Homework 0: Alohamora!

Abhinav Modi
Masters of Engineering in Robotics
University of Maryland, College Park
Email: abhi1625@umd.edu
USING 3 LATE DAYS

I. INTRODUCTION

Boundary detection and Image classification are two fundamental problems that are answered using image processing techniques. Thus, anyone looking to start learning in the field of Computer Vision needs to know about these problems and should attempt to learn different techniques required to solve the two. This document is divided into two phases in order to address the two problems with different techniques: Phase1-Probability of Boundary Technique for edge and boundary detection and Phase2-A deep learning approach for image classification on CIFAR-10 dataset.

II. PHASE 1: SHAKE MY BOUNDARY

In this section we perform boundary detection using a method called "PB-Lite". The outline of the process if shown in the figure below:
The first step in this process is to filter the image and find a texton map which is essentially the texture map of the image. This is computed by clustering the filter responses with K-Means clustering.

Filtering is used to access the low level features in the image. This helps us to measure and aggregate the regional texture, brightness and color properties. Different scales and orientations of a particular filter are used so that various types of textures can be addressed. Here, we have used three filter banks namely: Oriented DoG filters, Leung-Malik Filters and Gabor Filters.

A. Oriented DoG Filters

The Oriented Difference of Gaussian Filter is generated by taking the difference of two normal Gaussian Filters with same variance but the centers are of each is shifted by an amount equal to the standard deviation. This filter can also be created by convolving a simple Sobel Filter and a Gaussian kernel. The figure below shows the gaussian filter bank used for generating the results shown in the future sections.

The filter bank is generated by using 3 different scale values and 15 orientations for each scale, linearly-spaced from $0\,\mathrm{deg}$ to $360\,\mathrm{deg}$. Hence, the total number of filters is $3 * 15 = 45$.

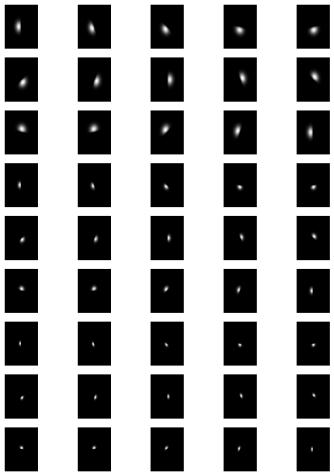

Fig. 1. DoG Filter Bank

B. Leung-Malik Filters

The Leung-Malik filter bank is a collection 48 filters with multiple scales and orientations. It consists of first and second order derivatives of Gaussians, Laplacian of Gaussian(LOG) filters and 4 Gaussian filters. All these filters account for different types of features in the image. The filter bank is shown in the figure (2)

C. Gabor Filters

Gabor filters are inspired based on the way human visual system. A Gabor filter is generated by modulating a gaussian kernel with a sinusoidal plane wave. This is a linear filter that basically analyses if there is any specific frequency(governed by λ) content in the image in specific directions around the point of interest. The Gabor filterbank used in this project is shown in the figure(3) The filter bank with is generated for λ=1 with three scales:[9,16,25]. Also, for each scale value, 15 filters with different orientations, uniformly spaced from 0 to 360 degrees are generated.

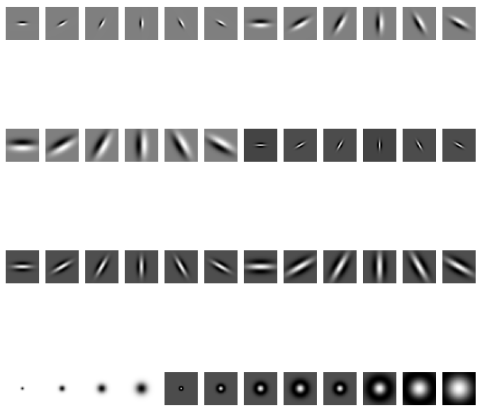

Fig. 2. LM filter bank

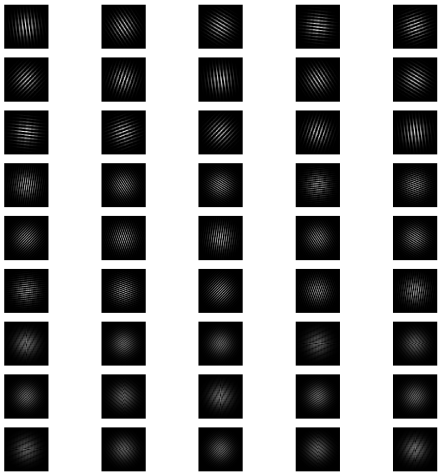

Fig. 3. Gabor filter bank

D. Texton Map $\mathcal{T}$

Once the filtering process of the image is complete, we end up with a stack of images of size $m \times n \times N$, where m, n are the dimensions of the image and N is the total number of filters used. Thus, each pixel value can now be represented as a distribution of these N values. Each distribution is then represented by a unique Texton-ID. These different distributions for all the pixels are then clustered into K textons using K-Means. This generates an image which captures the texture changes in the original image. Texton Maps for all the test set images be seen in the following figures:

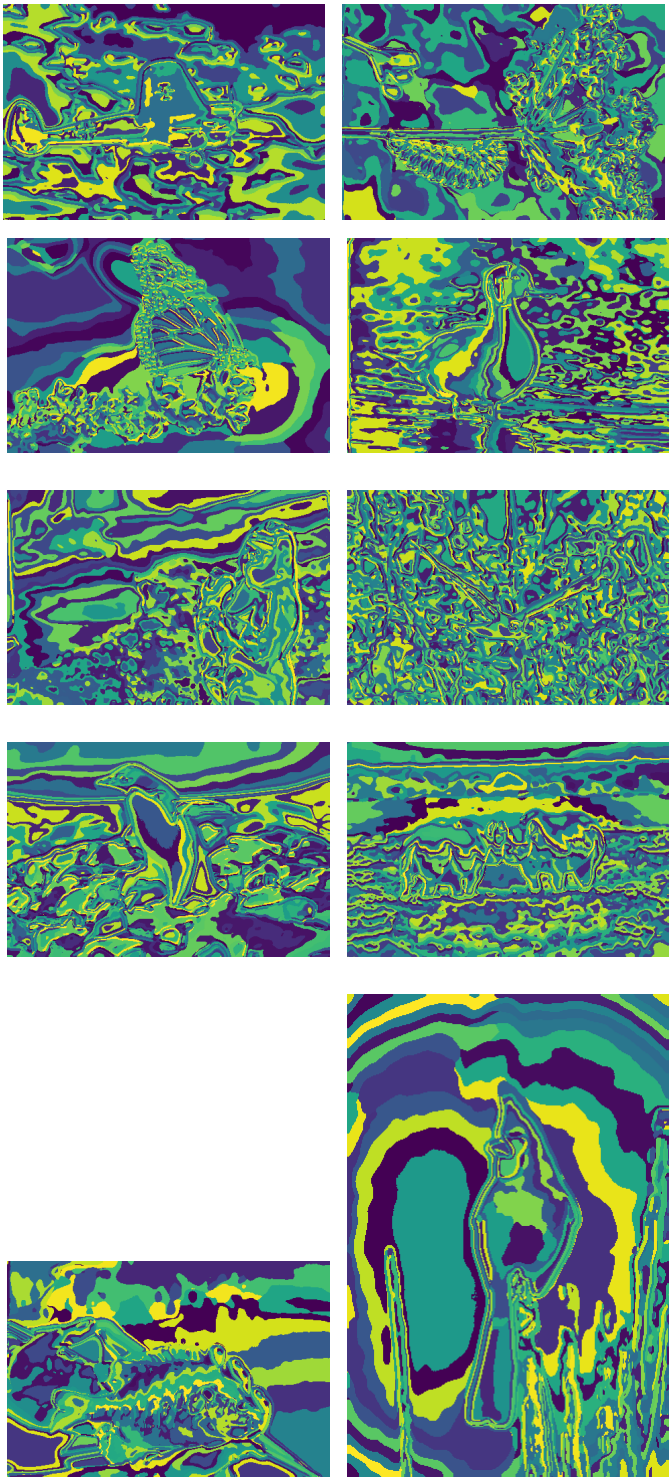

Fig. 4. Texton Maps of the 10 test images

E. Brightness Map $\mathcal{B}$

The Brightness map captures the changes in intensity of light in the image. Similar to Texton map generation the K-Means clustering of the grayscale image is performed for

K=16 and the output can be seen below:

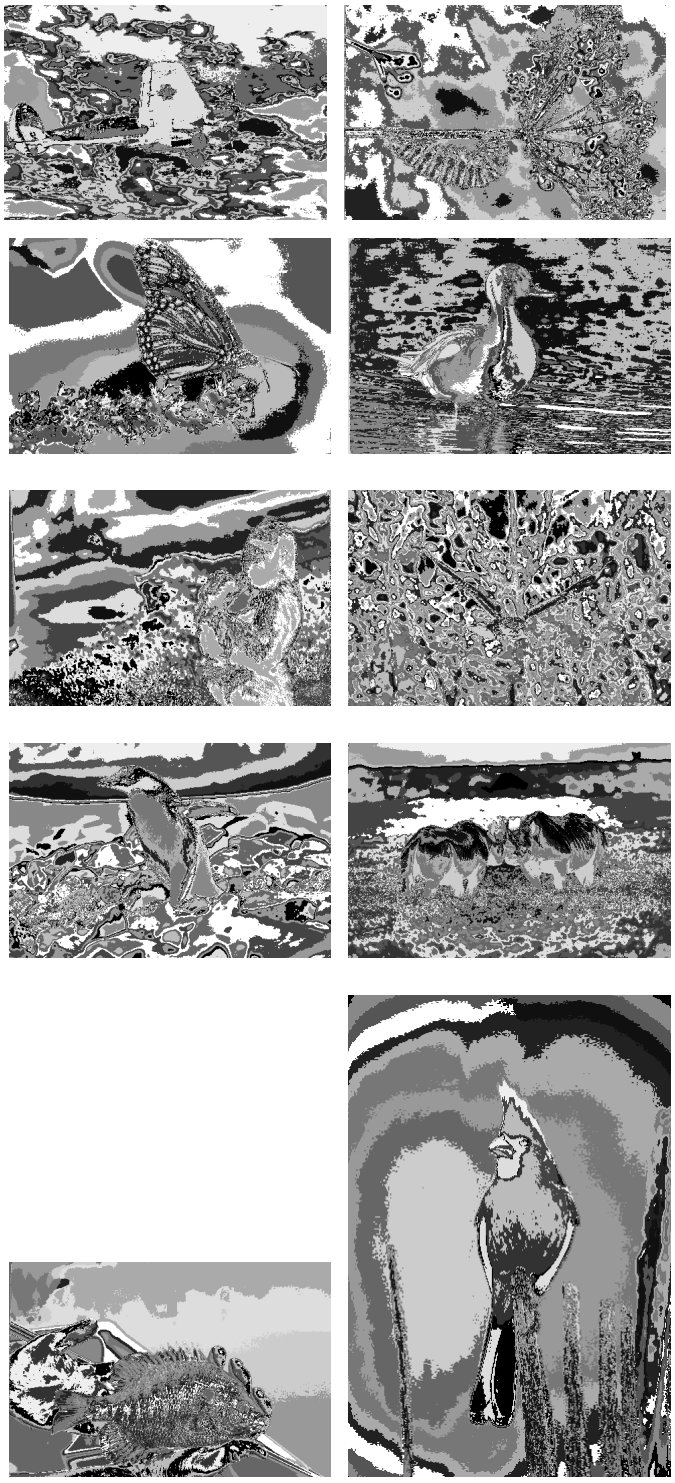

Fig. 5. Brightness Maps of the 10 test images

F. Color Map $\mathcal{C}$

The Color map captures the changes in color/ chrominance in the image. The color values were clustered using K-Means clustering into 16 clusters. This generates an output image which can be seen below:

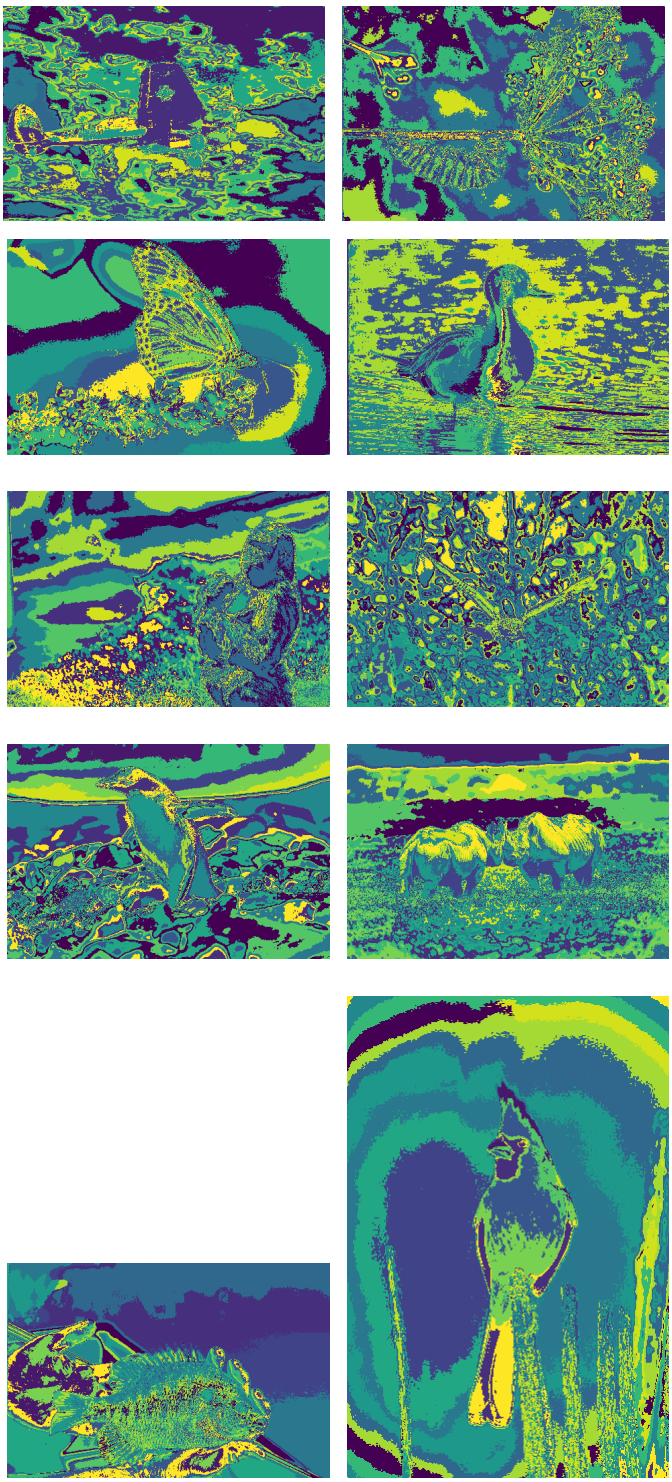

Fig. 6. Color Maps of the 10 test images

G. Gradient Maps

The Maps generated above are used to calculate gradient maps $\mathcal{T}_\}$,$\mathcal{B}_\}$ and $\mathcal{C}_\}$. These maps encode the texture, brightness and color distributions changing at each pixel. These are generated by comparing the values at each pixel by convolving the image with a left/right half-disc pair centered at the pixel. The basic concept behind this is that if the values are similar the gradient should be small and if the values are dissimilar, the gradient will be large.

The half-disks are generated by multiplying an array of size equal to the radius/scale of the circular disk with all values which lie inside this radius equal to 1 and rest 0, with an array of equal size but where one half of the array is 0s and the other half consists of 1s. This multiplication results in a half-disk which can be rotated to produce the desired half-disk mask.

Here if you rotate the disk after you've multiplied the two arrays will result in pixel voids. This can be avoided by rotating the rectangular block matrix of 0s and 1s and then by applying a "logical OR" operator on them. This gives the required half-disk masks which are shown below:

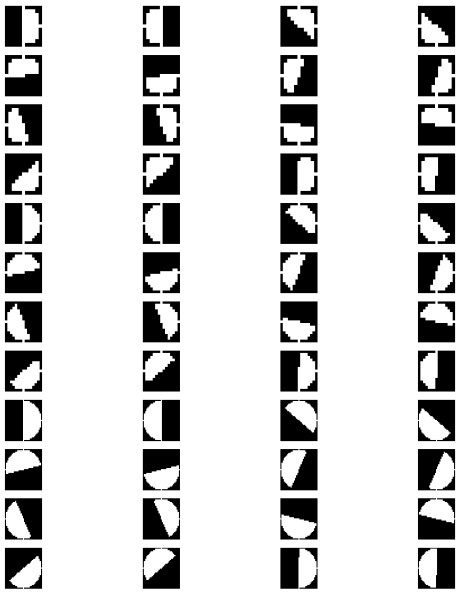

Fig. 7. Half-Disk Masks for scales=[5,7,16]

Using the above generated Half-Disk masks we compute the $\mathcal{T}_\}, \mathcal{B}_\}$ and $\mathcal{C}_\}$ maps by comparing the distributions generated using each half-disk pair with a χ^2 measure. The binning scheme is defined for $\mathcal{K}$ indexes which is equal to the number of K-Means clusters for each $mathcalT, mathcalB, mathcalC$. This procedure is repeated for all the half-disk pairs to generate a 3D matrix of size $m \times n \times N$ where m, n are the dimensions of the image and N is the number of filters.

The output of the mean of each $\mathcal{T}_\}, \mathcal{B}_\}$ and $\mathcal{C}_\}$ is given as follows:

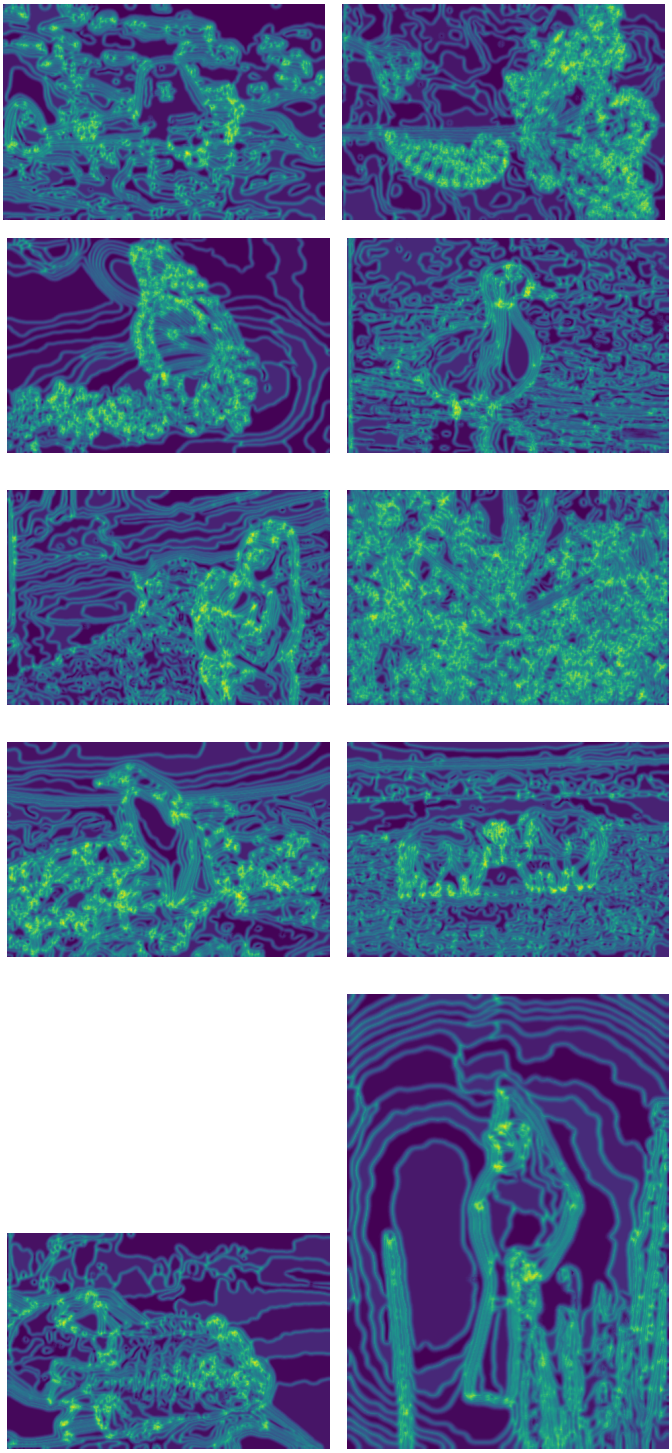

Fig. 8. Texton Gradients-Tg of the 10 test images

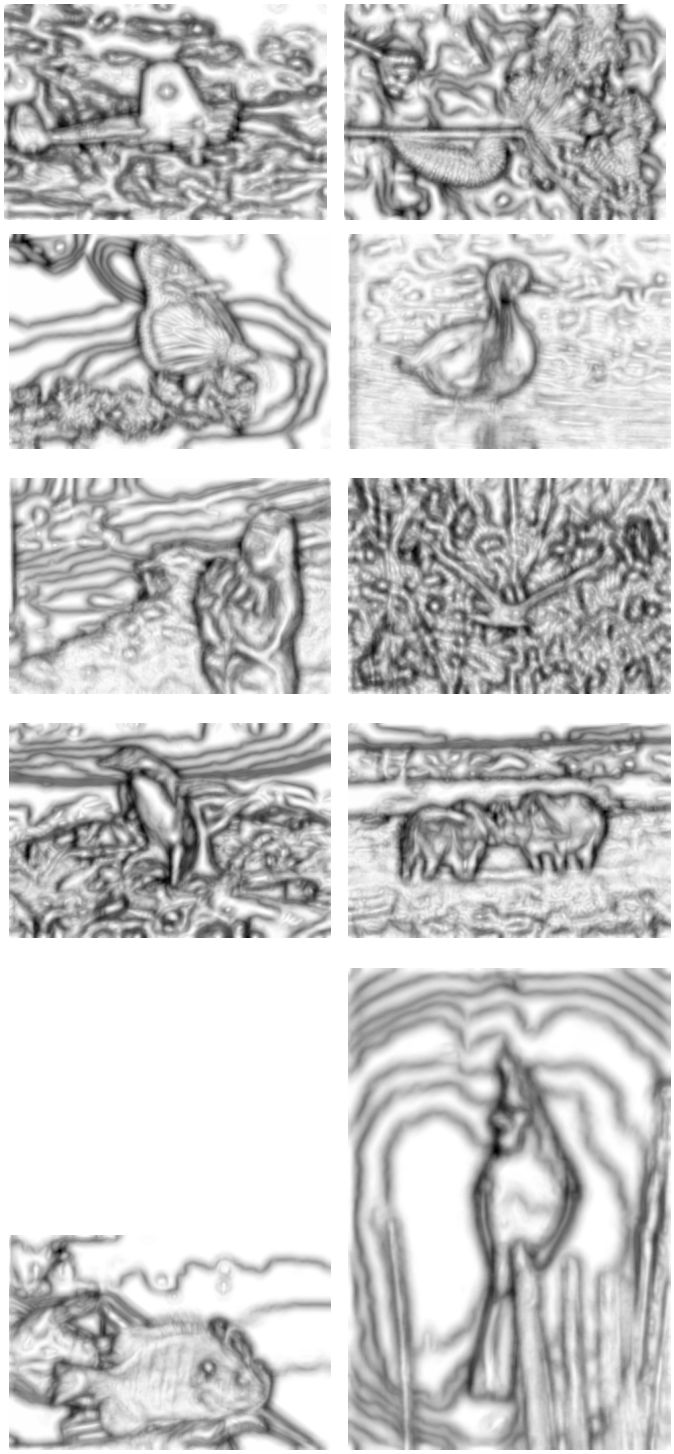

Fig. 9. Brightness Gradients-Bg of the 10 test images

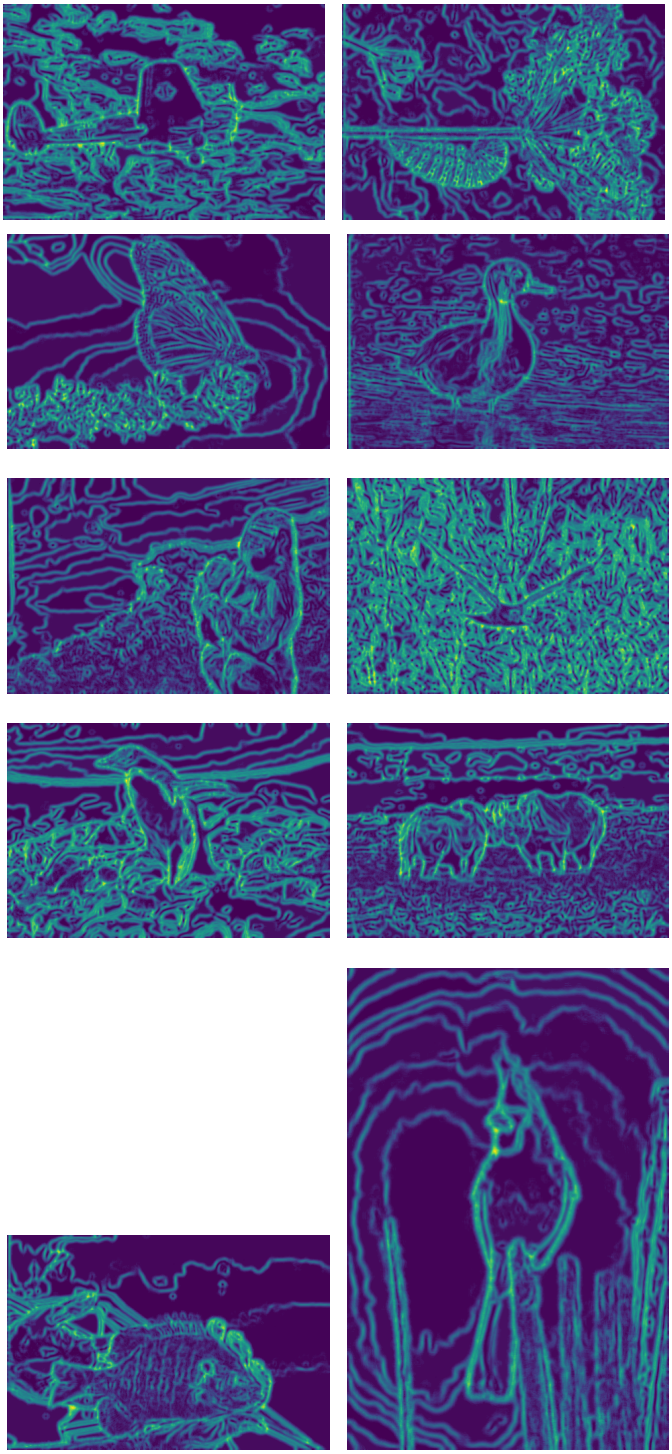

Fig. 10. Color Gradients-Cg of the 10 test images

H. PB-Lite Output

Finally, the gradient maps generated are combined with Canny and Sobel baselines using the equation:

$$PbEdges = \frac{(\mathcal{T}_i + \mathcal{B}_i + \mathcal{C}_i)}{3} \odot (w_1 * cannyPb + w_2 * sobelPb) \quad (1)$$

The $\odot$ is the Hadamard operator which is the element-wise multiplicatin of the arrays in the equation. The choice of the weights w_1 and w_2 is based on the canny and sobel baselines and the features we want from each baseline. The only constraint is that $w_1 + w_2 = 1$. The Canny and Sobel

outputs and the resulting Pb-lite outputs are shown in the figures below:

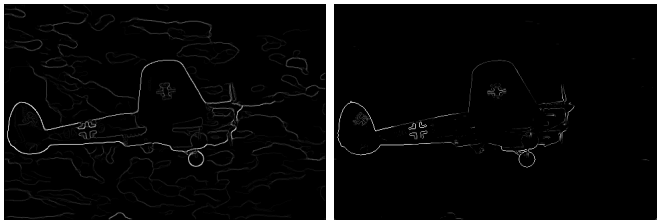

Fig. 11. (a)Canny baseline of image 1, (b)Sobel baseline of image 1

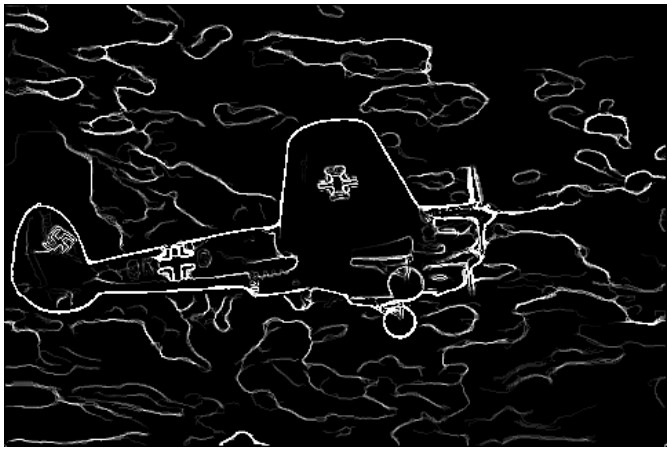

Fig. 12. (c)PB-Lite output of image 1

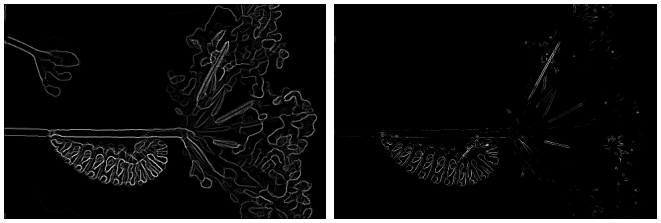

Fig. 13. (a)Canny baseline of image 2, (b)Sobel baseline of image 2

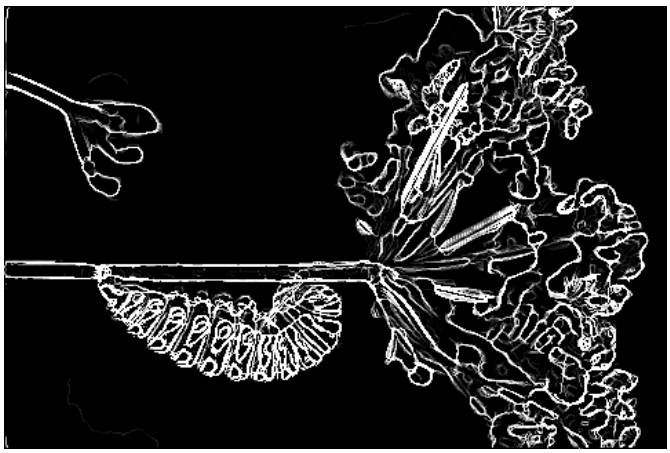

Fig. 14. (c)PB-Lite output of the image 2

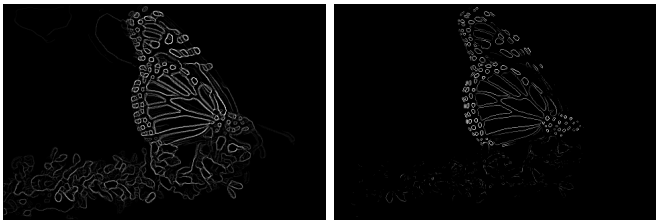

Fig. 15. (a)Canny baseline of image 3, (b)Sobel baseline of image 3

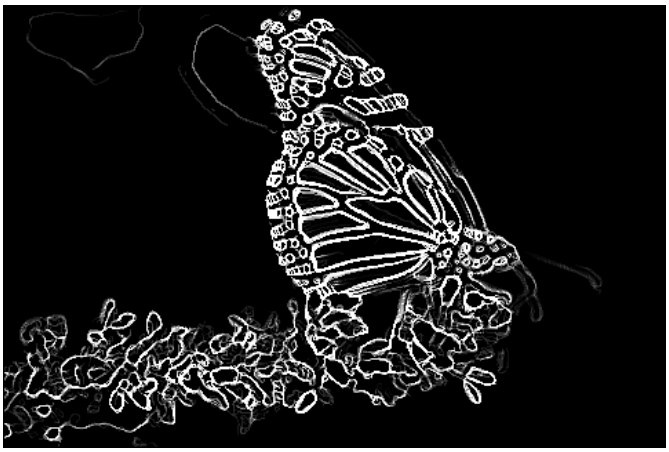

Fig. 16. (c)PB-Lite output of the image 3

Fig. 17. (a)Canny baseline of image 4, (b)Sobel baseline of image 4

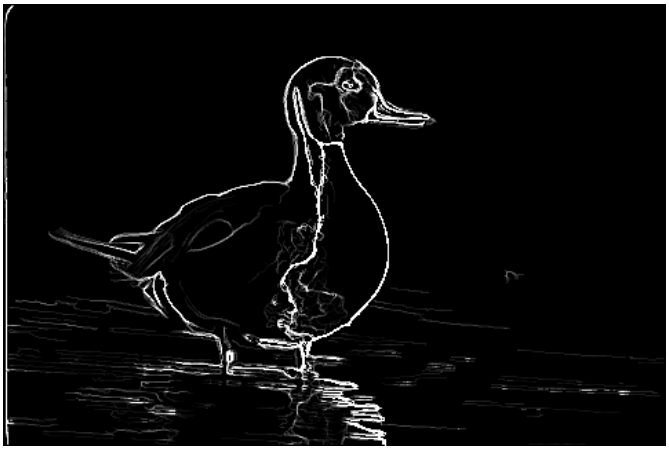

Fig. 18. (c)PB-Lite output of the image 4

Fig. 19. (a)Canny baseline of image 5, (b)Sobel baseline of image 5

Fig. 20. (c)PB-Lite output of the image 5

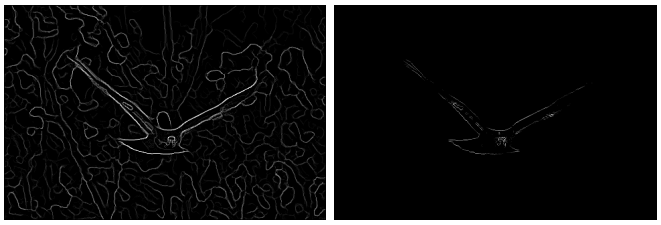

Fig. 21. (a)Canny baseline of image 6, (b)Sobel baseline of image 6

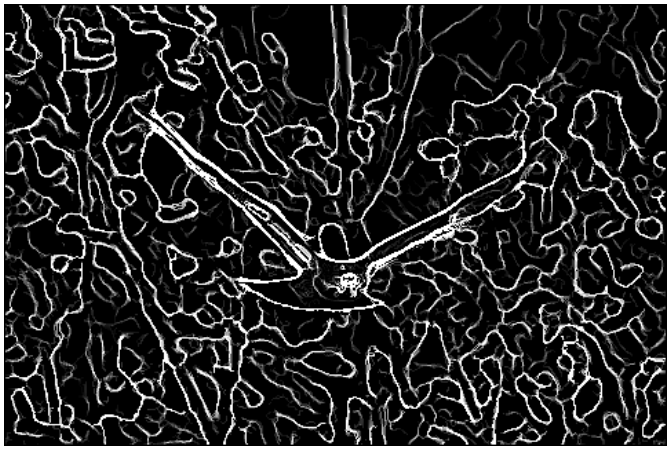

Fig. 22. (c)PB-Lite output of the image 6

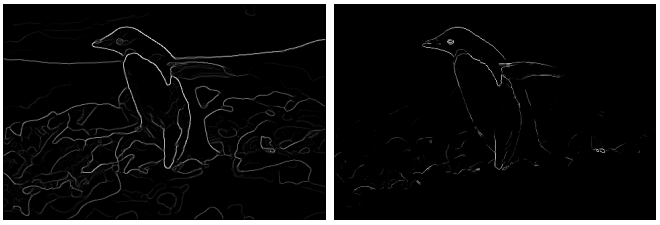

Fig. 23. (a)Canny baseline of image 7, (b)Sobel baseline of image 7

Fig. 24. (c)PB-Lite output of the image 7

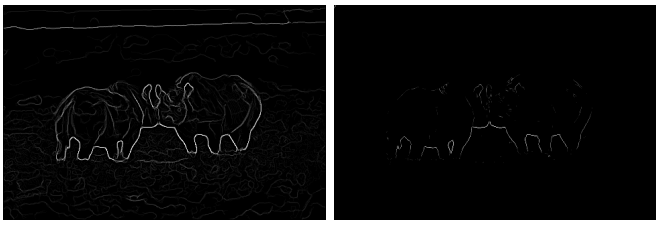

Fig. 25. (a)Canny baseline of image 8, (b)Sobel baseline of image 8

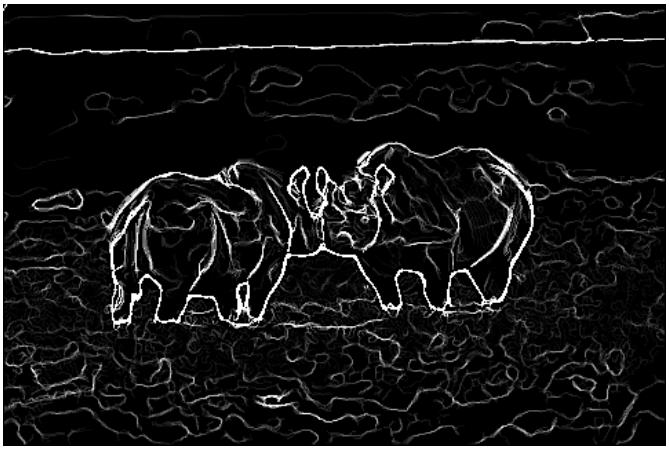

Fig. 26. (c)PB-Lite output of the image 8

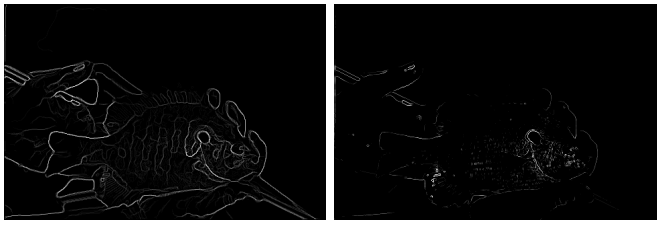

Fig. 27. (a)Canny baseline of image 9, (b)Sobel baseline of image 9

Fig. 28. (a)Canny baseline of image 9, (b)Sobel baseline of image 9

I. Discussion and Conclusion: Phase 1

While computing the PB-Lite output it was observed that changing just the weights of the canny and sobel baselines produced significant changes in the output image which can be seen in the two images below:

Fig. 31. PB-Lite output for different weights for the Canny baseline

This happens due to the fact that Canny baseline also includes weaker unwanted edges due to the averaging operation performed while calculating the canny baseline.

For the scope of this project, these weights for canny and sobel are calculated by trial and error. But the original paper mentions about an optimization algorithm to calculate these weights. Moreover, in the output of PB-Lite, there are still more textures left. The fearures are weak but can be removed by changing the scales and orientations of the filter banks used.This can help produce better results.

Fig. 29. (a)Canny baseline of image 10, (b)Sobel baseline of image 10

Fig. 30. (c)PB-Lite output of the image 10

III. PHASE 2:A DIVE ON DEEP LEARNING

A. Convolutional Neural Network

This is an initial deep learning model. It is a simple neural network with 4 convolution layers followed by 2 fully connected layers and a softmax output layer. The architecture is shown in fig(32) This architecture did not use any data augmentation while training and also no standardization technique was used. The network reached about 80% training accuracy in about 45 epochs as can be seen in the fig(33). The important thing to note here is that even though the training accuracy is pretty high, the test accuracy reaches a maximum of 67.33% for the same number of epochs as the train set, fig(34) and fig(35)

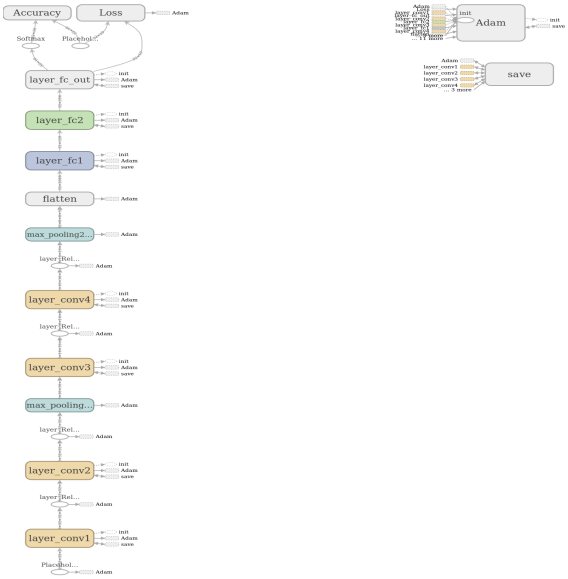

Fig. 32. Alex Net - Simple CNN architecture

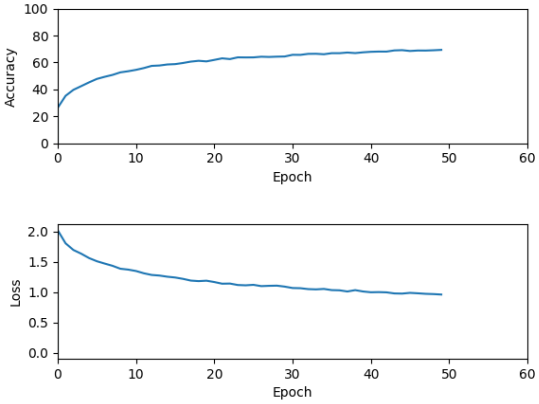

Fig. 33. Training and Loss vs Epochs for Alex Net(CNN)

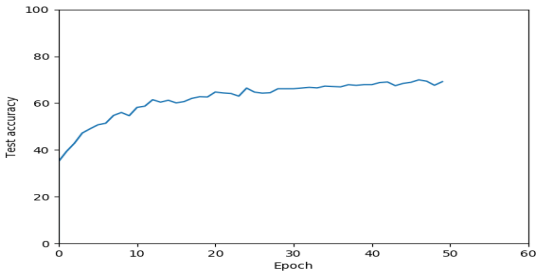

Fig. 34. Test accuracy vs Epochs for Alex Net(CNN)

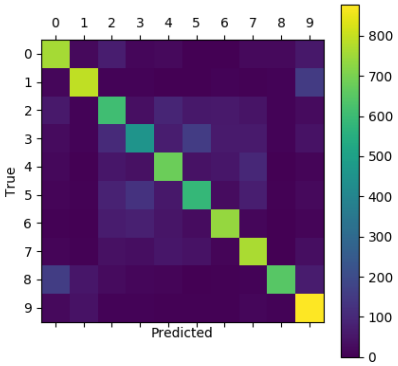

Fig. 35. Confusion Matrix for test results for Alex Net

Parameter	Value
Batch Size	32
Learning Rate	0.001
Optimizer	Adam
Number of Epochs	50
Max test Accuracy(last epoch)	69.03%
No. of Parameters	428426
Inference time(s)	0.087596s

TABLE I
PARAMETERS FOR THE DEEP NETWORK USED IN SECTION I

1) Improving accuracy: The previous network gives acceptable results but its accuracy can be improved by slightly tweaking the architecture. We first standardize the dataset within values [-1,1]. Next we also apply data augmentation techniques wherein we do random noise addition as well random left and right image rotation. The final touch is adding batch normalization layers after each convolution layer. By doing this we force the input of every layer to have approximately the same distribution in every training step. This prevents the system from the internal

covariate shift problem and decrease training time. This is shown in Fig(36). We get considerable improvement in the test accuracy as the accuracy improves over the previous model with a maximum of 75.65%. The model statistics are shown in figs(37), (38) and (35)

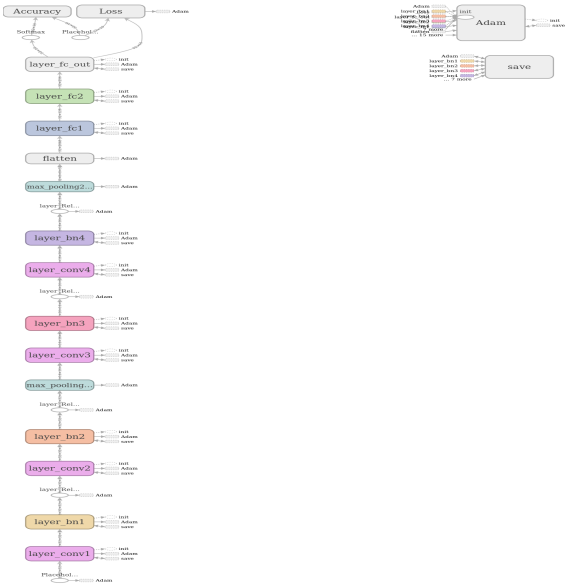

Fig. 36. Alex Net - Simple CNN architecture with batch normalization layers

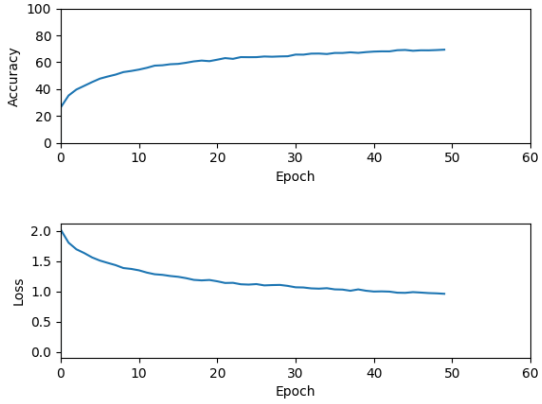

Fig. 37. Training and Loss vs Epochs for Alex Net(CNN) with batch normalization layers and data augmentation

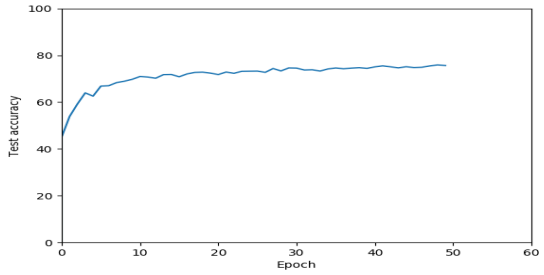

Fig. 38. Test accuracy vs Epochs for Alex Net(CNN) with batch normalization layers and data augmentation

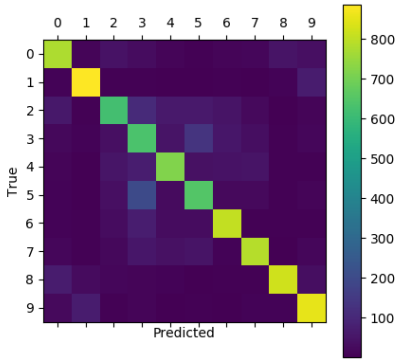

Fig. 39. Confusion Matrix for test results for Alex Net with batch normalization layers and data augmentation

Parameter	Value
Batch Size	32
Learning Rate	0.001
Optimizer	Adam
Number of Epochs	50
Max test Accuracy(last epoch)	75.65%
No. of Parameters	428746
Inference time(s)	0.08945s

TABLE II
PARAMETERS AND RESULTS FOR ALEX NET WITH BATCH NORMALIZATION AND DATA AUGMENTATION

B. Residual Neural Network(ResNet)

Keeping the standardization and data augmentation as it is we implement the ResNet architecture. A Residual Network, or ResNet is a neural network architecture which solves the problem of vanishing gradients in the simplest way possible, i.e., by applying skip connections in a general residual block as shown in fig(40). This allows the network to accommodate deep layers without having the vanishing gradient problem. This network was only trained for 25 epochs and the training and test set accuracy vs epochs can be seen in the fig(41) and (42). Only 3 residual layers with 2 convolution layers each were used in this ResNet network. As you can see the training accuracy reaches about 80% for only 25 epochs and the test

set accuracy reaches to around 58% for such a short duration of training. This can also be inferred from the rate of decrease of loss per epoch.

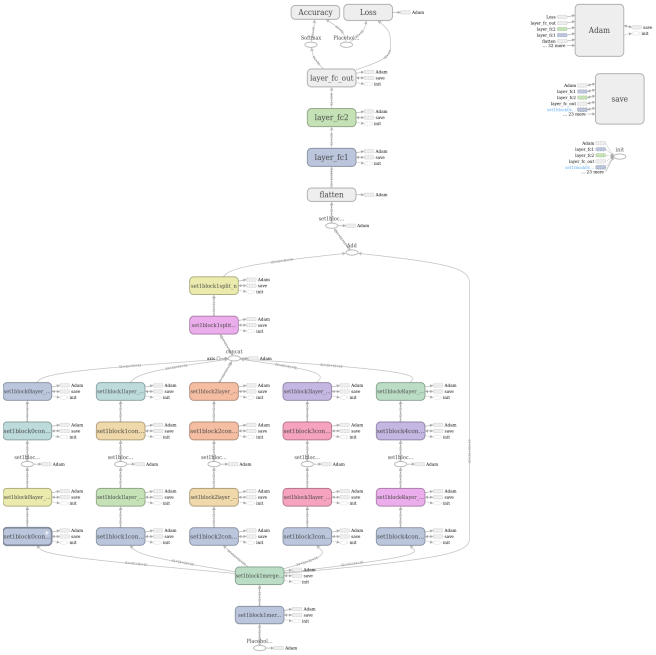

Fig. 40. ResNext architecture with skip connection and cardinal layers

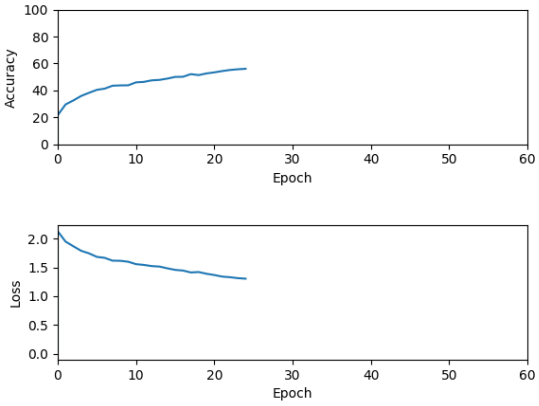

Fig. 41. Training and Loss vs Epochs for ResNext

C. ResNext

ResNext is a recent improvement on the ResNet architecture. In addition to utilizing the concept of residual learning framework from ResNet, the concept of **"Cardinality"** is introduced in this network. Cardinality is the size of the set of split transformations an input goes through before to is passed on to the fully connected layers, as shown in fig(44). In ResNext architecture, the input is split into different paths(number of split paths is equal to the cardinality) and convolutions are performed in each of these split paths. The outputs of these split layers is then concatenated and added to the input itself, followed by application of non-linearity. It is empirically shown in the paper that even under the restricted condition of maintaining complexity, increasing cardinality is able to improve classification accuracy. Moreover, increasing cardinality is more effective than going deeper or wider when we increase the complexity of the network.

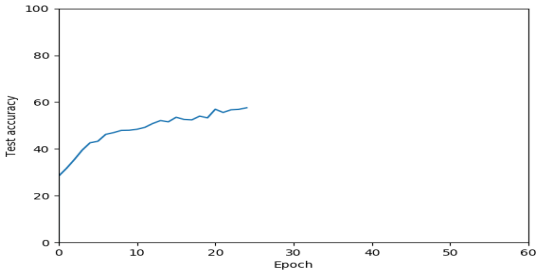

Fig. 42. Test accuracy vs Epochs for ResNext

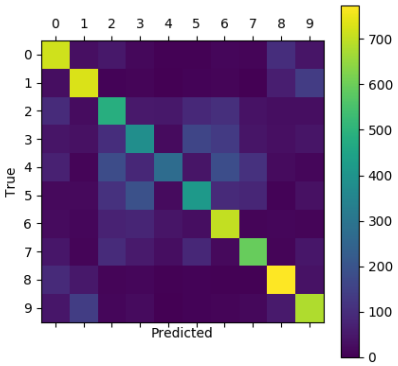

Fig. 43. Confusion Matrix for test results for ResNext

Parameter	Value
Batch Size	32
Learning Rate	0.001
Optimizer	Adam
Number of Epochs	25
Max test Accuracy(last epoch)	57.5%
No. of Parameters	2333002
Inference time(s)	0.10937s

TABLE III
PARAMETERS AND RESULTS FOR RESNET

For this project ResNext architecture implemented is a single residual layer with a cardinality of 5. Each cardinal(split) layer has 2 convolution layers. The outputs from these layers are concatenated and then passed through another convolution layer. Finally the output is added to the original input and then passed through an activation layer(ReLu) to the fully connected layers. The training and test accuracy results can be seen in the figures(45), (46) and (47)

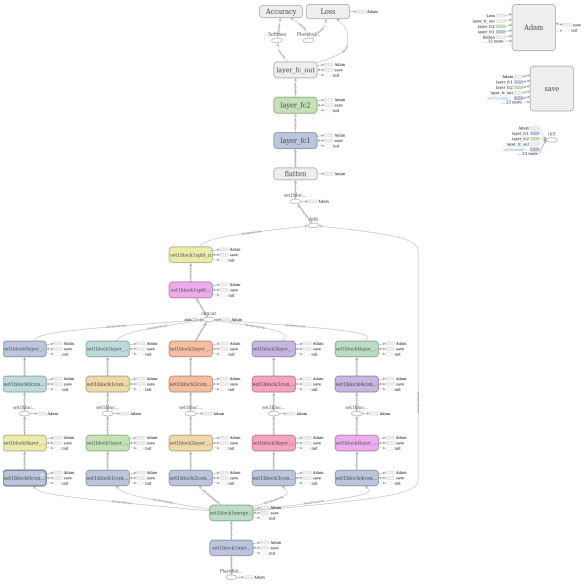

Fig. 44. ResNext architecture with skip connection and cardinal layer

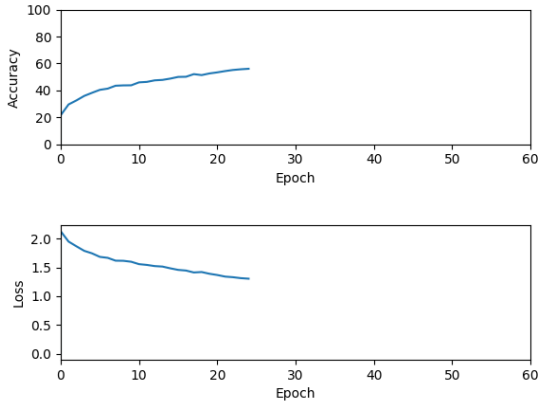

Fig. 45. Training and Loss vs Epochs for ResNext

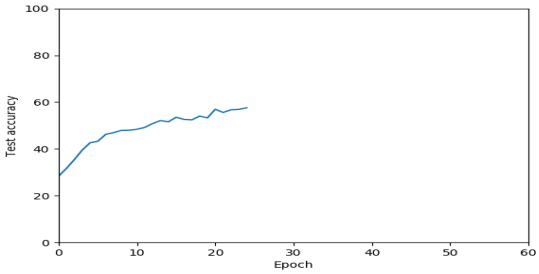

Fig. 46. Test accuracy vs Epochs for ResNext

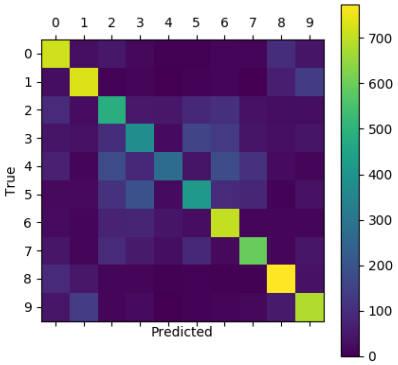

Fig. 47. Confusion Matrix for test results for ResNext

Parameter	Value
Batch Size	32
Learning Rate	0.001
Optimizer	Adam
Number of Epochs	50
Max test Accuracy(last epoch)	57.57%
No. of Parameters	2222442
Inference time(s)	0.12734s

TABLE IV
PARAMETERS AND RESULTS FOR RESNEXT

D. DenseNet

In the previous sections we have seen that that the accuracy and efficiency of the CNNs can be substantially improved by making shorter connections between layers close to the input and those close to the output. In DenseNet each layer connects to every other layer in a feed-forward fashion. This can be seen in the fig(48) This solves the vanishing gradient problem, allows stronger feature propagation and reuse. This leads to a decrease in the number of features without compromising on training and test accuracy.

The DenseNet architecture implemented for this project contains 4 convolution layers in the dense block. The input image is first convoluted and the output is feeded to all the convolution layers in succession. This process is repeated

for all the convolution layers. The output is then finally concatenated and through an activation layer it is sent on to the fully connected layers. The loss over epochs, training and test accuracy over epochs can be seen in the figs(51) and (**??**)

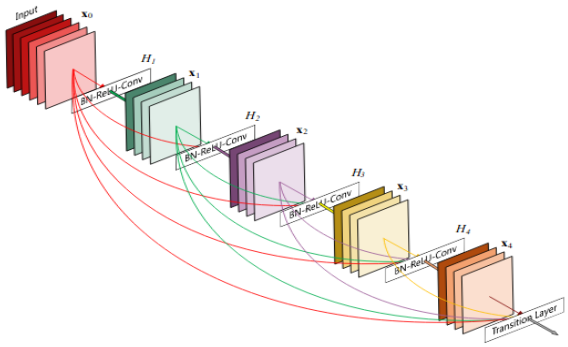

Fig. 48. Flow of the input inside the Dense block

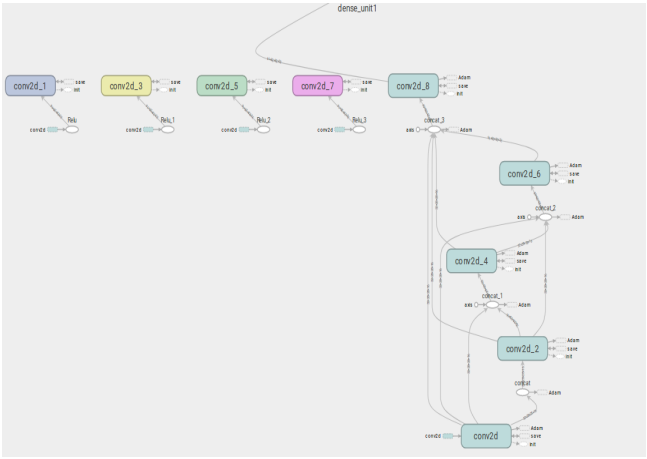

Fig. 49. Flow of the input inside the Dense block as implemented

Parameter	**Value**
Batch Size	32
Learning Rate	0.001
Optimizer	Adam
Number of Epochs	50
Max test Accuracy(last epoch)	46.36%
No. of Parameters	1738714
Inference time(s)	0.1021696s

TABLE V
PARAMETERS AND RESULTS FOR DENSENET

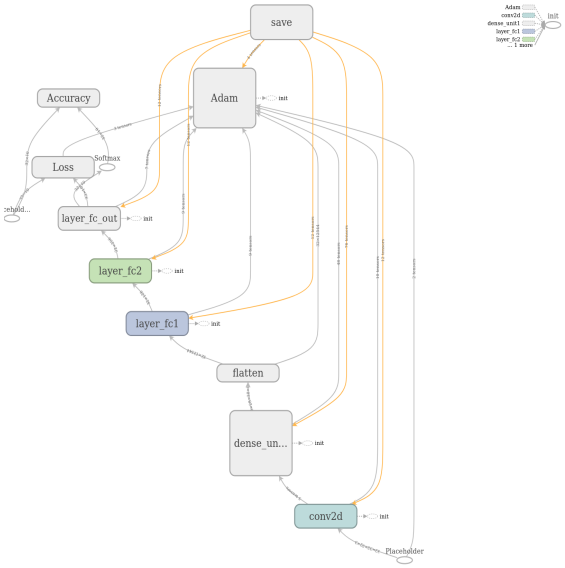

Fig. 50. DenseNet architecture

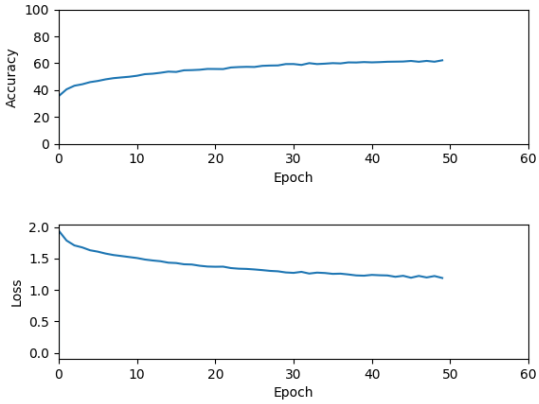

Fig. 51. Training and Loss vs Epochs for DenseNet

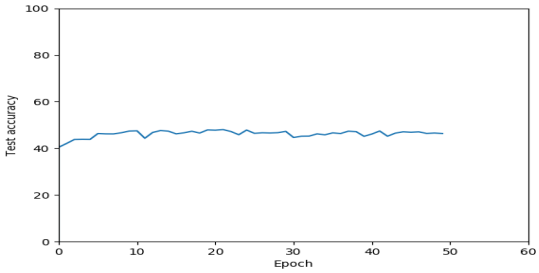

Fig. 52. Test accuracy vs Epochs for DenseNet

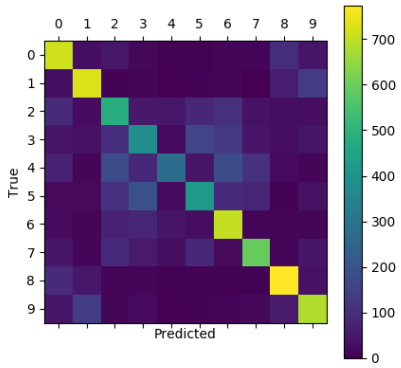

Fig. 53. Confusion Matrix for test results for ResNext

E. Conclusion and Discussion: Phase 2

In this project we implemented different neural network architectures on CIFAR-10 dataset to develop an image classifier. The hyperparameters that were used to train each network architecture are provided in a table format in each corresponding subsection. The results and accuracy of each network is provided in the corresponding subsection. A confusion matrix displaying the predictions vs true labels can be seen in the figs(35), (39), (43), (47) and (53). This concludes the results obtained from training each neural network.

From the exercise of implementation of different types of Neural Nets it has been observed that each architecture has its own advantages and disadvantages. A simple CNN takes very small time to train but the test results are not very accurate. Theoretically it can be said that by making the CNN deeper and deeper the accuracy of the network will improve, but it is not the case. As we increase the number of convolution layers the complexity of the network increases and after a certain limit the accuracy of this network decreases. This problem is called "Overfitting" the data. To counter this problem these different types of architectures were formed. These new networks avoid the overfitting problem in their own ways: by adding skip connections(ResNet), by introducing Cardinality(ResNext) or by feed-forwarding the input directly to every other layer in succession(DenseNet).

There is still a limit to the accuracy we can extract from such models. All the architectures that were discussed above rely on a supervised learning approach, in which there is an implicit assumption that there is enough data to train these models. The dataset given to us was a smaller version of the CIFAR-10 dataset. So, we applied techniques for data augmentation like geometrically transforming the image, cropping etc to generate more data so that the models could be trained better. But there can't ever be enough data. But adhering to the positives of such models, they still produce better results in terms of image classification than traditional image processing approaches.

REFERENCES

[1] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik, *Contour Detection and Hierarchical Image Segmentation*IEEE Trans. Pattern Anal. Mach. Intell. 33, 5 (May 2011), 898-916 Knuth: Computers and Typesetting,
http://dx.doi.org/10.1109/TPAMI.2010.161

[2] Gabor Filters.
https://en.wikipedia.org/wiki/Gabor$_f$ilterHvaas − $Labs/Tensorflow - Turorials$.
https://github.com/Hvass-Labs/TensorFlow-Tutorials

[3] Official Tensor Flow Turorials.
https://www.tensorflow.org/tutorials/images/deep$_c$nn

[4] Samuel L. Smith, Pieter-Jan Kindermans, Chris Ying, Quoc V. Le, *Don't Decay the Learning Rate, Increase the Batch Size*

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, *Deep Residual Learning for Image Recognition*
https://dblp.org/rec/bib/journals/corr/HeZRS15

[6] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, Kaiming He *Aggregated Residual Transformations for Deep Neural Networks*
https://arxiv.org/abs/1611.05431

[7] Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger *Densely Connected Convolutional Networks*
https://arxiv.org/abs/1608.06993