HW0 Alohomora
CMSC733

Srividya Ponnada
M.S. Computer Science
University of Maryland
College Park, MD, 20740
Email: sponnada@umd.edu

***Abstract*—This report is a compilation of my experiments using image processing techniques and deep learning to understand different techniques used to implement boundary detection and image classification in Computer Vision.**

I. Phase 1: Shake My Boundary

In the first phase, we implement the PB-Lite Algorithm for the boundary detection task. It uses the texture and color gradients along with the discontinuities in image intensities to detect the edges, unlike Canny or Sobel methods which use only the discontinuities in image intensities.

The process is to filter the images to access the low-level features and aggregate the regional features by developing the texton (Texture), brightness and color maps.

Here 3 filter banks, namely DoG filters, Leung-Malik filters and Gabor filters have been used to extract the texture properties of the images.

A. Filters

1) Oriented Derivative of Gaussian Filters: The Sobel operators (S_x, S_y) are convoluted with the Gaussian gradients along the x and y-axis. To get the filter for desired orientations we use the orientation angles to get the derivative of the Gaussian, by taking the cosine of the angle along G_x and the sine of the angle along G_y and adding them.

The filter bank is generated by using 2 different scale values and 16 orientations for each scale, linearly spaced from 0 deg to 360 deg. Hence, the total number of filters is 32.

2) Leung-Malik Filters: This filter bank has 48 filters consisting of first and second order Derivative of Gaussian Filters, Laplacian Filters, and Gaussian Filters. The Gaussian is plotted with an elongation factor of 3 and is transformed using a rotation matrix for orientations. Sobel operator is convoluted with Gaussuan to calculate both the first and second derivative of Gaussians. 8 Symmetrical Gaussians are convoluted with the Laplacian Kernel to get the Laplacian filter. Gaussian filters at 4 different scales are finally appended to give the desired filter bank.

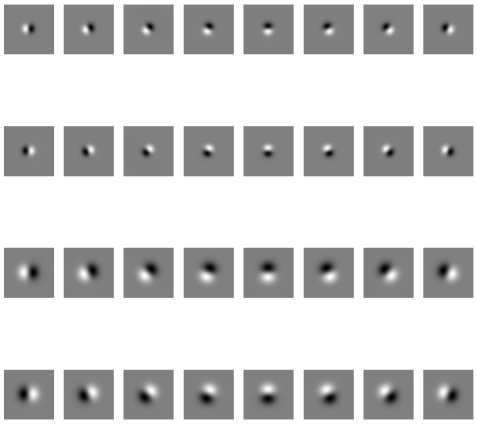

Fig. 1. Derivative of Gaussian

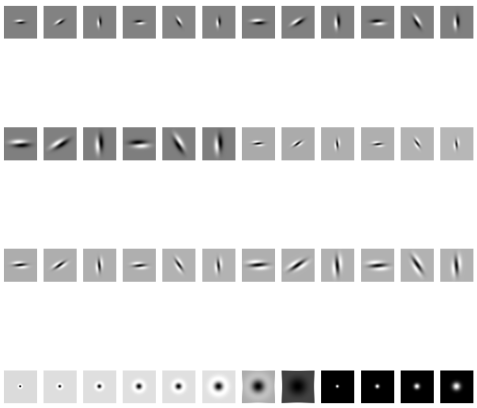

Fig. 2. Leung-Malik Filters - Large

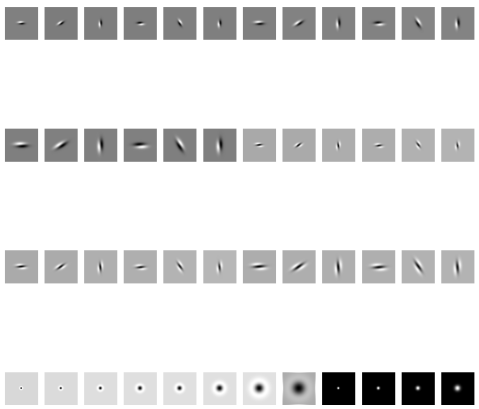

Fig. 3. Leung-Malik Filters - Small

3) Gabor Filters: A Gaussian kernel is modulated with a sinusoidal plane wave to generate a Gabor filter. It observes content with specific frequency in the image, in specific directions around the point of interest. To implement this filter I took scales of 10 and 30, 12 orientations, 3 frequencies and a filter size of 50.

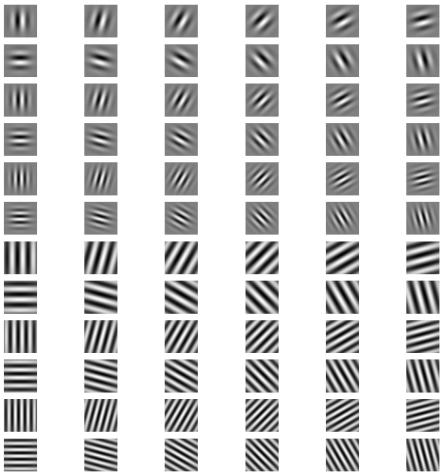

Fig. 4. Gabor Filters

B. Maps

1) Texton Maps: We use filters to identify the texture properties of an image. The 3 filter banks consist of multiple filters and this results in a vector of filter responses that have the texture property of each pixel encoded in them. These vectors are further clustered together using K-means algorithm where K = 64, the output of this clustering is a Texton Map (τ). The corresponding images are shown in Fig 5.

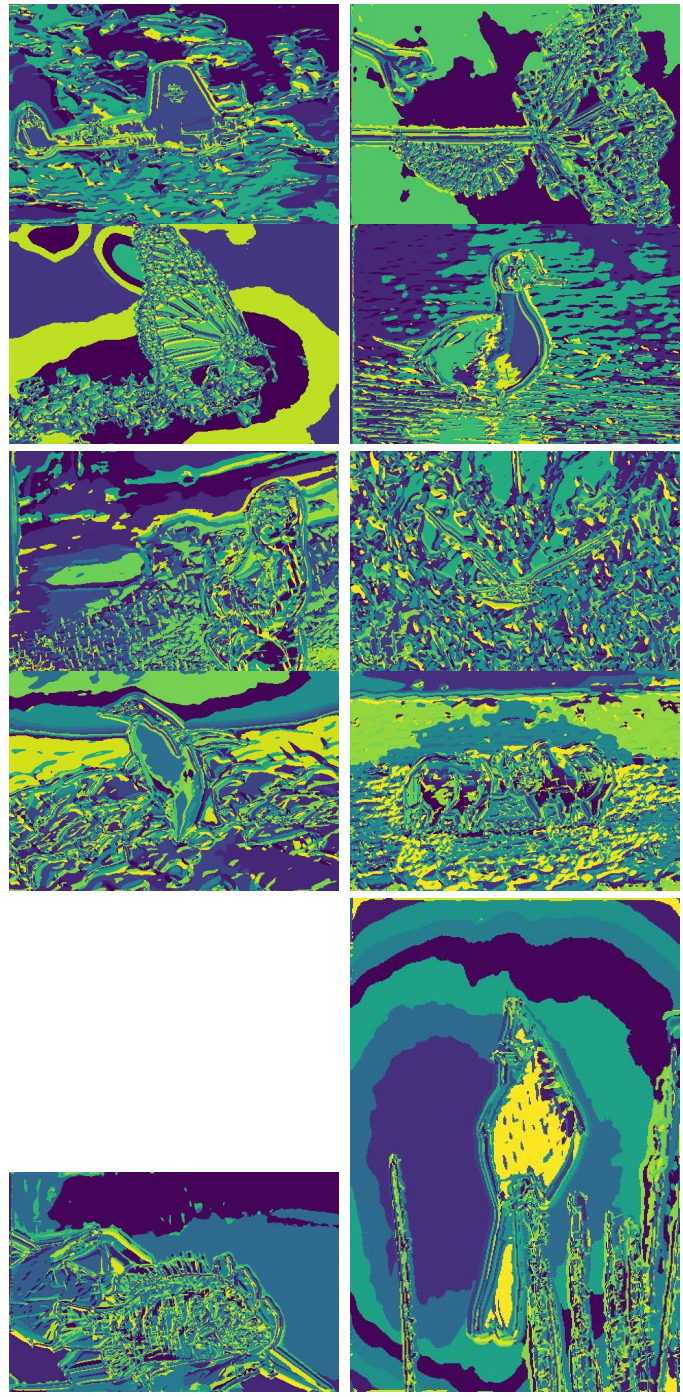

Fig. 5. Texton Maps of given images

2) Brightness Maps (β): Here the changes in intensities of light in the image are captured. Based on the brightness value each pixel is clustered using K-means clustering with K = 16. The grayscale images are used to generate this map. The

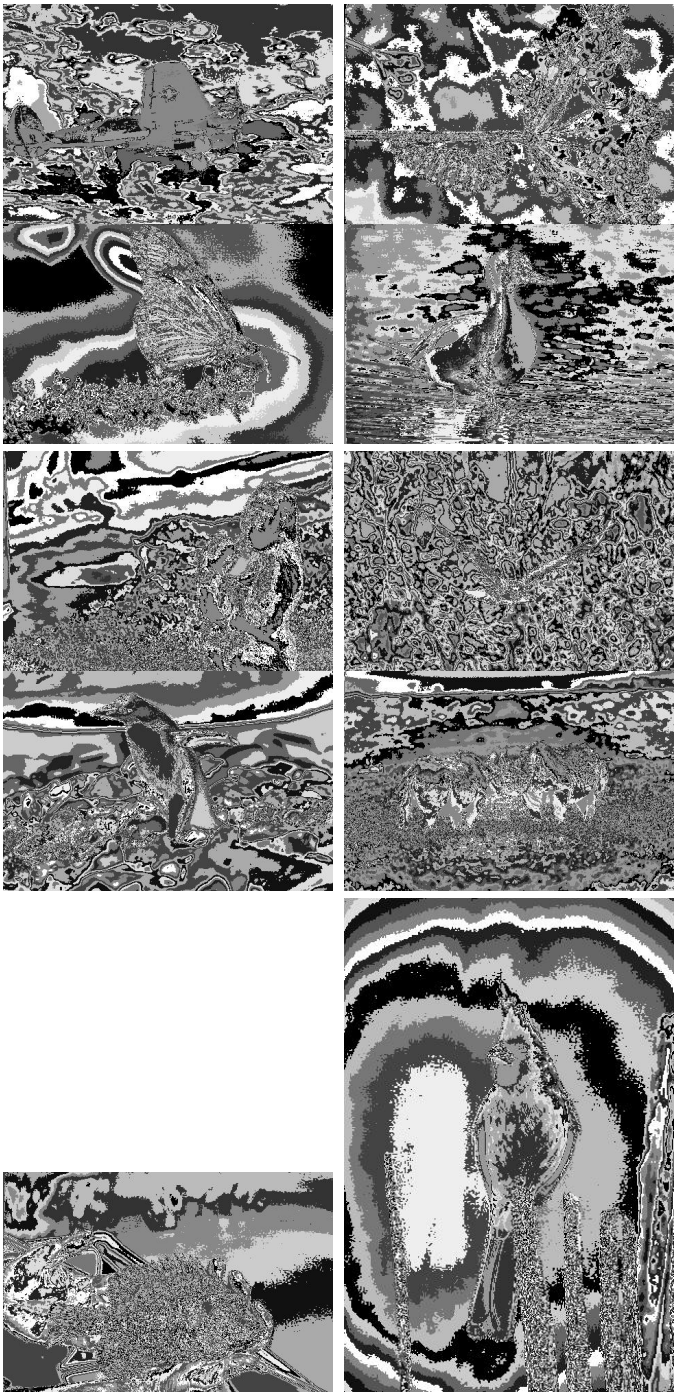

Fig. 6. Brightness Maps of given images

images are shown in Fig 6.

3) Color Maps (ς): The images are clustered (K-means clustering using K = 16) based on the color of the pixel using the RGB value. The images are shown in Fig. 7.

C. Gradients

1) Half Disk Masks: Half Disks encode the changing texture, brightness, and color distributions at each pixel. When

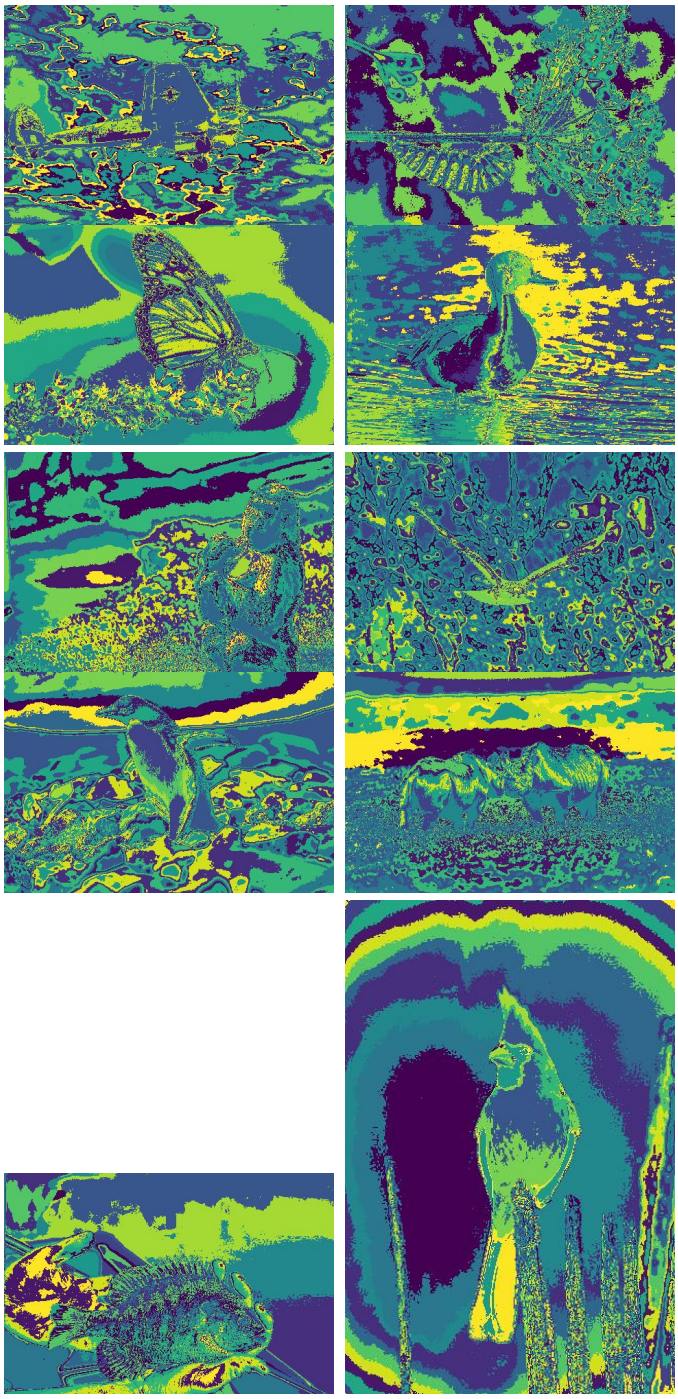

Fig. 7. Color Maps of given images

these are convoluted with each pixel the gradient is high if the values are dissimilar and they are low if the values are similar. Fig 8. shows the half-disk masks for 5 scales and 16 orientations.

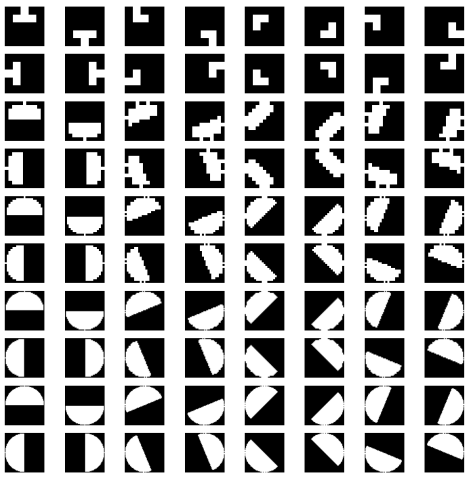

Fig. 8. Half Disk Masks Filters

2) Chi-Square Distance: The images are first convoluted with left or right half disk pairs and we get 2 histograms. N matrices with pixel value corresponding to χ^2 distance are obtained for N pair of hard disks. The average of these matrices is used to compute the gradient of τ, β, ς. The values of the K index for texton, brightness, and color gradients are 64, 16, and 16 respectively.

D. PB-Lite

The features of Canny and Sobel baselines are combined with the gradients of τ, β, ς.

$PbEdges = (t+g+b)/3 \circ (w1 * cannyPb + w2 * sobelPb)$

$\circ$ represents the Hadamard product operator, which is element-wise multiplication between two matrices. $w1$ and $w2$ represent the weights to get the Canny and Sobel features respectively. The constraint is that $w1 + w2 = 1$.

E. Analysis of PB-Lite Output

PB-Lite outputs have less noise or background texture considered. Initial experiments were tried with different numbers of Texton_bin, Brightness_bin, and Color_bin values to generate the gradient maps. The intuition was that the features would be more evident when we increase the bin values and the n_init values and at the same time increasing the number of half disks increased the sharpness of the image generated by PB-Lite edge detector. The optimal bin values were as stated before; 64, 16, and 16 (Tbin, Bbin, and Cbin). The advantage of PB-Lite is that it avoids the detection of noise or disturbance in the background and we get a more clear edge detection of the object we focus on in the image. Also some experiments

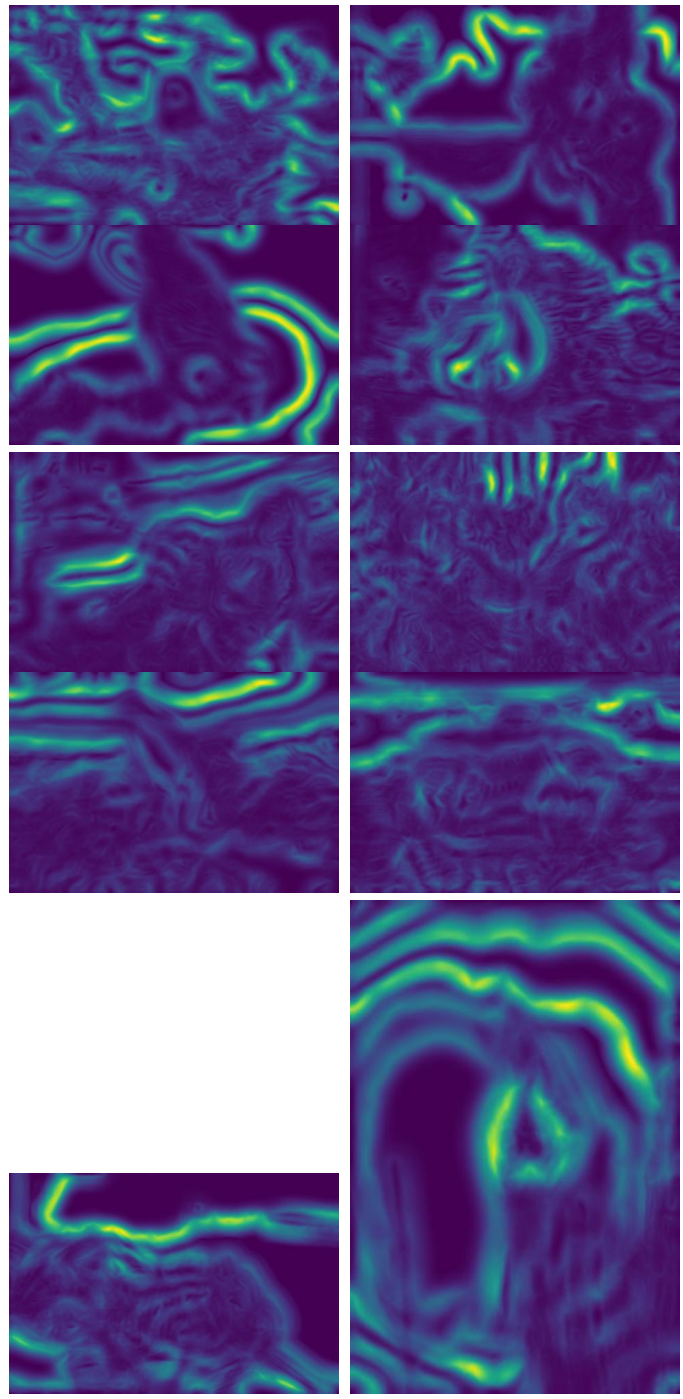

Fig. 9. Texture Gradients of given images

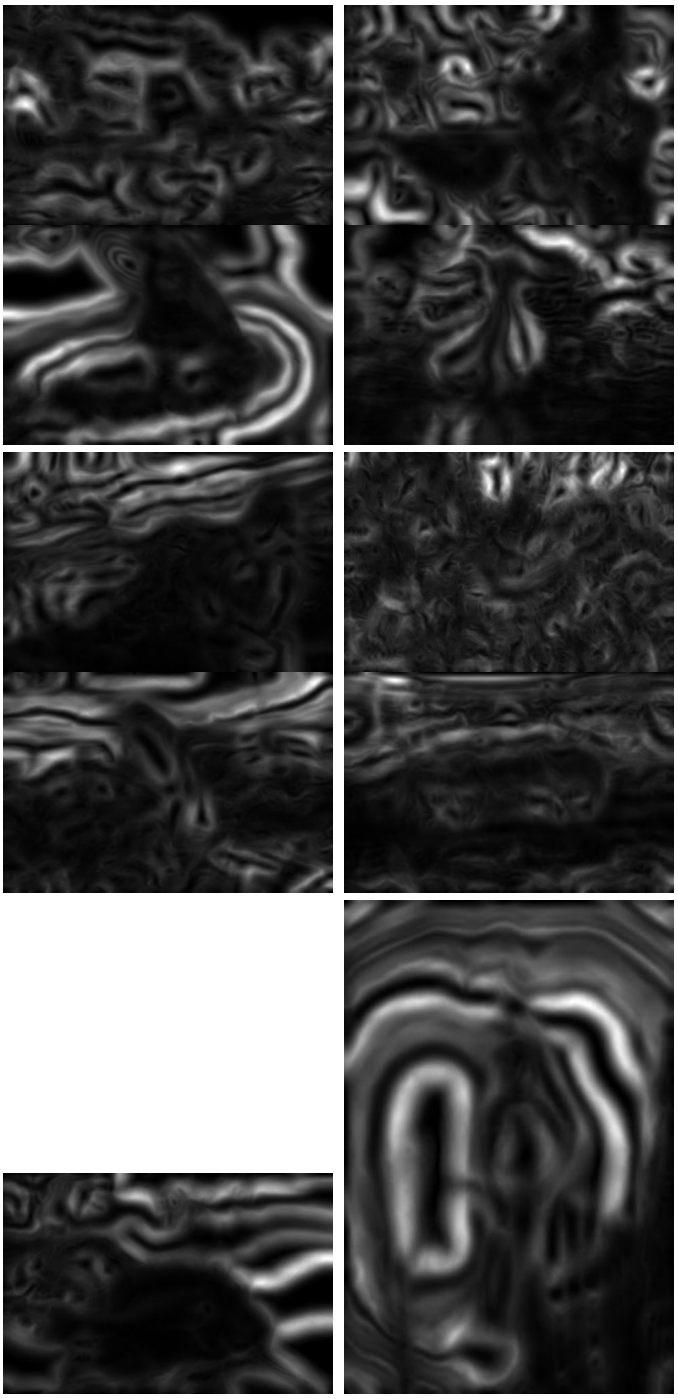

Fig. 10. Brightness Gradients of given images

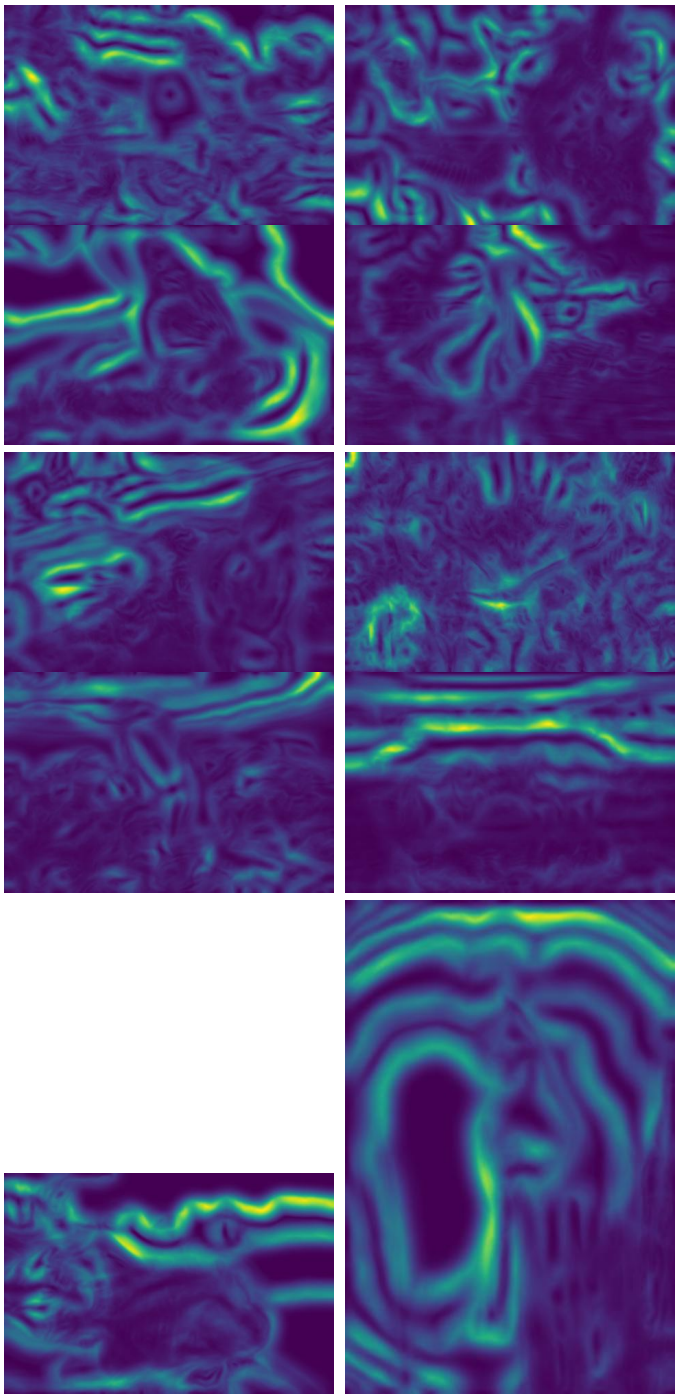

Fig. 11. Color Gradients of given images

with the Canny and Sobel features, weight values $w1$ and $w2$ resulted in both of them being 0.5 each giving the most optimal edge detection output for PB-Lite.

II. Phase 2: Deep Dive on Deep Learning

In this experiment, a simple neural network and its modified version are used for the classification of the CIFAR10 dataset (50000 training samples and 10000 test samples). Further, a comparison of the architectures of ResNet, ResNext, and Densenet is done. The starter code had a few outdated pieces of code which had to be rectified, also due to limited hardware resources and time, was able to experiment with only a few parameters. For a few experiments i.e. to run the modified neural network and the Densenet I used the free T4 GPU available on Colab. Once this was exhausted I had to use my laptop's CPU to train the other models.
The simple neural network had 3 convolution and 2 fully connected layers. The softmax function was used for loss calculation and Adam optimizer was used to train the network. The learning rate was 0.001, the batch size was 128, and the number of epochs the model was trained was 20.
The modifications for the modified neural network include increasing the network's depth (improves learning), increasing filter size (extracting more features), and batch normalization. The improvement can be seen in the training and test loss and accuracy vs epochs, test graphs, and the confusion matrices. The batch size used is 128 with a learning rate 0.001, trained for 20 epochs. The accuracy for test of the Neural Network and the modified network are 60.93% and 65.3%.

In ResNeXt cardinality is introduced keeping the residual learning framework of ResNet. The input goes through several split transformations based on cardinality before being passed to fully connected layers. Convolutions are applied independently on the input based on the cardinality. The outputs from the paths are concatenated and added to the input, and non-linear activation. Classification accuracy increases with cardinality while maintaining network complexity Increasing cardinality is more effective to improve performance compared to increasing the depth or width of the model.

The DenseNet Architecture solves the vanishing gradient problem each layer is connected to every layer in the feed-forward network. It strengthens feature propagation and reuse therefore decreasing the number of features by maintaining the the accuracy.

The first Residual Block consists of 32 filters, and the second one has 64. They are connected to 2 fully connected layers with 100 and 10 units each. The residual block is made up of 2D convolutional layers, Batch Normalization, and Relu. There are 3 such layers and the input is taken into consideration right before the final relu. The batch size used to train this model is 128 for 10 epochs with a learning rate of 0.001. I obtained an accuracy of 10.73% after training it for 10 epochs which took close to 2 and a half hours. I will have to look into this further. Due to limited time and resources I have continued to implement ResNext and Densenet where I

Fig. 12. Canny, Sobel, and PB-Lite Outputs respectively

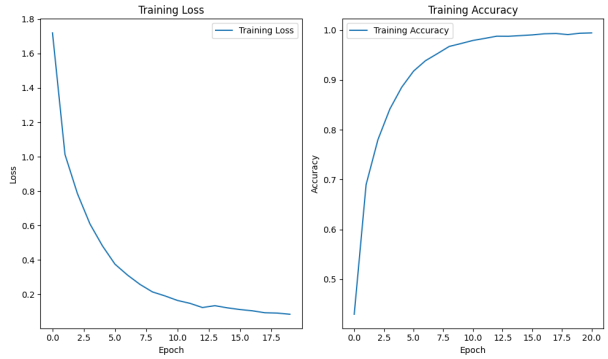

Fig. 13. Neural Network - Training Loss and Accuracy Vs Epochs

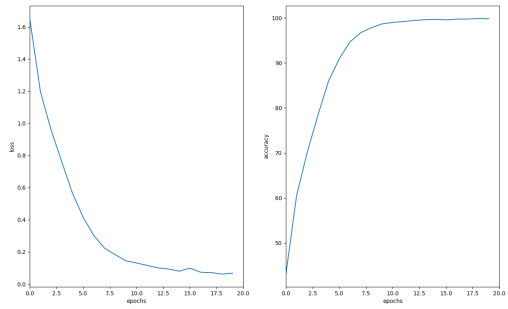

Fig. 14. Neural Network Modified - Training Loss and Accuracy Vs Epochs

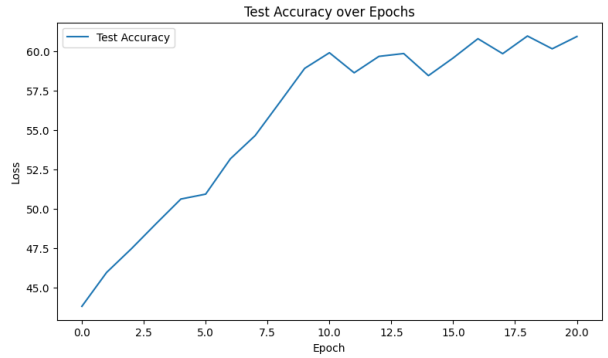

Fig. 15. Neural Network - Test Accuracy Vs Epochs

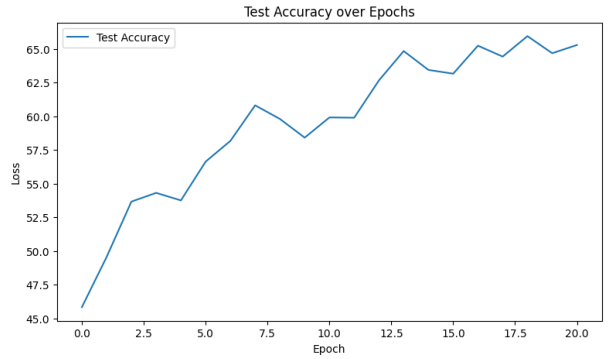

Fig. 16. Neural Network Modified - Test Accuracy Vs Epochs

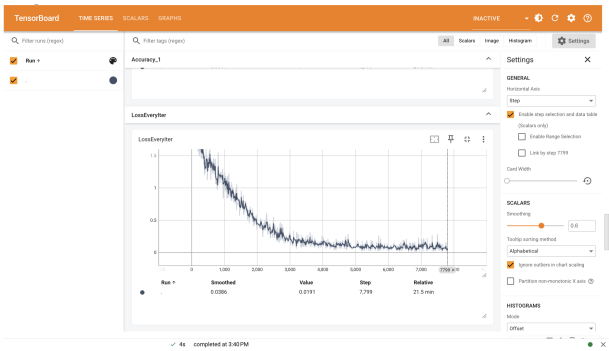

Fig. 17. Neural Network - TensorBoard Loss (Logs)

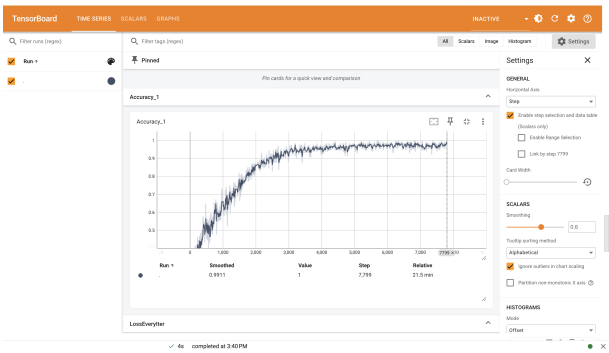

Fig. 18. Neural Network - TensorBoard Accuracy (Logs)

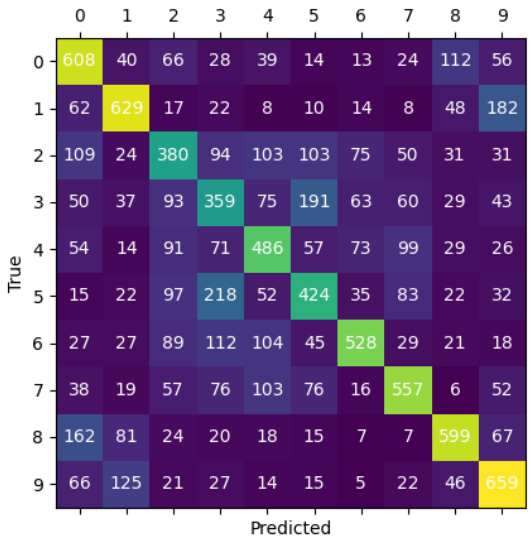

Fig. 19. Neural Network - Confusion Matrix (Test)

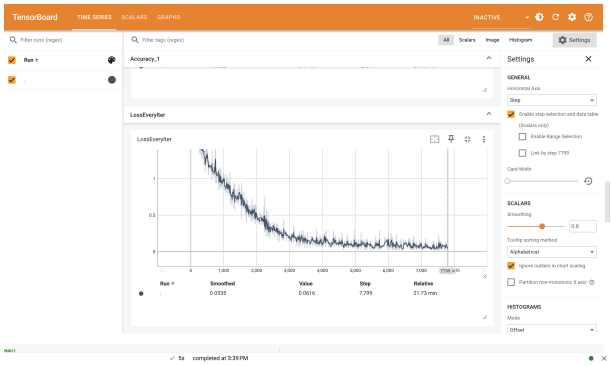

Fig. 20. Neural Network Modified - TensorBoard Loss (Logs)

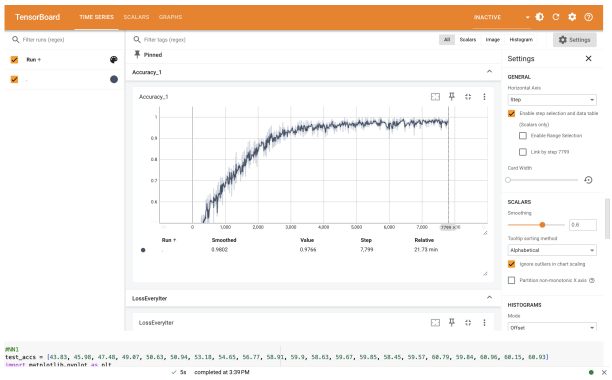

Fig. 21. Neural Network Modified - TensorBoard Accuracy (Logs)

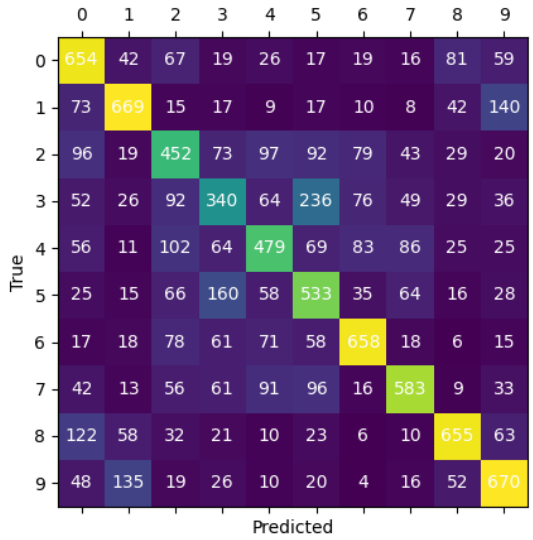

Fig. 22. Neural Network Modified - Confusion Matrix (Test)

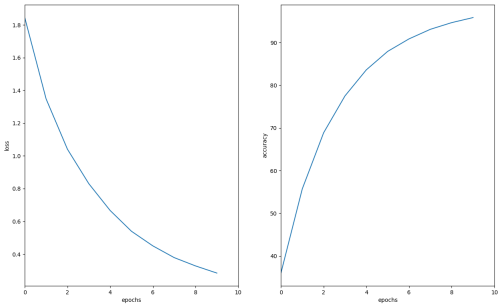

Fig. 23. ResNext - Training Loss and Accuracy Vs Epochs

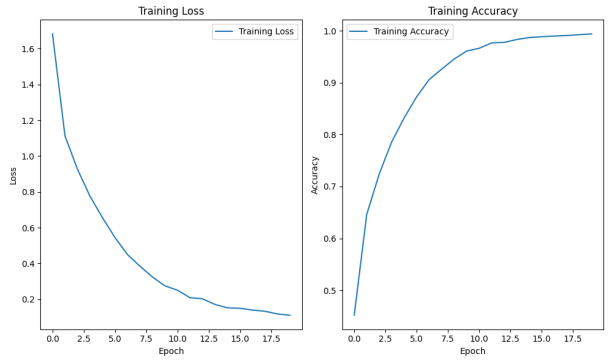

Fig. 24. DenseNet - Training Loss and Accuracy Vs Epochs

got an accuracy of 56.7% and 85.63% respectively. I trained ResNext on CPU and DenseNet on GPU. I think this huge difference might be because of the resource constraint. But I got a training accuracy of 95.85% for ResNext for 10 epochs. For Densenet I got an accuracy of 99.38%, I trained it for 20 epochs on T4 GPU on Colab.

A. Conclusion

Edge Detection was performed for given images using the PB-Lite algorithm. Implemented simple architectures and trained and tested the networks from simple neural networks,

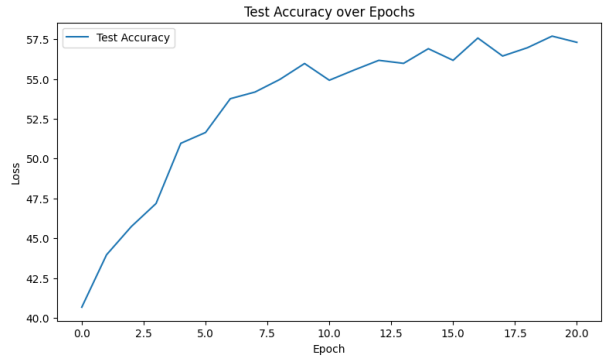

Fig. 25. ResNext - Test Accuracy Vs Epochs

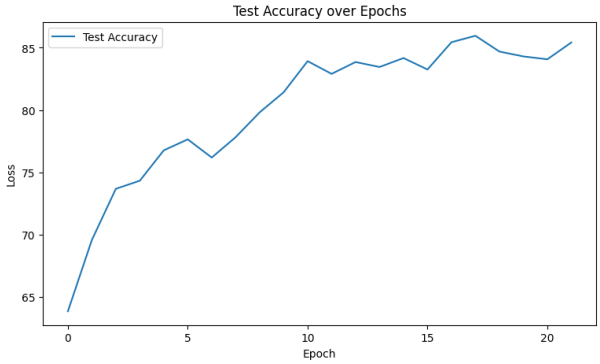

Fig. 26. DenseNet - Test Accuracy Vs Epochs

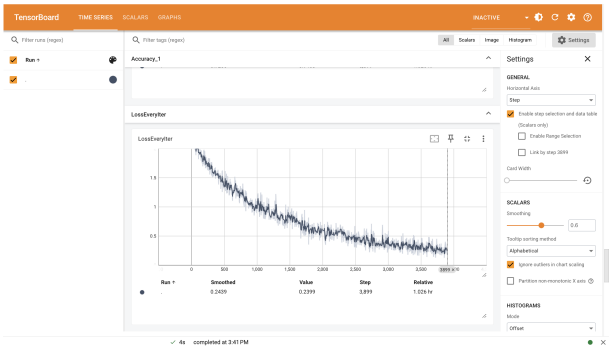

Fig. 27. ResNext - TensorBoard Loss (Logs)

modified Neural Network, ResNet, ResNext, and DenseNet by gaining increasing accuracies.

References

References

[1] https://cmsc733.github.io/2022/hw/hw0/
[2] https://www.robots.ox.ac.uk/ vgg/research/texclass/filters.html
[3] https://en.wikipedia.org/wiki/Gabor_filter
[4] https://www.cs.cornell.edu/courses/cs6670/2011sp/lectures/lec02 filter.pdf
[5] https://github.com/taki0112/Densenet-Tensorflow
[6] https://github.com/pytorch/vision/blob/6db1569 c89094cf23f3bc41f79275c45e9fcb3f3/torchvision/models/dense net.pyL126
[7] https://arxiv.org/abs/1608.06993v5

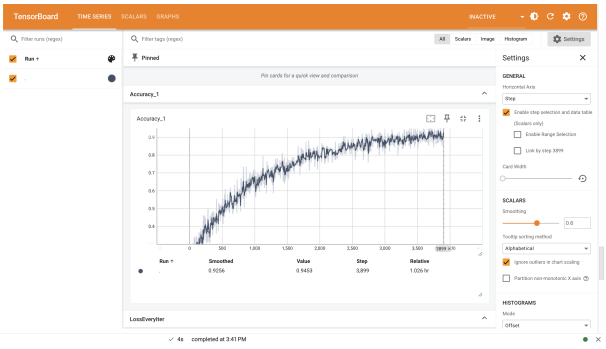

Fig. 28. Resnext - TensorBoard Accuracy (Logs)

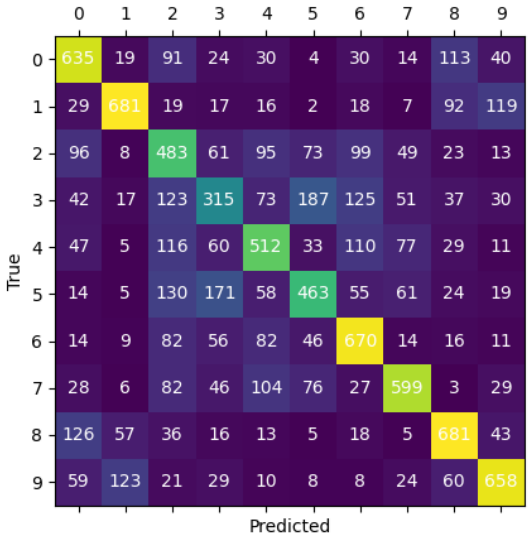

Fig. 29. ResNext - Confusion Matrix (Test)

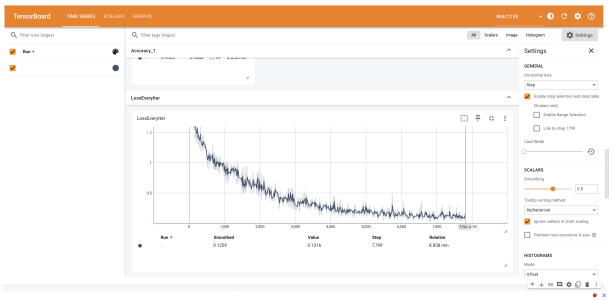

Fig. 30. DenseNet - TensorBoard Loss (Logs)

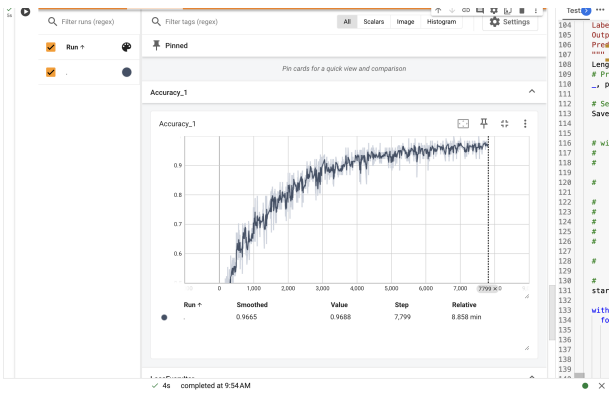

Fig. 31. DenseNet - TensorBoard Accuracy (Logs)

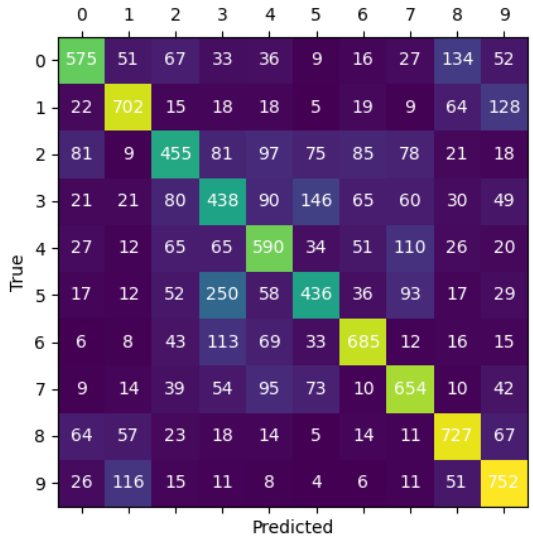

Fig. 32. DenseNet - Confusion Matrix (Test)

[8] https://github.com/pytorch/vision/blob/6db1569c89094cf23f3 bc41f79275c45e9fcb3f3/torchvision/models/resnet.pyL124

[9] https://arxiv.org/abs/1611.05431v2

[10] https://github.com/sakshikakde/probability-based-edge-detection

[11] https://github.com/abhi1625/CMSC-733/tree/master/Abhi1625_hw0