HW0: Alohomora
CMSC733

Sakshi Kakde
M. Eng. Robotics
University of Maryland
College Park, MD, 20742
Email: sakshi@umd.edu

***Abstract*—This report describes in brief my solutions for homework 0. The homework consists of two phases: To implement pb-lite algorithm for edge detection and to dive deep into deep learning.**

I. Phase 1: Shake my boundaries

In this section, a simple of version of PB boundary detection algorithm has been implemented. The classical approaches like Canny and Sobel edge detectors measures discontinuities in the image intensities to detect edges. The PB algorithm considers texture and color information along with intensity, making it a better performing algorithm.

A. Filter Banks

Filter banks are a set of filters, that are applied over an image to extract multiple features. In this homework, filter banks have been used to extract the texture properties. The following subsections will describe in brief the implementation of DoG filters, Leung-Malik filters and Gabor filters.

1) Oriented Derivative of Gaussian Filters: The filter uses the associative property of the convolution. Sobel operators S_x and S_y have been used to compute the gradient of a Gaussian(G) along x and y axis. To get the derivative of Gaussian at a desired angle, I used the following relationship:

$$G_(x, y) = |G_x| * cos(\theta) + |G_y| * sin(\theta)[1]$$

Where, G_x is first derivative of Gaussian(G) along x axis, G_y is first derivative of Gaussian(G) along y axis and θ is the orientation angle of derivative of Gaussian. Refer fig. 1 for the generated filterbank.

2) Leung-Malik Filters: The Leung-Malik filter consists of first and second order derivatives of Gaussians filters, Laplacian filters and Gaussian filters. Initially, a 2D elliptical Gaussian was plotted with an elongation factor of 3, and was later transformed using a rotation matrix to get the desired orientation. Sobel operators were used to calculate the first derivative. The results were again convoluted with Sobel operators to get second derivative. Symmetrical Gaussians with 8 scales were convoluted with a Laplacian kernel to get the Laplacian filters. A total of 4 Gaussian filters were used at different scales. Refer fig. 2 for the generated filterbank.

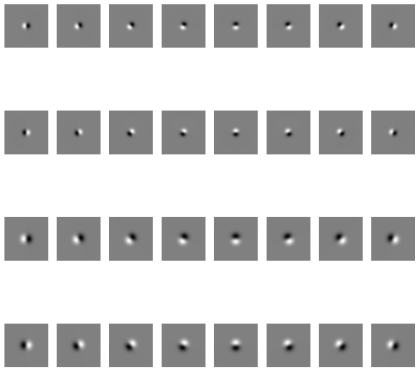


Fig. 1. Oriented DoG filters: 2 scales, 16 orientations

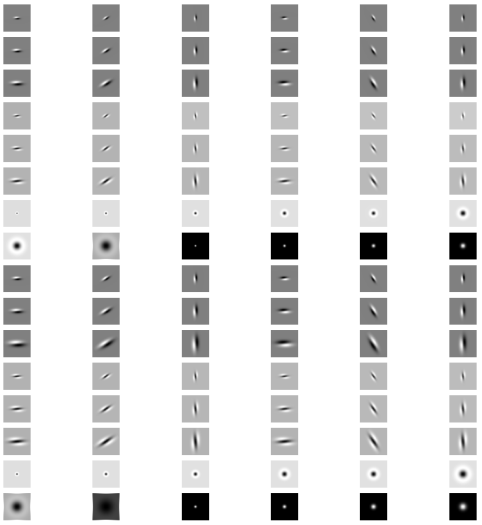


Fig. 2. Leung-Malik Filters: LM large and LM small

3) Gabor Filters: To begin with, a 2D sine matrix was generated at the desired orientation, which was then used to modulate a 2D symmetrical Gaussian matrix. Refer fig. 3 for the generated filterbank.

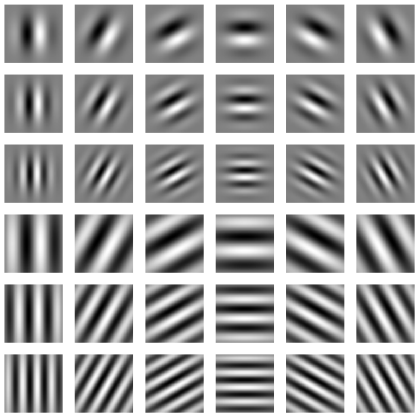


Fig. 3. Gabor filter: 2 scales, 6 orientation, 3 frequencies

B. Texton Maps

The filters described in the previous section are to detect the texture properties in an image. Since each filter bank has multiple filters and three such filter banks have been used, the result is a vector of filter responses. This vector of filter response associated with each pixel is encoding a texture property. Based on this filter response vectors, pixels with similar texture property were clustered together using K-mean algorithm(K= 64). The output of K-mean clustering is a Texton map(τ).

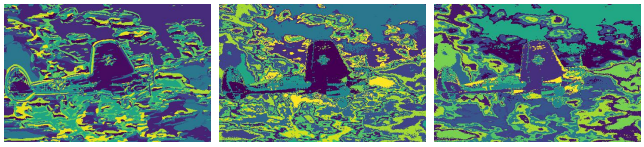
C. Brightness Maps

The image was clustered based on the brightness value for each pixel. The images were first converted to gray scale and K-mean algorithm(K=16) was used to get brightness maps(β).

D. Color Maps

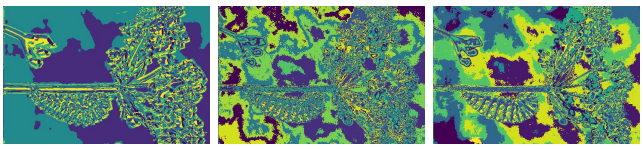
The image consists of three color channals(RGB), describing the color property at each pixel. The images have been clustered using the RGB value using K-mean algorithm(K=16) to obtain color maps(ζ).

Refer fig. 4 to fig. 13 for texton, brighteness and color maps for all images.



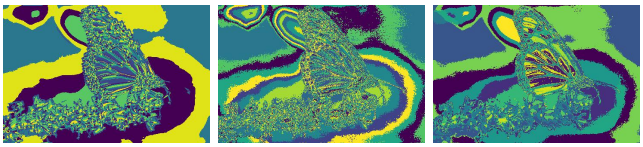
(a) (b) (c)

Fig. 4. τ, β, ζ for image 1



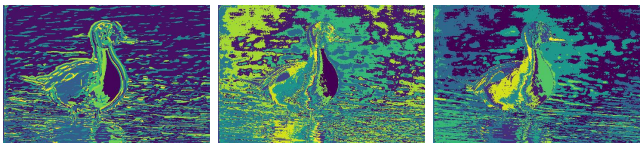
(a) (b) (c)

Fig. 5. τ, β, ζ for image 2



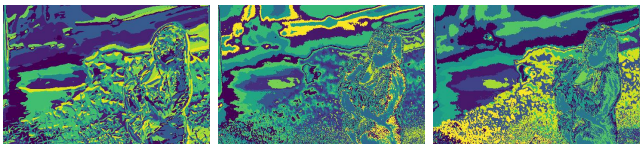
(a) (b) (c)

Fig. 6. τ, β, ζ for image 3



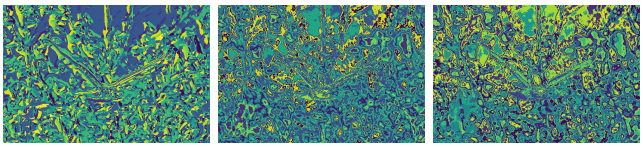
(a) (b) (c)

Fig. 7. τ, β, ζ for image 4



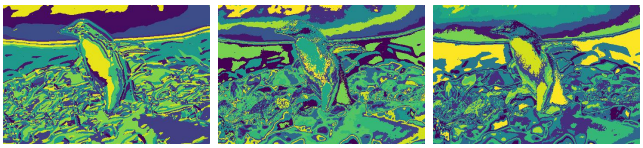
(a) (b) (c)

Fig. 8. τ, β, ζ for image 5



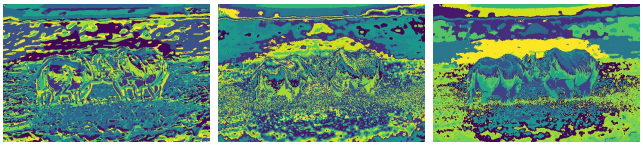
(a) (b) (c)

Fig. 9. τ, β, ζ for image 6



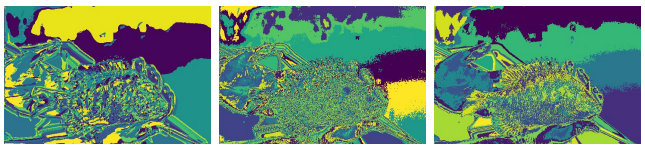
(a) (b) (c)

Fig. 10. τ, β, ζ for image 7



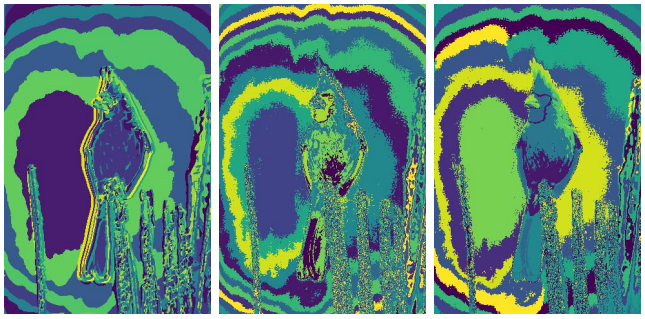
(a) (b) (c)

Fig. 11. τ, β, ζ for image 8



(a) (b) (c)

Fig. 12. τ, β, ζ for image 9



(a) (b) (c)

Fig. 13. τ, β, ζ for image 10

E. Texture, Brightness and Color Gradients

1) Half disc masks: Half Disk masks are binary images of half circle. They have been used to compute the change of texture or brightness distributions at different scales and angles. They have been used to compare the texture, color and brightness properties over the disk area, rather than the immediate pixel, thus making it a softer filter as compared to the Sobel operator. Refer fig. 14 for the generated masks.

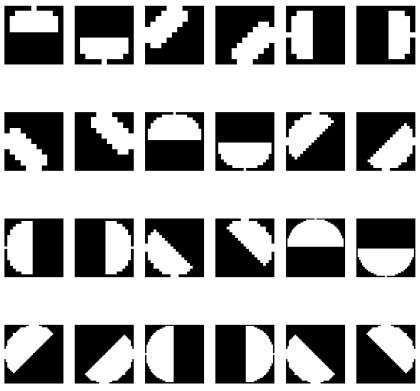


Fig. 14. Half Disk Masks

2) Chi-square distance: To compute the χ^2 distance, first, the images have been convoluted with left/right half-disc pairs to get two histograms g_i and h_i. The χ^2 distance is computed using the below formula.

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

The χ^2 distance has been computed for each texton, brightness and color map. For N half disk pairs, we get N matrices with pixel value corresponding to the χ^2 distance. An average of these N matrices have been used to get the gradient of τ, β, ζ.

F. Sobel and Canny baseline

The outputs from Sobel and Canny edge detector are combined using weighted average method.

G. Pb-lite output

In the final step, the features from baseline methods(Canny and Sobel operator) were combined with the gradients of τ, β, ζ using the following equation:

$$PbEdges = \frac{t+g+b}{3} \circ (w_1 * cannyPb + w_2 * sobelPb)$$

The $\circ$ is used to represent Hadamard product operator, which is elementwise multiplication between two matrices. w_1 and w_2 are set as 0.5, thus making the weighted average a simple mean. The output of pb-lite algorithm is visualized along with the Sobel and canny baselines. Refer fig. 15 to fig. 24.

(a) (b) (c)

Fig. 15. Canny, Sobel and PB lite output for image 1

(a) (b) (c)

Fig. 16. Canny, Sobel and PB lite output for image 2

(a) (b) (c)

Fig. 17. Canny, Sobel and PB lite output for image 3

(a) (b) (c)

Fig. 18. Canny, Sobel and PB lite output for image 4

(a) (b) (c)

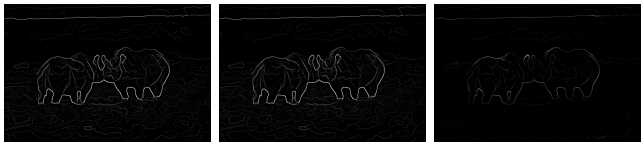
Fig. 19. Canny, Sobel and PB lite output for image 5

(a) (b) (c)

Fig. 20. Canny, Sobel and PB lite output for image 6

(a) (b) (c)

Fig. 21. Canny, Sobel and PB lite output for image 7

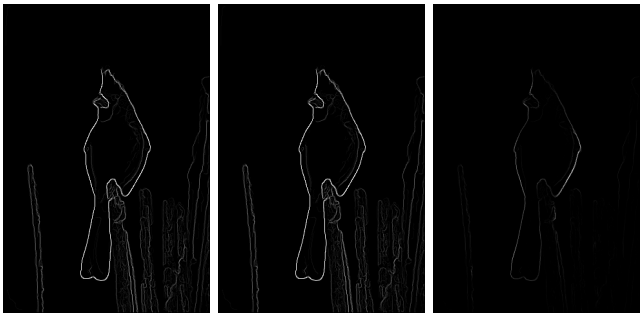


(a) (b) (c)

Fig. 22. Canny, Sobel and PB lite output for image 8

(a) (b) (c)

Fig. 23. Canny, Sobel and PB lite output for image 9



(a) (b) (c)

Fig. 24. Canny, Sobel and PB lite output for image 10

H. Analysis

The pb-lite algorithm is able to detect the edges of the main object in an image. Though the edges of pb-lite algorithm are not as prominent as for the Canny and Sobel detectors, there is considerable reduction in noise which is mainly caused due in the uneven texture, brightness and color. I thought increasing the number of half disk masks might help is getting better results, since more change in texture, color and brightness will be captured. I tried with 12 orientation and 4 scales half disk masks, and after a comparison with 8 orientation and 3 scales half disks masks, it was observed that there was a little increase in the sharpness of the edges for a few images. However, for some images, the edges became blurry. Similarly, I think increasing the number of Gabor filters could help in better texture segmentation. Due to time limitations, I am unable to experiment with that.

II. Phase 1: Deep Dive into Deep Learning

In this section, a basic neural network and its modified version for classification on CIFAR10 dataset have been described. Later, a case study for ResNet, ResNext and DenseNet architecture was conducted. A simplied versions(with lesser number of filters and lesser depth) for these architecture were implemented in TensorFlow and the results are presented. Due to limited hardware resources and time, I was able to experiment with only a few parameters.

A. My first neural network

This is a simple network with two convolution layers followed by two fully connected layers. The optimization technique used is Adam's optimizer and Softmax function is used to calculate the loss. A learning rate of 0.001 was used to train the network and a batchsize of 50 was used. Refer fig. 25(a) for the block diagram of the architecture. The testing accuracy is 45.2 %.

B. Modified network

To the above described model, I made the following modifications:

1) Batch normalization
   Reason: helps in properly initializing neural networks by explicitly forcing the activation throughout a network to

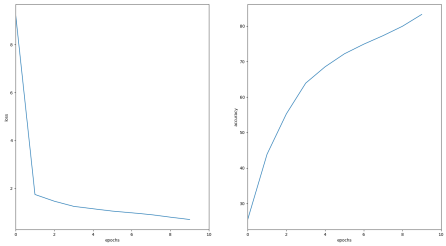
(a) First Neural Network (b) Modified Neural Network

Fig. 25. Block diagram for first and modified neural networks

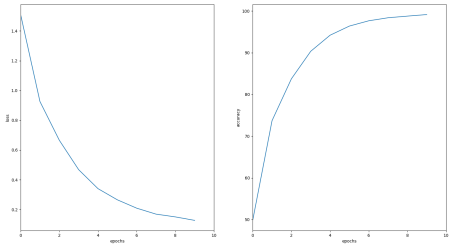
take on a unit Gaussian distribution at the beginning of the training.[3]

2) Image Standardization
   Reason: It might help the network to learn the parameters more quickly.
3) Filter size
   Reason: As the filter size increases, more features can be extracted. Thus increasing the accuracy of the network.
4) Network Depth
   Reason: It may help in learning more complex and non-linear functions.

The network after modification is shown in fig. 25(b). The comparison between first neural network and the modified version is made using the loss and accuracy vs epoch graphs(refer fig 26, fig 27) and confusion matrices(refer fig 28, fig 37). The testing accuracy is 58.62 %. A learning rate of 0.001 was used to train the network and a batchsize of 50 was used.



(a) Training loss and accuracy for simple neural network

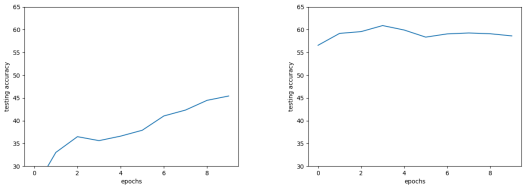


(b) Training loss and accuracy for modified neural network

Fig. 26. Training loss and accuracy for simple and modified neural network

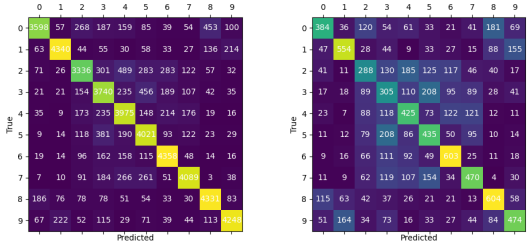
C. Case Studies

1) ResNet: The intuition behind the ResNet architecture is that it uses network layers to fit a residual mapping, instead



(a) (b)

Fig. 27. Testing accuracy for simple and modified neural network



(a) Training (b) Testing

Fig. 28. Confusion matrix for training and testing for simple neural network.

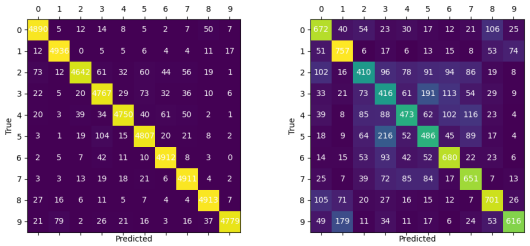
of trying to fit the desired mapping[3]. Thus the network will learn what it needs to add/subtract from the original input to get the desired output. The block diagram for my implementation of the ResNet architecture is provided in TO DO. Please note that is not an actual ResNet architecture, but a simple implementation to understand the idea(Fig. 30). The accuracy I got is 58.28 %. A learning rate of 0.001 was used to train the network and a batchsize of 50 was used.

2) ResNext: The ResNext architecture follows split transform strategy[4]. As per the research, having a high cardinality can get us better results than having high depth and width. The block diagram for my implementation of the ResNext architecture is provided in fig. 31. Please note that is not an actual ResNext architecture, but a simple implementation to understand the idea. The accuracy I got with this model is 56.4 %. A learning rate of 0.001 was used to train the network and a batchsize of 50 was used.

3) DenseNet: Instead of drawing representational power from extremely deep or wide architectures, DenseNets exploit the potential of the network through feature reuse[5]. The idea behind this architecture is that by connecting the outputs from previous layers, the network requires fewer parameters than an equivalent traditional CNN, as there is no need to learn redundant feature[5]. The block diagram for my implementation of the DenseNet architecture is provided in fig. 32. Please note that is not an actual DenseNet architecture, but a simple implementation to understand the idea. The accuracy I got with this model is 60.17 %. A learning rate of 0.001 was used to train the network and a batchsize of 50 was used.

D. Analysis

A brief comparison of all the networks implemented is presented in the table below. There is an increase in accuracy



(a) Training (b) Testing

Fig. 29. Confusion matrix for training and testing for modified neural network.

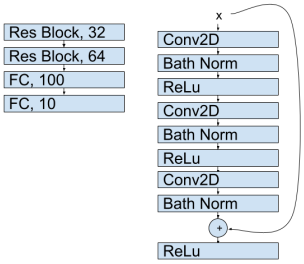


Fig. 30. ResNet block diagram(L), residual block diagram(R).

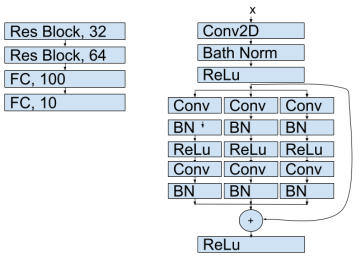


Fig. 31. ResNext block diagram(L), residual block diagram(R).

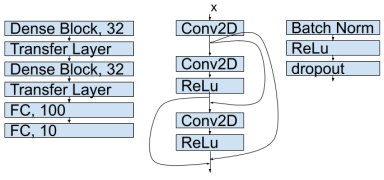
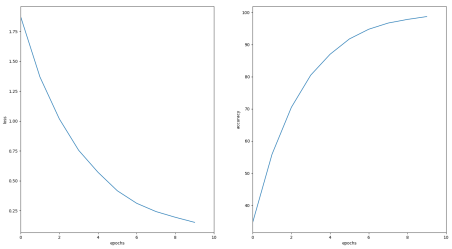
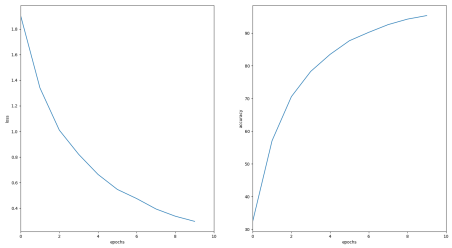


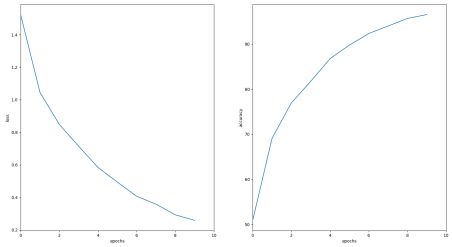
Fig. 32. DenseNet block diagram(L), dense block diagram(M), transfer layer(R).



(a) Training loss and accuracy vs epochs for ResNet

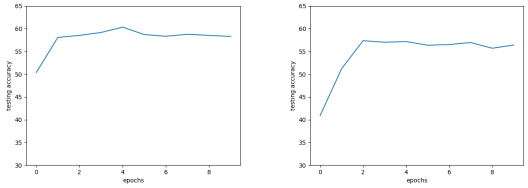


(b) Training loss and accuracy vs epochs for ResNext

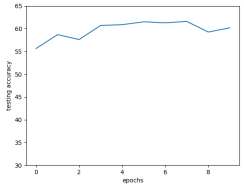


(c) Training loss and accuracy vs epochs for DenseNet

Fig. 33. Loss and accuracy vs epochs

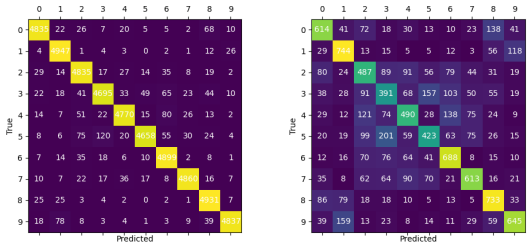


(a) Testing accuracy vs epochs for ResNet (b) Testing accuracy vs epochs for ResNext



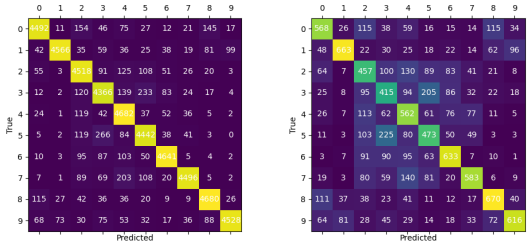
(c) Testing accuracy vs epochs for Densenet

Fig. 34. Testing accuracy vs epochs



(a) Training (b) Testing

Fig. 35. Confusion matrix for training and testing for ResNet.



(a) Training (b) Testing

Fig. 36. Confusion matrix for training and testing for ResNext.

for the modified neural network, as against the simple first neural network. The ResNet, ResNext and DenseNet architectures did not perfume as expected. Looking at the graphs, it seems like the ResNet model is getting over-fitted. Due to limited time and resources, I could not increase the network depth and filter size beyond a particular value. I think this could have been a reason for the inaccuracies. The best accuracy is obtained using the DenseNet architecture.

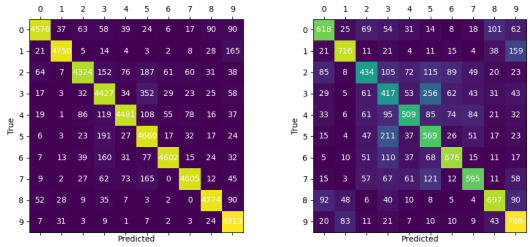
Model	No. of parameters	Training Accuracy	Testing Accuracy
First Neural Network	2170166	80.072%	45.2%
Modified Neural Network	3745686	96.61%	58.62%
Resnet	6790774	96.53%	58.28%
ResNext	1724582	90.82%	56.40%
DenseNet	4086134	92.224 %	60.17%

III. Conclusion

For edge detection in an image, pb lite algorithm has been implemented. I was able to train and test my first neural network for classification problem on CIFAR10 dataset along with modifications to increase the accuracy. Case studies for ResNet, ResNext and DenseNet was conducted and a simple and basic block for each was implemented.

References

[1] https://www.cs.cornell.edu/courses/cs6670/2011sp/lectures/lec02_filter.pdf



(a) Training (b) Testing

Fig. 37. Confusion matrix for training and testing for DenseNet.

[2] https://github.com/taki0112/Densenet-Tensorflow
[3] https://cs231n.github.io/neural-networks-3/
[4] https://medium.com/@14prakash/understanding-and-implementing-architectures-of-resnet-and-resnext-for-state-of-the-art-image-cc5d0adf648e
[5] https://towardsdatascience.com/understanding-and-visualizing-densenets-7f688092391a