 47: Training accuracy and loss for ResNet9

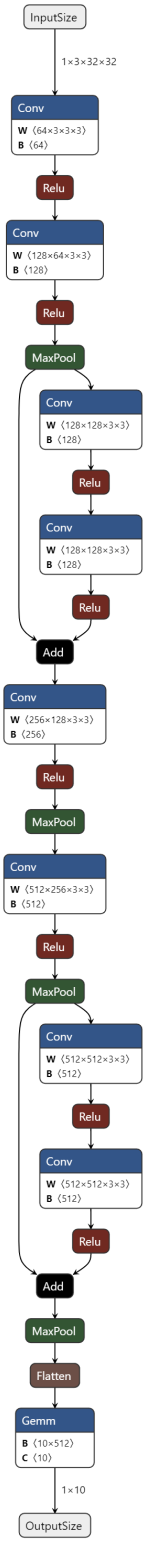


Fig. 48: ResNet9

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 64, 32, 32]           1,792
       BatchNorm2d-2          [-1, 64, 32, 32]             128
              ReLU-3          [-1, 64, 32, 32]               0
            Conv2d-4         [-1, 128, 32, 32]          73,856
       BatchNorm2d-5         [-1, 128, 32, 32]             256
              ReLU-6         [-1, 128, 32, 32]               0
         MaxPool2d-7         [-1, 128, 16, 16]               0
            Conv2d-8         [-1, 128, 16, 16]         147,584
       BatchNorm2d-9         [-1, 128, 16, 16]             256
             ReLU-10         [-1, 128, 16, 16]               0
           Conv2d-11         [-1, 128, 16, 16]         147,584
      BatchNorm2d-12         [-1, 128, 16, 16]             256
             ReLU-13         [-1, 128, 16, 16]               0
           Conv2d-14         [-1, 256, 16, 16]         295,168
      BatchNorm2d-15         [-1, 256, 16, 16]             512
             ReLU-16         [-1, 256, 16, 16]               0
         MaxPool2d-17           [-1, 256, 8, 8]               0
           Conv2d-18           [-1, 512, 8, 8]       1,180,160
      BatchNorm2d-19           [-1, 512, 8, 8]           1,024
             ReLU-20           [-1, 512, 8, 8]               0
         MaxPool2d-21           [-1, 512, 4, 4]               0
           Conv2d-22           [-1, 512, 4, 4]       2,359,808
      BatchNorm2d-23           [-1, 512, 4, 4]           1,024
             ReLU-24           [-1, 512, 4, 4]               0
           Conv2d-25           [-1, 512, 4, 4]       2,359,808
      BatchNorm2d-26           [-1, 512, 4, 4]           1,024
             ReLU-27           [-1, 512, 4, 4]               0
         MaxPool2d-28           [-1, 512, 1, 1]               0
          Flatten-29                 [-1, 512]               0
           Linear-30                  [-1, 10]           5,130
================================================================
Total params: 6,575,370
Trainable params: 6,575,370
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 9.07
Params size (MB): 25.08
Estimated Total Size (MB): 34.17
----------------------------------------------------------------
```

Fig. 49: ResNet9 model