Homework 0 - Alohomora

Niranjan Kumar Ilampooranan
MS Robotics Graduate Student
Worcester Polytechnic Institute
(Using 3 late days)

***Abstract*—Two different applications - Boundary Detection and Image Classification - are the focus of this homework and the solutions for the above using the specified algorithms and architectures are discussed in this report. Boundary detection - Phase 1 - involves filter design for convoluting over the test image set to obtain a mapping of confidence values on the presence of boundaries of the picture (Pb) and comparing the obtained results with Canny and Sobel - two famous algorithms for the same application. For image classification, a simple neural network is developed and augmented as per given suggestions along with training and testing with the CIFAR-10 dataset along with other architectures such as ResNet, ResNeXt, and DenseNet. The performance of these networks are provided in the form of plots and confusion matrix.**

PHASE 1

The first phase of the homework involves developing Pb-lite, a boundary detection algorithm, from scratch (albeit a simplified version of Probability of boundary algorithm) and pitting it against the baselines of well-known algorithms - Canny and Sobel. To do so, some filters such as Gabor and Leung-Malik are required along with half-disk masks. This shall be explored in subsequent sections in which the entire process behind developing pb-lite is explained. The crucial landmarks are as follows.

- Developing DoG, LM, Gabor, and Half-Disk filter banks
- Generating the Texton, Brightness, and Color Maps along with obtaining gradient maps T_g, B_g, and C_g respectively
- Using the above to obtain the response of image to pb-lite and compare with the Sobel and Canny baselines

A. Filter Generation

The DoG filter bank is quite straightforward in its generation. First, Gaussian filters of different scales are convoluted with Sobel kernel and is further rotated to obtain filters of various orientations. For this homework, scale of 1 and $\sqrt{2}$ is used. The number of orientations for which the filter was rotated after convolution is 16. This gives a total of 32 oriented DoG filters. The filters are shown below in Fig. 1.

As far as the LM filters are concerned, there are four different sets to it (for each of LM large and LM small filter banks). The LM small filter bank is shown in Fig. 2 and LM large filter bak is shown later in Fig. 3. The first two sets involve obtaining the first and second order derivative of the Gaussian filter for different sets of scaling and orientations. Furthermore, there is an elongation factor in the standard deviation in the x and y components involved ($\sigma_y = 3\sigma_x$). This gives 36 filters in total for these sets.

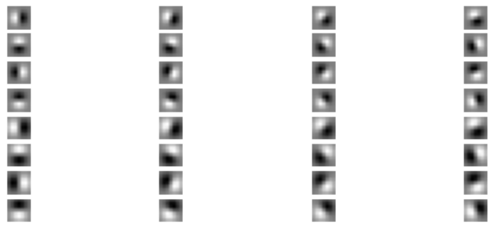
Fig. 1. Oriented DoG Filter Bank

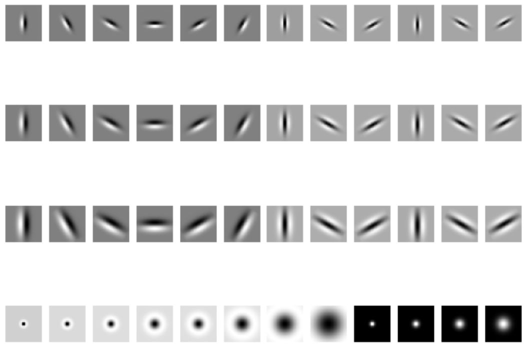
Fig. 2. LM Small Filter Bank

In the third set, eight Laplacian of Gaussian (Laplacian operator over Gaussian) filters are generated and regular Gaussian filters for the final set. The different between LM large and LM small filter banks are the usage of different scaling factor between them (LM large uses a larger scaling factor/standard deviation).

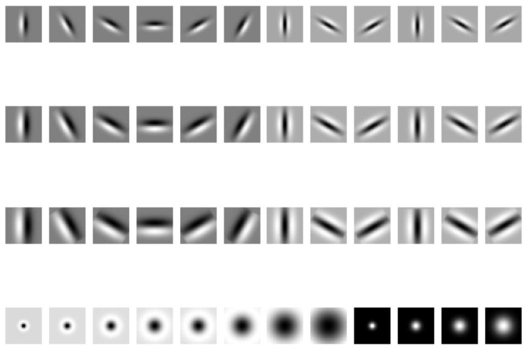
Fig. 3. LM Large Filter Bank

The Gabor filter bank can be generated by convoluting the standard Gaussian kernel with sinusoid (equivalent to multiplying the Fourier transforms of both the signals). The result is shown in Fig. 4. Multiple filters are obtained for different scales and orientations.

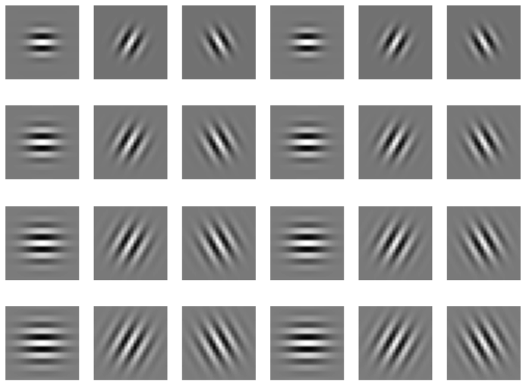

Fig. 4. Gabor Filter Bank

Finally, the half-disk masks or filters are obtained by populating a blank image (2D array of zeros) with ones in a semicircular region and the successive filter is populated in a way that it complements its predecessor. This filter is key in calculating the χ^2 distance which will be the focus of discussion next. The half-disk masks are shown below in Fig. 5. These masks shown are for different scaling and orientations (three scales and sixteen orientations respectively)

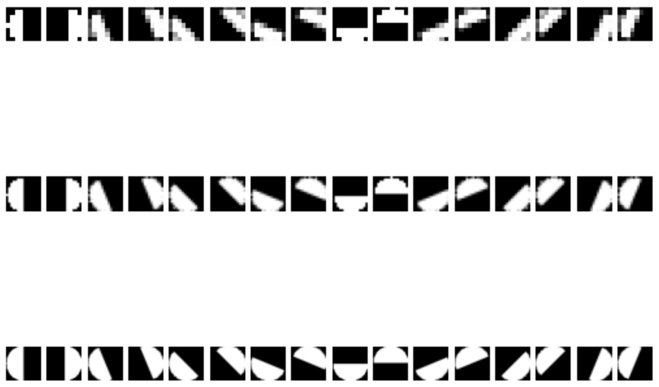

Fig. 5. Half-disk Mask Sets with Different Scales and Orientations

B. Texton, Brightness, and Colour Map

Now that the required filters are generated, the next focus would be generating the texton ($\mathcal{T}$), brightness ($\mathcal{B}$), and colour maps ($\mathcal{C}$). All these involve clustering sets of images based on different properties (response to filters, brightness, and colour). For the clustering, K-Means is used (64 clusters for texton, and 16 each for the latter two).

The results are shown in the array of figures from Fig. 6 to Fig. 15. For ($\mathcal{T}$), the test images are convoluted with different filter banks and all the responses are appended and then clustered. It is more straightforward in the case of $\mathcal{B}$ and $\mathcal{C}$ where the images are clustered as is (in the case of $\mathcal{B}$, the image is converted to grayscale before clustering).

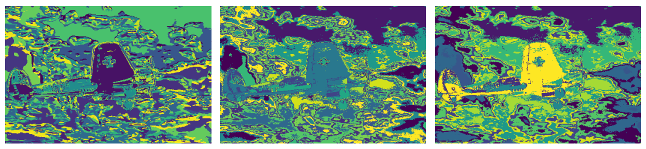

Fig. 6. $\mathcal{T}, \mathcal{B}, \mathcal{C}$ for Test Image 1

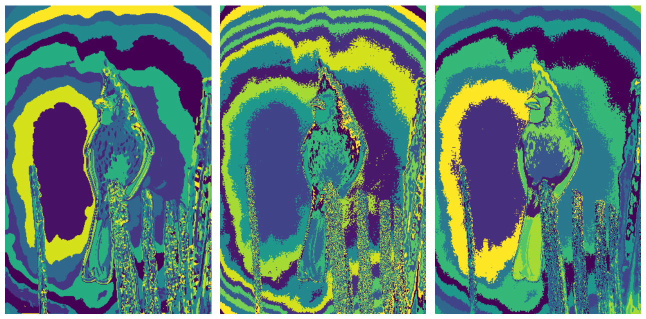

Fig. 7. $\mathcal{T}, \mathcal{B}, \mathcal{C}$ for Test Image 2

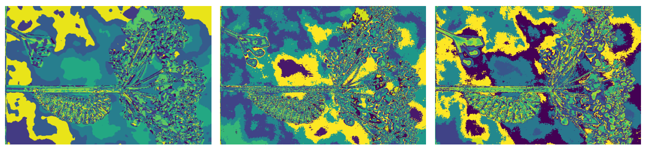

Fig. 8. $\mathcal{T}, \mathcal{B}, \mathcal{C}$ for Test Image 3

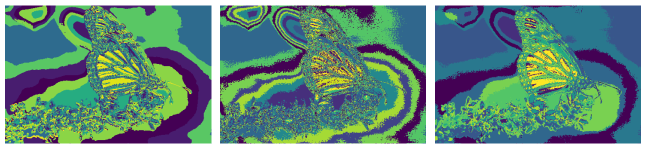

Fig. 9. $\mathcal{T}, \mathcal{B}, \mathcal{C}$ for Test Image 4

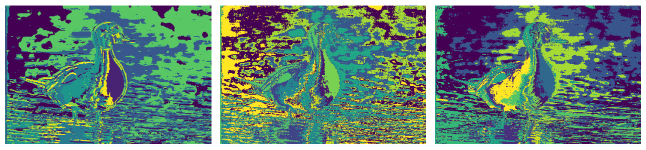

Fig. 10. $\mathcal{T}, \mathcal{B}, \mathcal{C}$ for Test Image 5

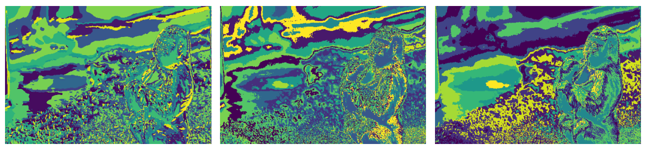

Fig. 11. $\mathcal{T}, \mathcal{B}, \mathcal{C}$ for Test Image 6

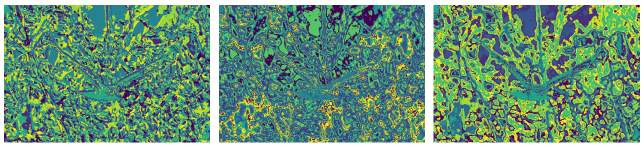

Fig. 12. $\mathcal{T},\mathcal{B},\mathcal{C}$ for Test Image 7

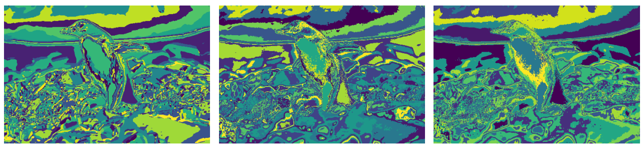

Fig. 13. $\mathcal{T},\mathcal{B},\mathcal{C}$ for Test Image 8

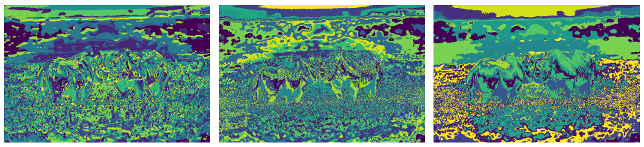

Fig. 14. $\mathcal{T},\mathcal{B},\mathcal{C}$ for Test Image 9

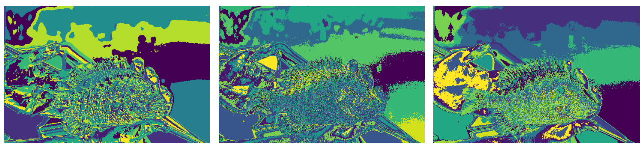

Fig. 15. $\mathcal{T},\mathcal{B},\mathcal{C}$ for Test Image 10

C. Texture, Brightness and Colour Gradients

In this section, the process used for obtaining texture, brightness, and colour gradients is discussed. Here is where the half-disk masks come into use. This process entails calculating the χ^2 distance in each of the maps, which captures the change in distribution of values - brightness for example - per pixel. It is a frequently used metric to compare two histograms, or in our case, **g** and **h**.

For the χ^2 distance, the histogram values are obtained by filtering the map with the left half and right half disks to obtain **g** and **h** respectively. It is quite efficient compared to using nested loops over pixel to calculate the same. The χ^2 distance is calculated as follows.

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

, where K is the number of clusters.

After the calculation, the average of this measure for each image is taken and stored. The results for Texture ($\mathcal{T}_g$), Brightness ($\mathcal{B}_g$) and Colour Gradients ($\mathcal{C}_g$) are shown below in figures Fig. 16 to Fig. 25.

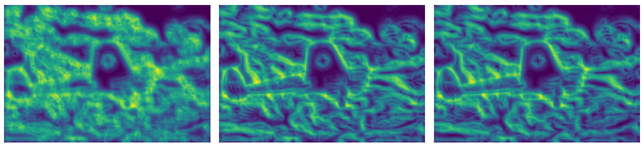

Fig. 16. $\mathcal{T}_g,\mathcal{B}_g,\mathcal{C}_g$ for Test Image 1

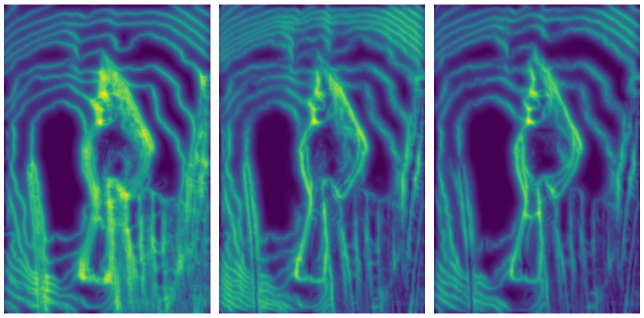

Fig. 17. $\mathcal{T}_g,\mathcal{B}_g,\mathcal{C}_g$ for Test Image 2

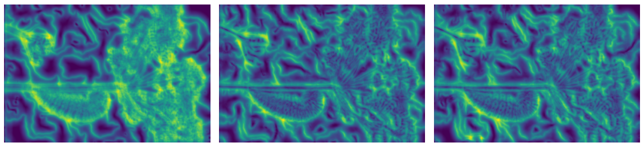

Fig. 18. $\mathcal{T}_g,\mathcal{B}_g,\mathcal{C}_g$ for Test Image 3

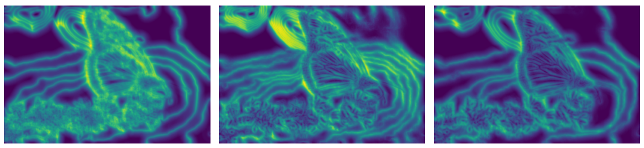

Fig. 19. $\mathcal{T}_g,\mathcal{B}_g,\mathcal{C}_g$ for Test Image 4

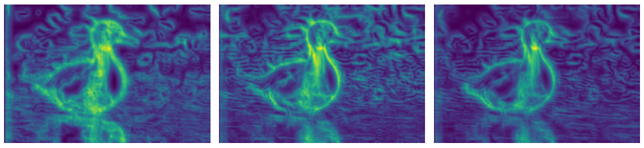

Fig. 20. $\mathcal{T}_g,\mathcal{B}_g,\mathcal{C}_g$ for Test Image 5

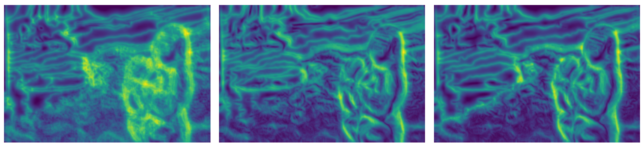

Fig. 21. $\mathcal{T}_g,\mathcal{B}_g,\mathcal{C}_g$ for Test Image 6

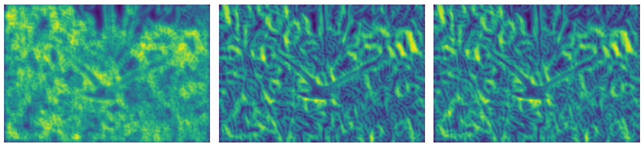

Fig. 22. $\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g$ for Test Image 7

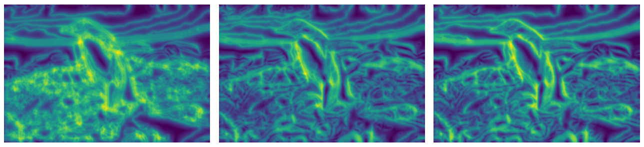

Fig. 23. $\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g$ for Test Image 8

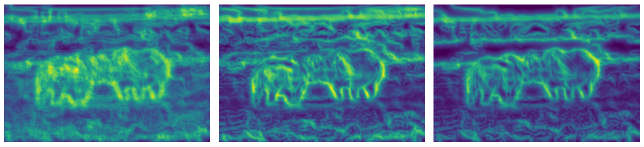

Fig. 24. $\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g$ for Test Image 9

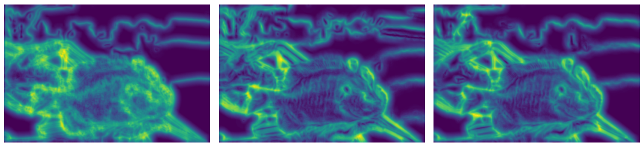

Fig. 25. $\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g$ for Test Image 10

D. Boundary Detection - Pb, Canny, and Sobel

With the sets of processed images in hand, the boundary detection can be performed using Pb-lite for comparison with the Canny and Sobel baselines. Before moving ahead with these algorithms, the ground truth images are provided in Fig. 26. Essentially this set would give an idea on the crucial features to be detected by these algorithms.

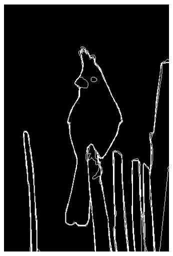

Fig. 26. Ground truth for Test Image 2

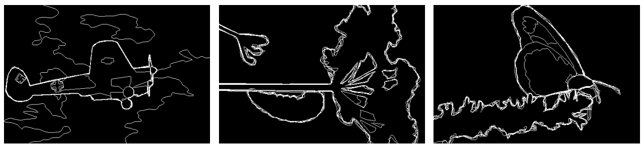

Fig. 27. Ground truth for Test Image 1, 3, 4

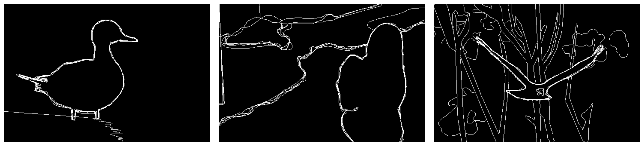

Fig. 28. Ground truth for Test Image 5, 6, 7

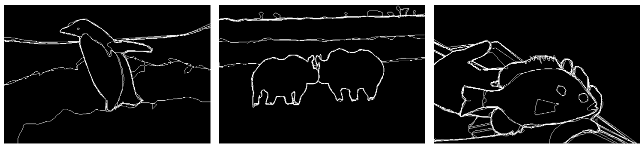

Fig. 29. Ground truth for Test Image 8, 9, 10

The results obtained from pb-lite is shown along with Canny and Sobel baseline results from Fig. 30 to Fig. 39. The confidence or probability of boundary at each pixel is calculated as follows based on the gradients obtained previously.

$$PbEdges = \frac{(\mathcal{T}_g + \mathcal{B}_g + \mathcal{C}_g)}{3} \odot (w1 * Canny + w2 * Sobel)$$

,where Canny and Sobel denotes the strength of boundaries conveyed by respective algorithm, $\odot$ denotes the Hadamard product operator and w1, w2 are the weights associated with them for Pb calculation.

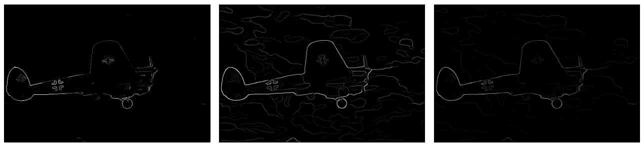

Fig. 30. Sobel, Canny, Pb for Test Image 1

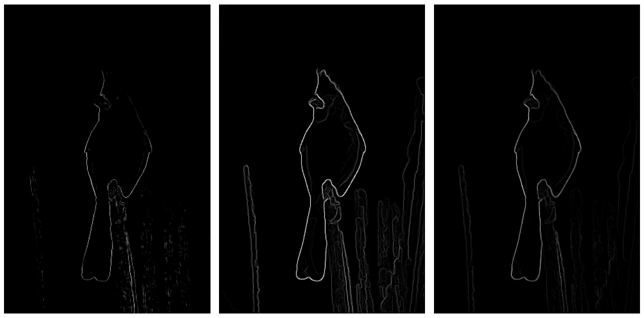

Fig. 31. Sobel, Canny, Pb for Test Image 2

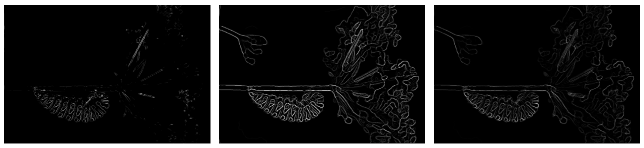

Fig. 32. Sobel, Canny, Pb for Test Image 3

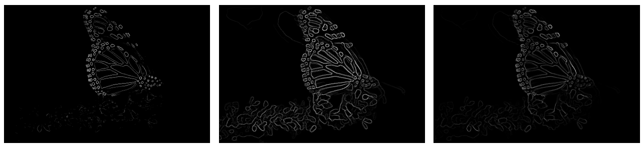

Fig. 33. Sobel, Canny, Pb for Test Image 4

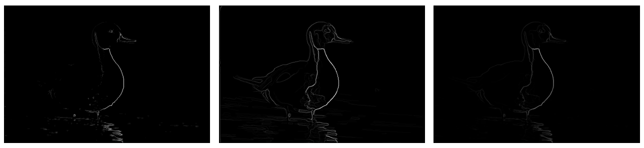

Fig. 34. Sobel, Canny, Pb for Test Image 5

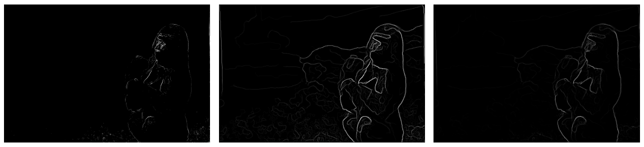

Fig. 35. Sobel, Canny, Pb for Test Image 6

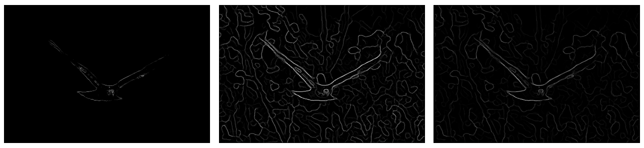

Fig. 36. Sobel, Canny, Pb for Test Image 7

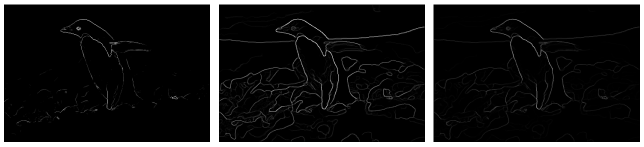

Fig. 37. Sobel, Canny, Pb for Test Image 8

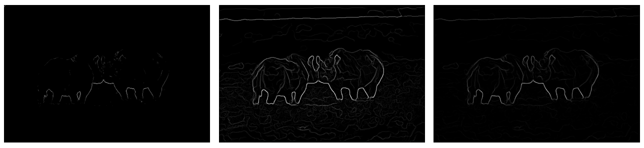

Fig. 38. Sobel, Canny, Pb for Test Image 9

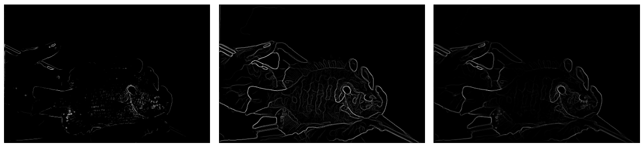

Fig. 39. Sobel, Canny, Pb for Test Image 10

E. Observations and Inference

On an overview, the Sobel seems to have performed worse compared to the other two while Canny provides the brightest boundary of the required features in the test images with the result of pb-lite not far off. Both Canny and pb-lite accentuates the required features in the test image but Canny seems to have outperformed pb-lite. Although one of the drawbacks observed in the results of Canny baselines, which is that it accentuate features that are not present in the ground truth, which also affects the output of pb-lite.

This performance difference could be leveled or even surpassed by pb-lite through one of the methods mentioned below, if not all.

1) Increase the weights given to Canny or make it dynamic, since equal, static weights to the baselines in the formula can only provide so much
2) Instead of taking the average of $\mathcal{T}_g, \mathcal{B}_g$, and $\mathcal{C}_g$, a preference could be given to one on a trial-based method by using weighted average to see which results in better performance
3) Changes in the filter scales or the filters, even, could result in marginal improvement in the performance as they directly correlated to generation of texton, brightness, and colour maps.

PHASE 2

For this section, the network trained on the CIFAR-10 dataset (50,000 images for train set and 10,000 images for test set) for image classification is discussed along with some results (Plots, Confusion Matrices). For starters, the networks used are the following.

- My Network (simple CNN)
- My Network - Modified
- ResNet
- ResNeXt
- DenseNet

A. My Network

This network is a basic convolutional neural network for classification of images consisting of convolutional layers and Maxpool for extracting various features from the input and downsampling, respectively. This output is then passed to fully connected layers that make a prediction on the class of image (array of probabilities).

Accordingly, the network has two pairs of convolutional layer and MaxPool after which there are pairs of fully connected layers and ReLU activation layer after each fully connected layer (named Linear). The architecture is depicted in Fig. 40 below. The small oval depicts ReLU. Also, the number of parameters for this network is 1,74,260.

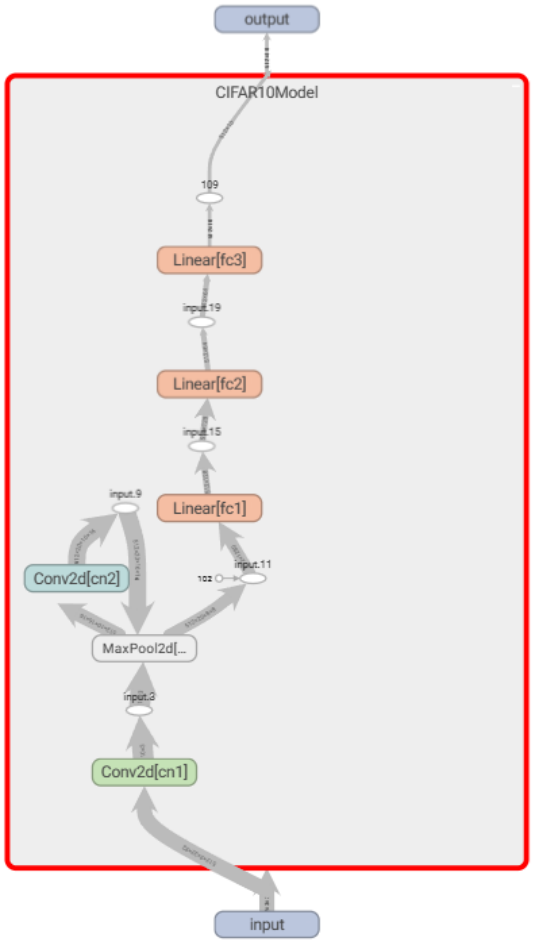

Fig. 40. Architecture of My Network

B. My Network - Modified

This network is quite similar to its predecessor along with some minor changes. This can be seen in the architecture shown in Fig. 41. To modify the current network, some tips used were the following.

- Added Batch Normalization between layers
- Increased batch size

First, the batch normalization layers (BatchNorm in the picture) is added after a convolutional layer and a fully connected layerThen, the minibatches were increased from 256 to 512. The number of parameters are 1,66,000. The results of these additions will be explored in the section of Analysis.

C. ResNet

An adaptation of the ResNet from the famous work is done here and is a simplified version of the same. The architecture of the implementation can be seen in Fig. 42. The added feature here compared to the previous networks is the ability of output from much previous layer to skip over the next layers and pass to a much further layer (depicted in Fig. 42). This allows to alleviate the issue of vanishing gradients in deep layers. The number of parameters in the implementation is 1,67,78,240.

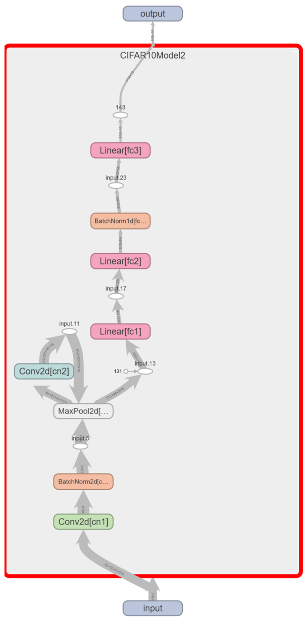

Fig. 41. Architecture of My Network - Modified

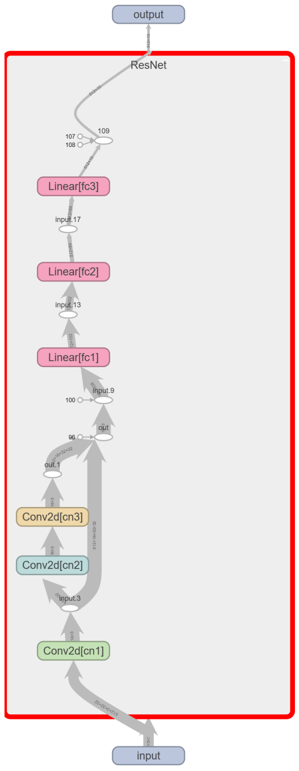

Fig. 42. Architecture of ResNet used

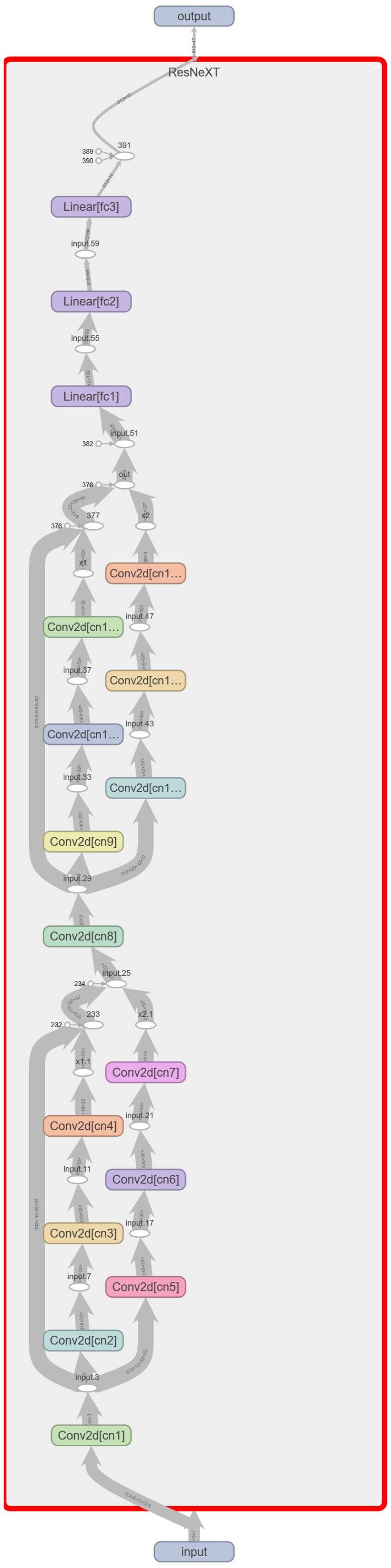

Fig. 43. Architecture of ResNeXt used

D. ResNeXt

In ResNeXt, another feature introduced to improve the performance is 'cardinality'. In simple terms, it is like a parallel network compared to the series of layers that was seen in the previous networks. The architecture of implemented ResNeXt is shown in Fig. 43. Also, the number of parameters are 29425.

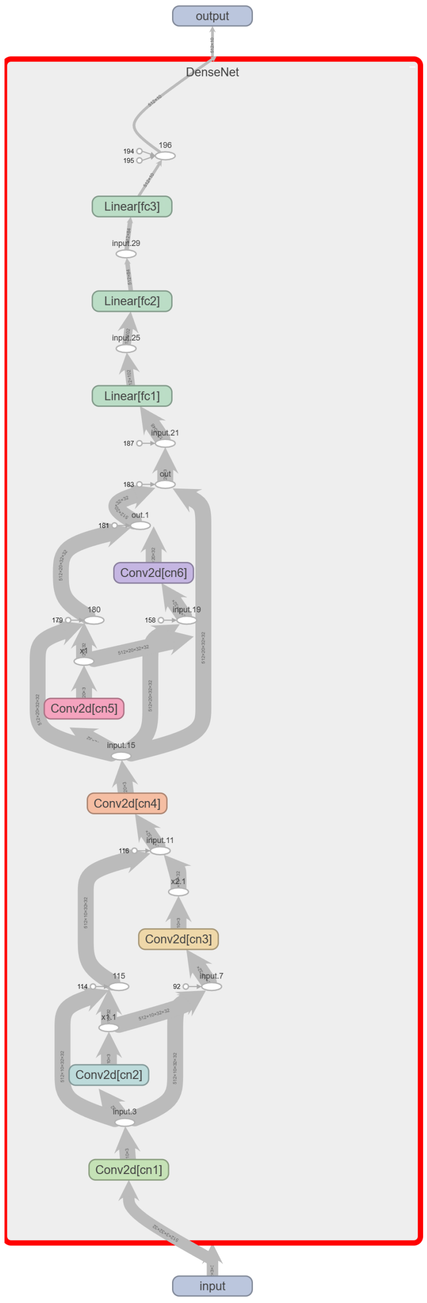

Fig. 44. Architecture of DenseNet used

E. DenseNet

DenseNet implementation is shown in Fig. 44 where the number of parameters are 2,10,69,804. Now that the description of all the used networks are given, the next step would be to analyse the results obtained during training and testing.

F. Observations and Inference

For all the networks, the optimizer used was Adam with a learning rate of 0.001. The kernel and stride in each of the convolutional layer were the same (3×3 kernel with stride of 1). To train the models, the number of epochs used were 25. Furthermore, Cross Entropy loss was used to train the network. The minibatches used for each of the network was 512, with an exception for MyNetwork which used 256.

The results of loss and accuracy are shown below for each of the networks from Fig. 45 to 54, with a downward trend for loss over time (epochs) and general upward trend of accuracy over time (epochs).

After these figures, an attempt is made to explain the performance difference between each of the networks along with confusion matrices for further clarification.

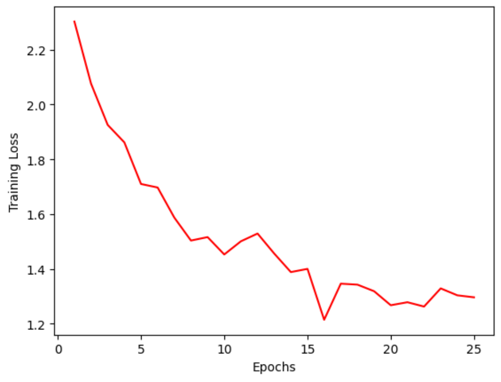

Fig. 45. Loss observed over Epochs during training of My Network

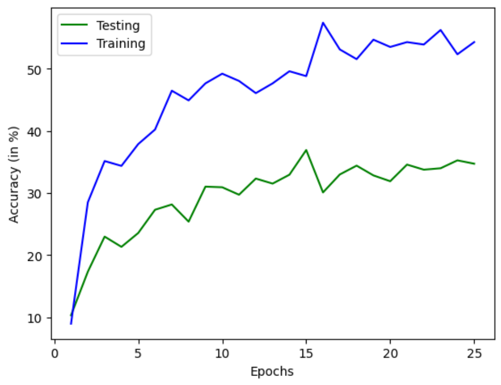

Fig. 46. Accuracy during training and Testing of My Network

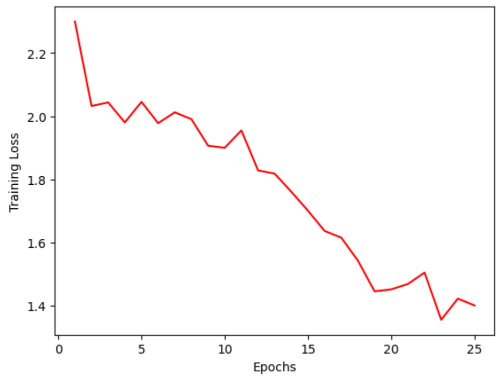

Fig. 47. Loss observed over Epochs during training of My Network - Modified

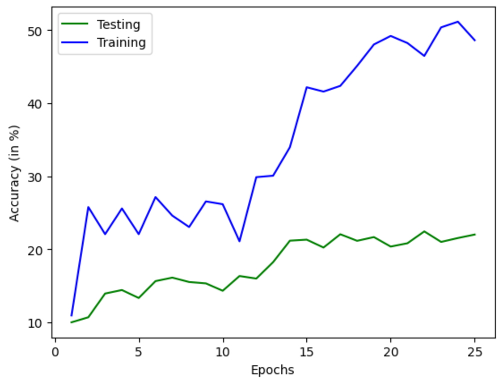

Fig. 48. Accuracy during training and Testing of My Network - Modified

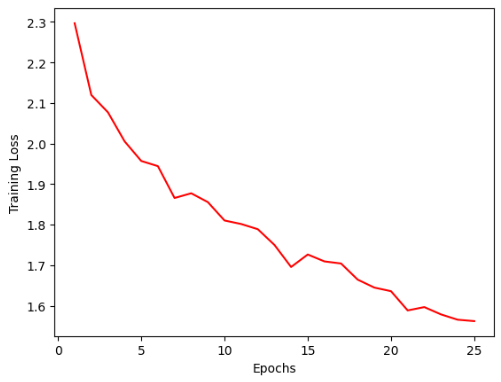

Fig. 49. Loss observed over Epochs during training of ResNet

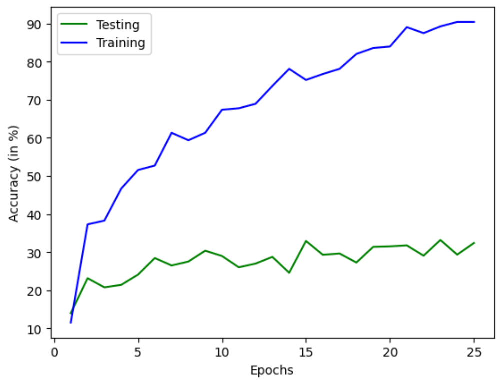

Fig. 50. Accuracy during training and Testing of ResNet

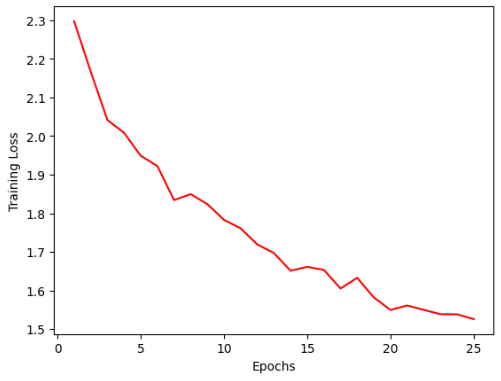

Fig. 51. Loss observed over Epochs during training of ResNeXt

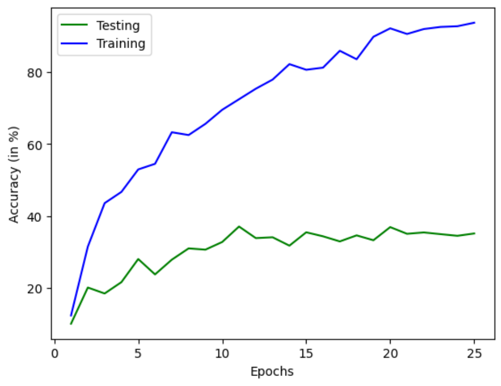

Fig. 52. Accuracy during training and Testing of ResNeXt

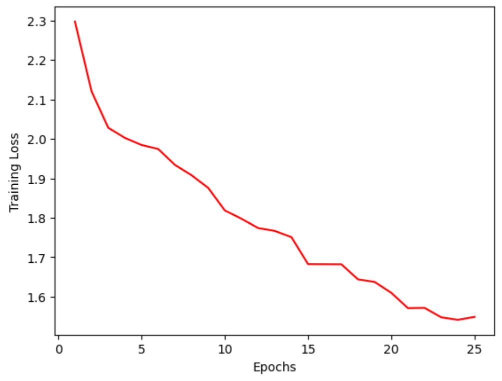

Fig. 53. Loss observed over Epochs during training of DenseNet

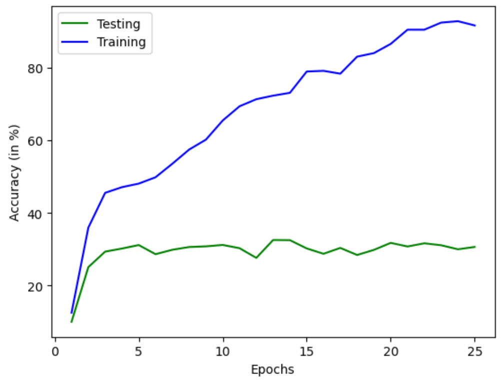

Fig. 54. Accuracy during training and Testing of DenseNet

As far as the losses are concerned, the metric recorded for the networks during testing can be seen to be similar over the networks. As far as accuracy is concerned, the recorded accuracy during testing for ResNeXt, ResNet, and DenseNet far exceeds the testing accuracy for MyNetwork and MyNetworkModified.

A conclusion cannot be formed outright with the same as the testing accuracy is overall similar with all the networks, with a marginal improvement seen in MyNetwork. This apparent and counterintuitive trend can be analysed with the help of other tools such as confusion matrix. The confusion matrices for all the networks generated during testing can be seen in the figures below from Fig. 55 to Fig. 59.

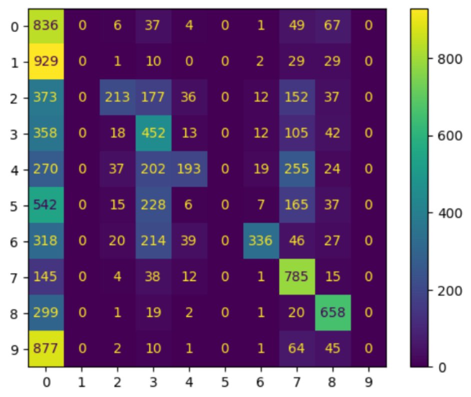

Fig. 55. Confusion matrix recorded during Testing of My Network

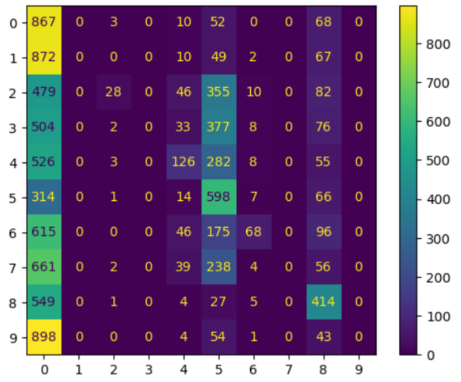

Fig. 56. Confusion matrix recorded during Testing of My Network - Modified

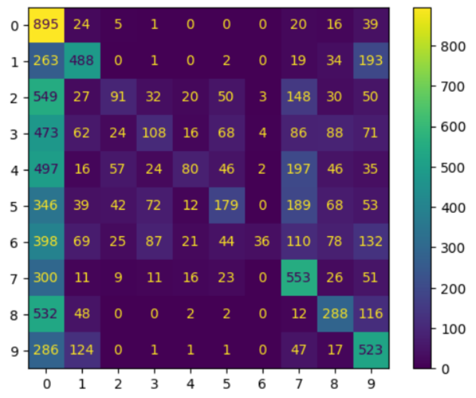

Fig. 57. Confusion matrix recorded during Testing of ResNet

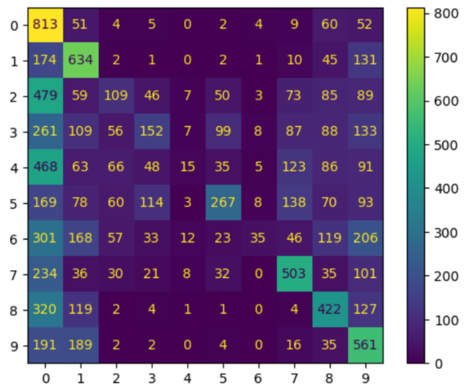

Fig. 58. Confusion matrix recorded during Testing of ResNeXt

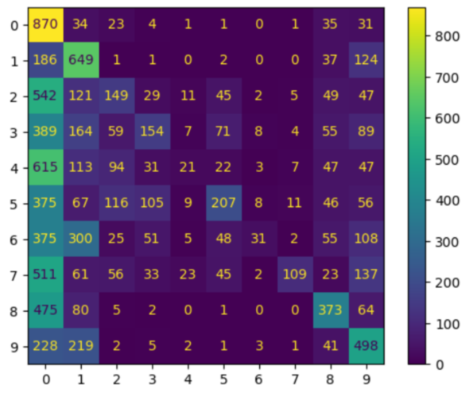

Fig. 59. Confusion matrix recorded during Testing of DenseNet

The label on the x-axis on the confusion matrices represent predicted labels (of each class as there are 10 classes of images) and the y-axis represents the true labels. The improvement trend seen in MyNetwork compared to other networks is easily thwarted by the confusion matrices.

The confusion matrix for MyNetwork show that the network predicts class 0 most cases and the better accuracy may be due to the improper distribution of dataset across classes. After introducing the changes, there was no improvement in the performance of the modified My Network apart from the faster training time. Between the two networks, My Network seem to fare worse in the case of predicting images in class 0-4 along with few more classes.

In the case of ResNet, the misclassifications seem to be spread out instead of concentration in a single class (but in all the network, most misclassifications is recorded in class 0 - predicting class 0 but the image belonged to another class). Finally, in the case of DenseNet and ResNeXt, better performance is recorded, as it can be seen with increase in prediction matches with true labels across most classes.

Supported along with the accuracy plots, it would be suffice to say that ResNeXt fared the best compared to the other networks. To improve the performance of these networks, the train set needs to be preprocessed and the distribution across classes need to be regulated.