RBE/CS549: Computer Vision Homework 0 - Alohomora

Shreya Bang
M.S. Robotics Engineering
Email: srbang@wpi.edu
Using 1 Late Day

***Abstract*—This assignment comprises of two phases. The Phase 1 presents the development of pb (probability of boundary) algorithm which improves the classical methods of edge detection like Canny and Sobel baselines. In Phase 2, multiple neural network architectures have been implemented and compared on the various criterion to study approaches to architectures work better for given CIFAR dataset.**

Index: Boundary Detection, Canny, Sobel, Deep Learning

I. PHASE 1: SHAKE MY BOUNDARY

A. Overview

The Phase 1 aims at the simplified version of Probability of boundary detection algorithm. It calculates the per-pixel probability of boundary by examining brightness, color, and texture information across multiple scale. This algorithm considers 4 steps:

1) Generating filter bank: Oriented DoG filters, Leung-Malik Filters and Gabor Filters
2) Computing Texton, Brightness and Color map
3) Finding the gradients of texture, brightness and color for each pixel
4) Combining the output with Canny and Sobel for boundary detection

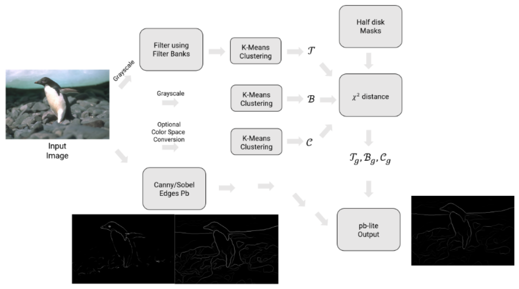


Fig. 1. Overview of the pb lite pipeline

B. Generation of filter bank

The first step of the pb lite boundary detection pipeline is to filter the image with a set of filter banks. We create three different sets of filter banks.

1) Oriented DoG Derivative of Gaussian filters: A simple DoG filter is created by convolving a Gaussian kernel with Sobel filter. Then rotated the results of DoG filters in 16 orientations ranging from 0 to 360 degree and 2 different scales which generated of 32 filters. The Sobel filter used is given by following:

$$S = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

For Oriented DoG filter bank, Gaussian kernel size = 17 and scales = [1,2] are taken. Illustration of the DoG filter bank is shown in Fig.2.

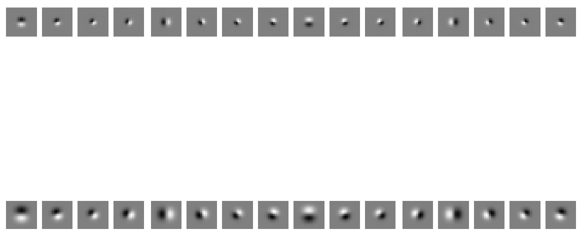


Fig. 2. Oriented DoG Filter Bank

2) Leung-Malik Filters: The Leung-Malik filters or LM filters are a set of multi scale, multi orientation filter bank with 48 filters. It consists of first and second order derivatives of Gaussians, Laplacian of Gaussian (LOG) filters and Gaussians. This filter has been applied in two versions: LM Small and LM Large. Scales considered for LM Small are $\sigma = \{1, \sqrt{2}, 2, 2\sqrt{2}\}$ and LM Large (LML) are $\sigma = \{\sqrt{2}, 2, 2\sqrt{2}, 4\}$. The first and second derivative of Gaussian occur at first three scales with $\sigma_x = \sigma$ and $\sigma_y = 3\sigma_x$ whereas the Gaussians occur at all the basic scales and LOG occur at σ and 3σ. The kernel size for Gaussian filter is 33. Illustrations of Leung-Malik Small and Large Filter Bank are shown in Fig.3 and Fig.4 respectively.

3) Gabor Filters: The gabor filter has a gaussian kernel modulated with a sinusoidal plane kernel. This consists of gabor filters with different standard deviation for the gaussian and different frequency for the sinusoids, which are further

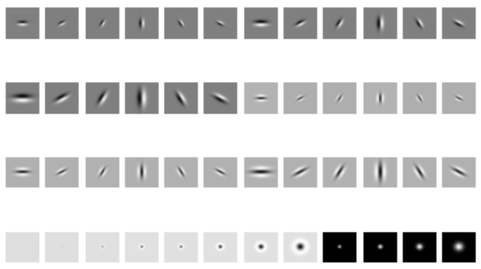


Fig. 3. LM Small Filter Bank

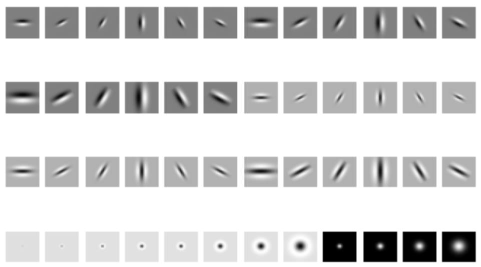


Fig. 4. LM Large Filter Bank

rotated to get different orientations. To generate these filters, it takes 5 variables. These variables are σ, θ, λ, ψ and γ, where λ is the standard deviation, θ is the rotation, λ is the wave length, ψ is the offset and γ is the offset of the filter in terms of width. Following is the illustration of Gabor filters generated at the scales = [5,7,9,11,13] with 8 orientations.

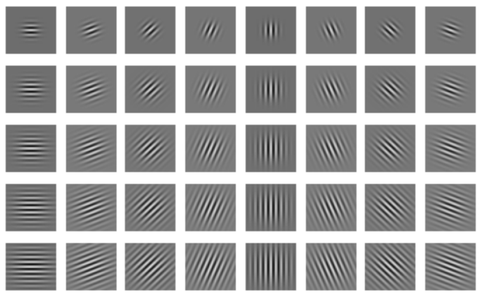
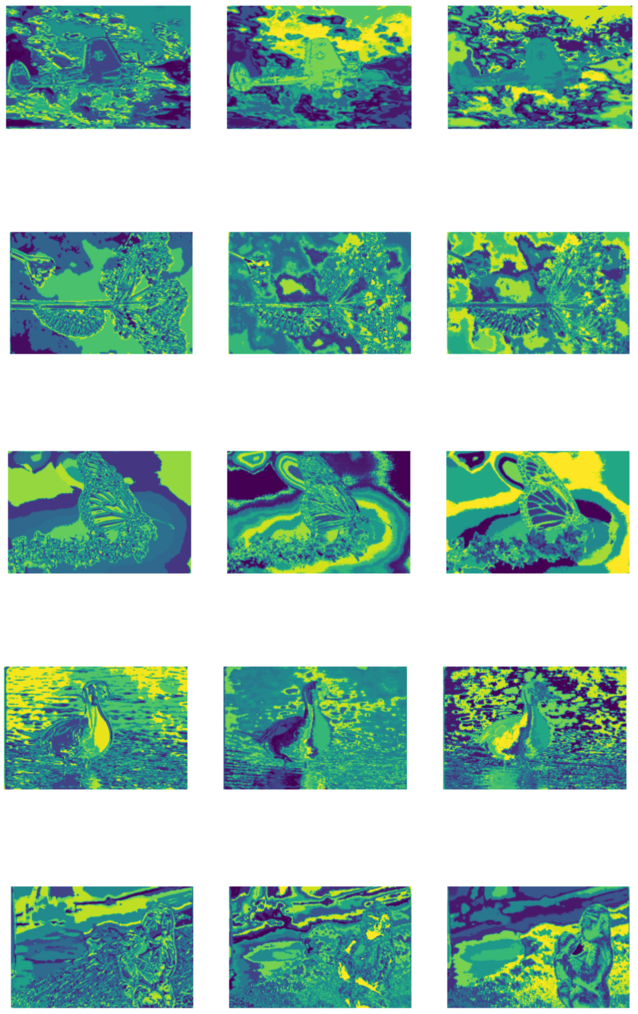


Fig. 5. Gabor Filter Bank

C. Texton, Brightness, and Color Map

To create Texton maps, all the filters (total 168) are applied to the image and a stack of resultant outputs is obtained. Now the task is to group the pixels having similar texture properties and give a discrete texton ID to each pixel. This is done by KMeans clustering method where the similar pixel values will be assigned to one cluster. In this case, 64 cluster centers are being used to categorize each pixel. We can infer that more number of clusters means more detail about the texture and vice-versa.

The texton map gives the information about texture properties in the image. To create Texton maps, individual filter from the filters bank (168, in this case) is applied on the image which results in a vector of filter responses centered on each pixel. Now, we have 168 filter responses at each pixel. Further, the pixels having similar texture properties are grouped using KMeans clustering method with number of clusters=64 and then a discrete texton ID is assigned to each pixel. Similarly, the brightness and color maps are used to encode the intensity and color values for each pixel respectively. Further, the brightness map is the default gray scale values clustered into 16 clusters and the colors is normalized RGB values clustered into 16 values. Results on the each image from BSDS500 dataset can be seen in Fig.6. (Figure(a): Texton Map, Figure(b): Brightness Map, Figure(c): Color Map)



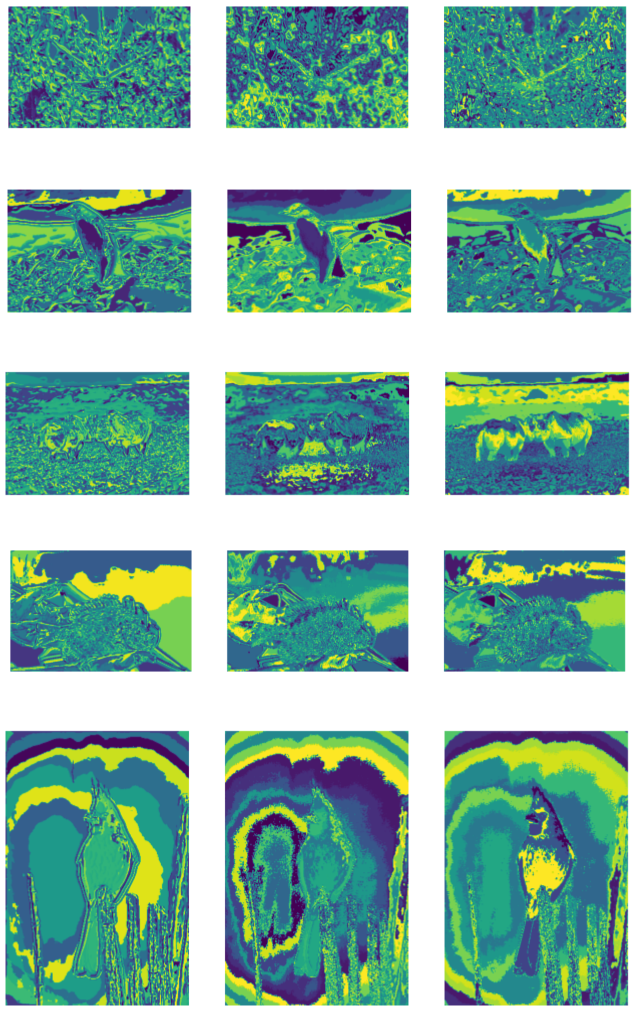


Fig. 6. Texton, Brightness and Color Maps

D. Texton, Brightness and Color Gradients

Gradient maps help us to define a series of local gradient measurements i.e. the change in distributions of texture, brightness and color at a particular pixel. For computing Texton, Brightness and Color gradients, we need to compute differences of values across different shapes and sizes. Hence, initially half disk mask and chi-square distance is generated.

1) Half-disc masks: The half-disc masks are simply pairs of binary images of half-discs. These discs are generated for 8 orientations. Illustration of Half-disc masks can be seen in Fig.7.

2) Chi-Distance: The Chi-Distance is used to calculate the distance between two histograms, that is, the difference between the distributions in left and right half-disc pairs. The Chi-Distance is calculated with the following equation:

The output afters computing texton, brightness and color gradients for all the images are given in Fig.8: (Figure(a): Texton Gradient, Figure(b): Brightness Gradient, Figure(c): Color Gradient)

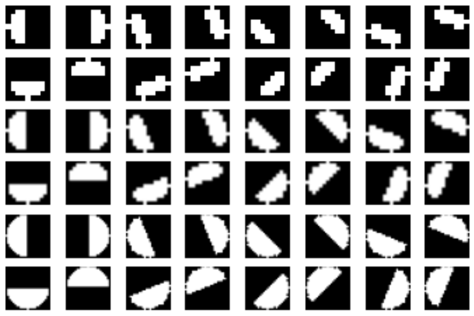
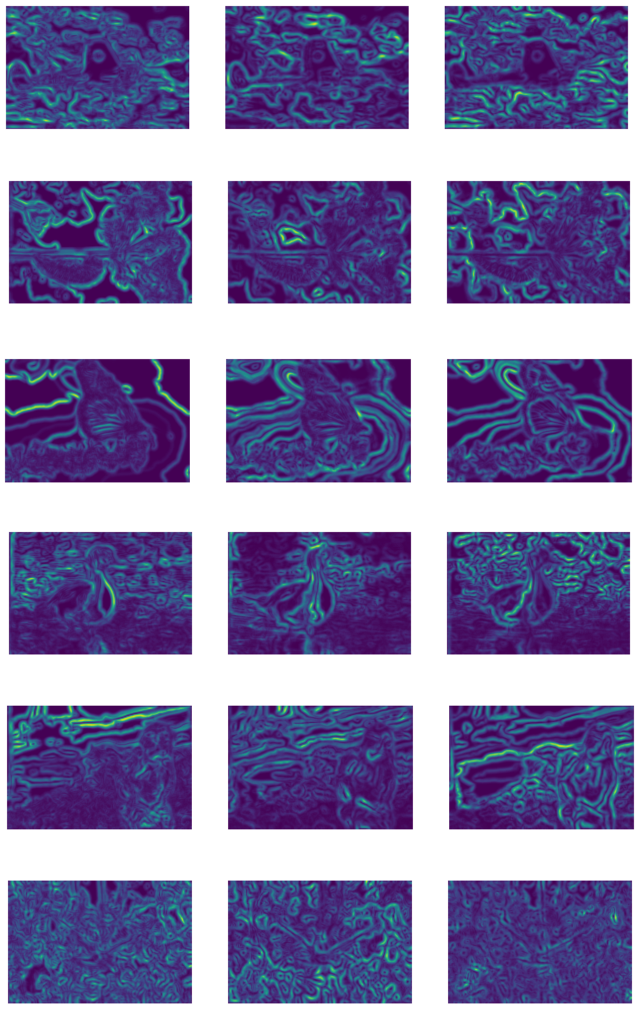


Fig. 7. Half-Disc Mask

$$\chi^2(g,h) = \frac{1}{2}\sum_{i=1}^{K}\frac{(g_i - h_i)^2}{g_i + h_i}$$



E. Pb-lite Output

The last step in the pipeline is to combine all the information of features obtained with Sobel and Canny edge detectors. For

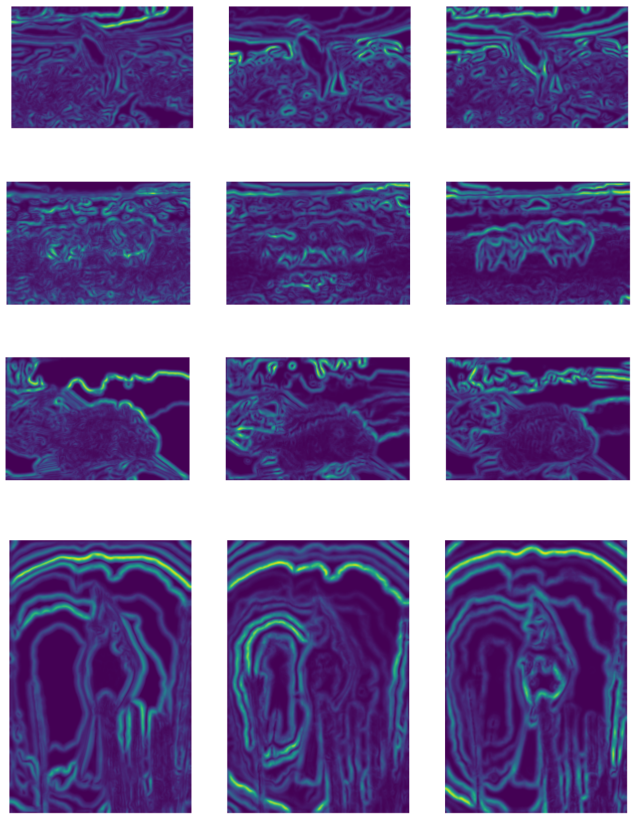


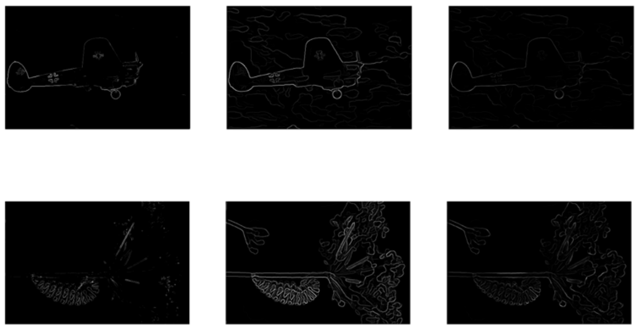
Fig. 8. Texton, Brightness and Color Gradients

the same, following equation is used.

Here, W1 and W2 in the equations are the weight. For the calculation purpose, values of W1 and W2 are taken as (0.9,0.1). However, one can make these weights dynamic.

$$PbEdges = \frac{(\mathcal{T}_g + \mathcal{B}_g + \mathcal{C}_g)}{3} \odot (w_1 * cannyPb + w_2 * sobelPb)$$

The comparative outputs between Canny baseline, Sobel baseline and Pb-lite output are shown in Fig.9.



F. Observation

Considering the above results, Pb-lite outperforms the Canny output in terms of cancellation of noise; whereas,

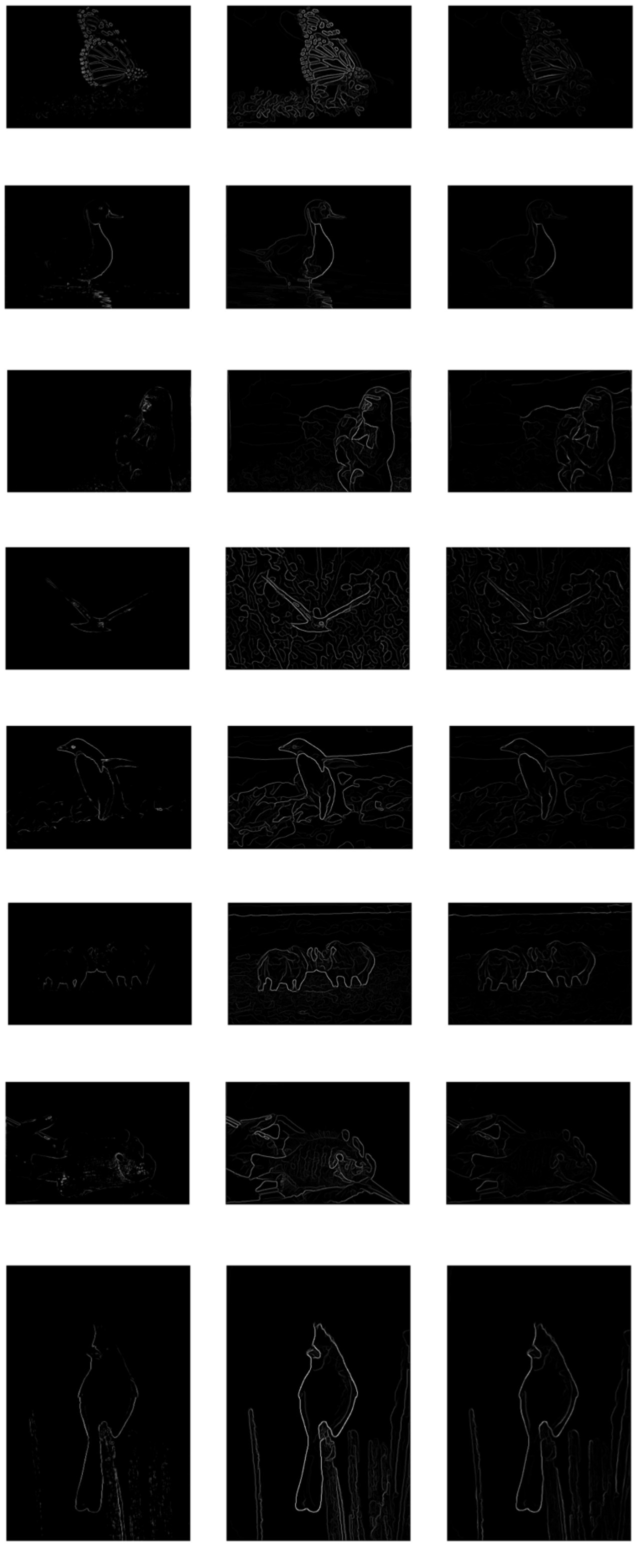


Fig. 9. Canny, Sobel and Pb-lite responses

outperforms the Sobel output in terms of edge detection. Since, Pb-lite can control the intensity of texture, brightness and color details, improvised responses can be obtained by choosing optimized weights. Also, filter bank plays an important role as responses can also be improvised with the optimum scales and kernel sizes.

II. PHASE 2: DEEP DIVE ON DEEP LEARNING

A. *Overview*

The Phase 2 focuses on the implementation of multiple neural network architectures CIFAR-10 dataset consisting of consists of 32x32 size 50,000 training and 10,000 test images for the image classification. Further these architectures are compared with each other on the basis of different factors like number of parameters, training and testing losses, training and testing accuracies.

B. *Basic Neural Network*

The basic architecture has 4 convolution layers with max pooling and 2 linear layers to classify the images into 10 classes.

The optimizer used is Adam with decay rate=0.0001 and learning rate=0.002. The network is structured with the following parameters.

1) Number of epochs: 15
2) Batch size: 100
3) Optimizer: ADAM

Without standardization and data augmentation, accuracy achieved on testing set is 48.57%. The training losses and accuracy has been plotted in Fig.10 and Fig.11. The confusion matrix for testing accuracy of simple neural network can be seen in Fig.12.

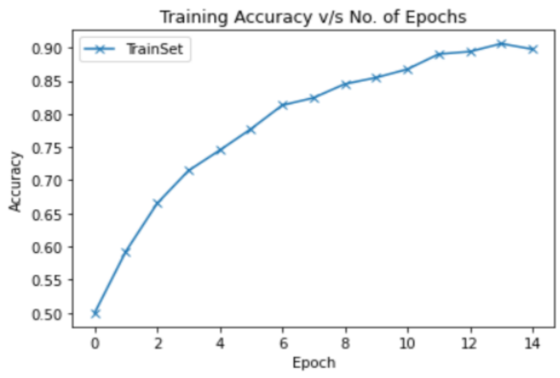


Fig. 10. Training Accuracy over Epoch of Basic Neural Network

C. *Modified Neural Network*

Considering the testing accuracies and losses of the Basic Neural Network, it has been modified in such a way to achieve more accuracy by decreasing the losses. For the same, basic model has been modified, which has 3 Convolution layers followed by 2 linear layes giving 10 as the output size. In the basic model, overfitting was observed. Therefore, to avoid overfitting, also to improve the learning speed of Neural Networks, batch normalization method has been opted. Further, addition of 3 batch normalization layers provided

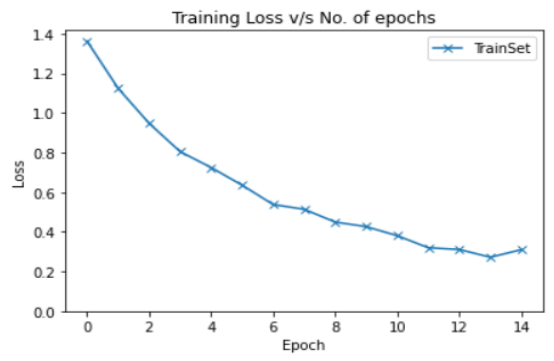


Fig. 11. Training Losses over Epoch of Basic Neural Network

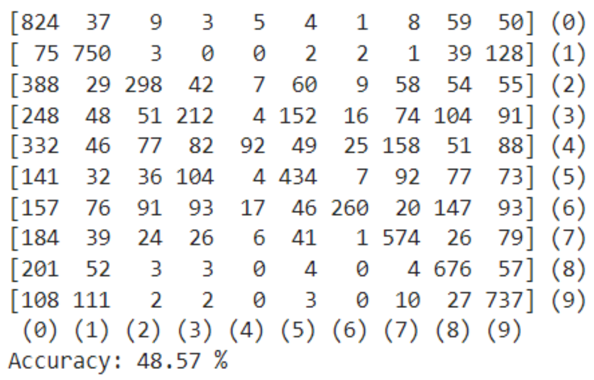

```
[824  37   9   3   5   4   1   8  59  50] (0)
[ 75 750   3   0   0   2   2   1  39 128] (1)
[388  29 298  42   7  60   9  58  54  55] (2)
[248  48  51 212   4 152  16  74 104  91] (3)
[332  46  77  82  92  49  25 158  51  88] (4)
[141  32  36 104   4 434   7  92  77  73] (5)
[157  76  91  93  17  46 260  20 147  93] (6)
[184  39  24  26   6  41   1 574  26  79] (7)
[201  52   3   3   0   4   0   4 676  57] (8)
[108 111   2   2   0   3   0  10  27 737] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 48.57 %
```

Fig. 12. Confusion Matrix for Testing Accuracy of Simple Neural Network

regularization. Keeping the optimizer used as same, Adam with decay rate=0.0001 and learning rate=0.002. This network is structured with the following parameters.

1) Number of epochs: 15
2) Batch size: 100
3) Optimizer: ADAM

The testing accuracy achieved with the modifications is 59.22%. The training losses and accuracy has been plotted in Fig.13 and Fig.14.

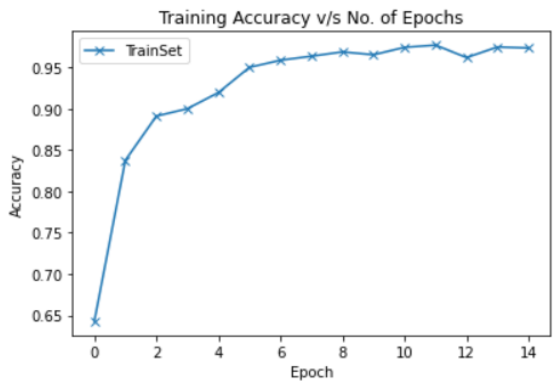


Fig. 13. Training Accuracy over Epoch of Modified Neural Network

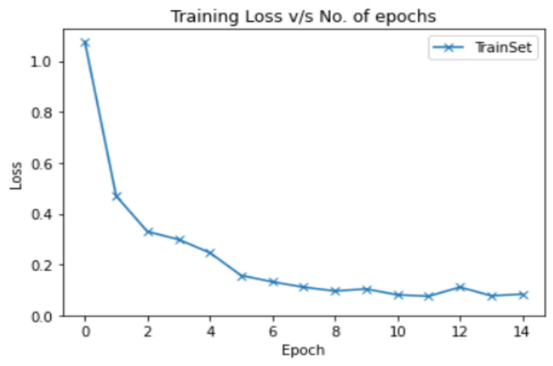
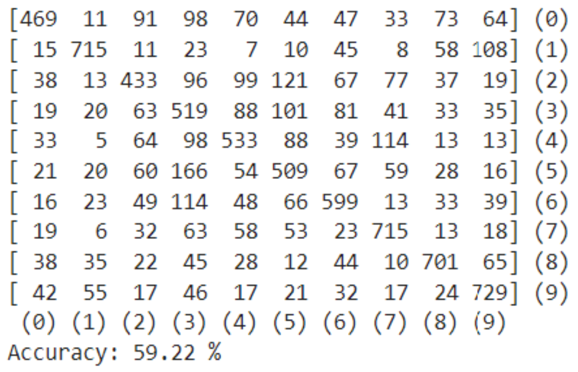


Fig. 14. Training Losses over Epoch of Modified Neural Network

The confusion matrix for testing accuracy of modified neural network can be seen in Fig.15.



```
[469  11  91  98  70  44  47  33  73  64] (0)
[ 15 715  11  23   7  10  45   8  58 108] (1)
[ 38  13 433  96  99 121  67  77  37  19] (2)
[ 19  20  63 519  88 101  81  41  33  35] (3)
[ 33   5  64  98 533  88  39 114  13  13] (4)
[ 21  20  60 166  54 509  67  59  28  16] (5)
[ 16  23  49 114  48  66 599  13  33  39] (6)
[ 19   6  32  63  58  53  23 715  13  18] (7)
[ 38  35  22  45  28  12  44  10 701  65] (8)
[ 42  55  17  46  17  21  32  17  24 729] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 59.22 %
```

Fig. 15. Confusion Matrix for Testing Accuracy of Modified Neural Network

D. *ResNet*

For the tasks like Image classifications, adding more layers to the neural network tend to increase the accuracy as more layers progressively learn more complex features. But it comes along with the problem of vanishing gradients. Hence to solve this problem, the skip connection in ResNet helps by allowing alternate shortcut path for the gradient to flow through. The ResNet architecture is implemented for the following parameters:

1) Number of epochs: 15
2) Batch size: 100
3) Optimizer: ADAM

The training losses and accuracy has been plotted in Fig.16 and Fig.17.

The confusion matrix for testing accuracy of modified neural network can be seen in Fig.18.

III. Conclusion

The architectures implemented in this homework over the CIFAR-10 dataset and the test accuracies achieved are tabulated below:

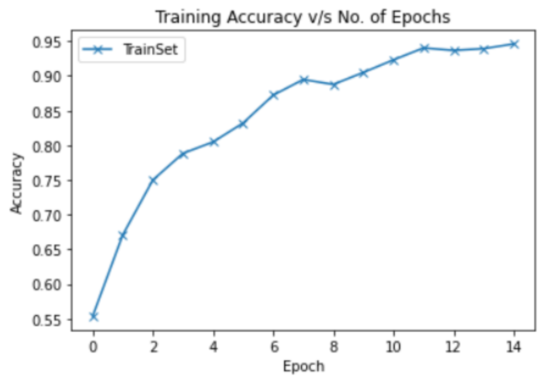


Fig. 16. Training Accuracy over Epoch of ResNet

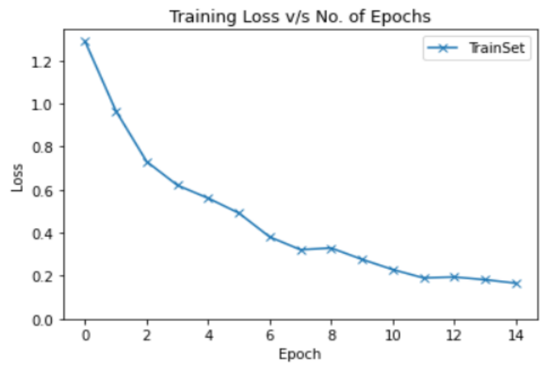
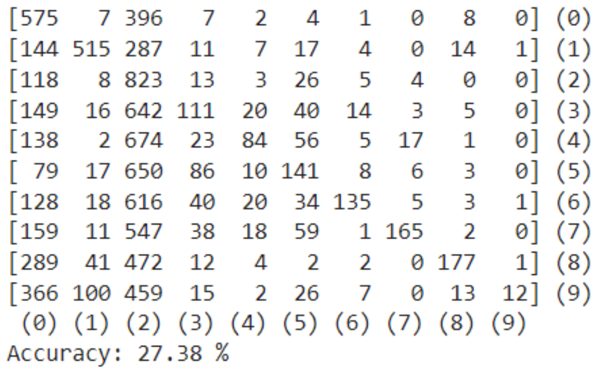


Fig. 17. Training Losses over Epoch of ResNet



```
[575   7 396   7   2   4   1   0   8   0] (0)
[144 515 287  11   7  17   4   0  14   1] (1)
[118   8 823  13   3  26   5   4   0   0] (2)
[149  16 642 111  20  40  14   3   5   0] (3)
[138   2 674  23  84  56   5  17   1   0] (4)
[ 79  17 650  86  10 141   8   6   3   0] (5)
[128  18 616  40  20  34 135   5   3   1] (6)
[159  11 547  38  18  59   1 165   2   0] (7)
[289  41 472  12   4   2   2   0 177   1] (8)
[366 100 459  15   2  26   7   0  13  12] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 27.38 %
```

Fig. 18. Confusion Matrix for Testing Accuracy of ResNet

From the above, we can infer that a little modification in the basic neural network tend to give better accuracy. The accuracy of the model can be improved with the addition of layers. Also tuning of the hyperparameters play an important role to achieve better accuracy.

Network	Test Accuracy
Basic Neural Network	48.57%
Modified Neural Network	59.22%
ResNet	27.39%