RBE 549 Homework 0 - Alohomora

UdayGirish Maradana
MS Robotics Engineering
Worcester Polytechnic Institute
Worcester, Massachusetts - 01609
Email: umaradana@wpi.edu
Using 1 late day

***Abstract*—This paper explains the work that I did as a part of RBE 549 Homework 0. The Homework is about creating a Probability of Boundary Detection algorithm and training neural networks such as ResNet, ResNext, DenseNet on CIFAR-10 Dataset.**
Index Terms - Probability of Boundary Detection, Convolutional neural networks

I. Introduction

The homework is divided into two phases. Phase I which explains more about the process of the Probability-based Boundary detection method. Phase II explains the process of training different convolutional neural networks on the CIFAR-10 dataset. Edge Detection is one of the primary algorithm in the computer vision as to understand any object I need to know its boundary and the Phase I essentially deals with. The other part which is CNN based Image classification which covers the basics of Neural network training using pytorch understanding different strategies etc.

II. Phase I: Shake My Boundary

A. Introduction

In this section I will go over the basics of the implementation of a light version of Probability based boundary detection algorithm explained in [1] . According to the original paper , the algorithm performs better than some of the Ill-known traditional approaches such as Canny and Sobel Detectors. The primary reason for this performance in most of the cases is due to the fact that the Pb lite algorithm takes care of false positive made because of the textures.

The algorithm primarily consists of four steps which are explained briefly in [1].

1) Constructing various types of **Filter Banks**
2) Building **Texture, Brightness and Color Maps**
3) Texture, Brightness and Color **Gradients**
4) **Pb lite output** combined with Canny and Sobel baselines.

I briefly discuss these things in the following subsections. Further, I use the **BSDS500 dataset** for comparing the Pb lite output with traditional Sobel and Canny filters. The overall architecture is explained in the Fig.1.

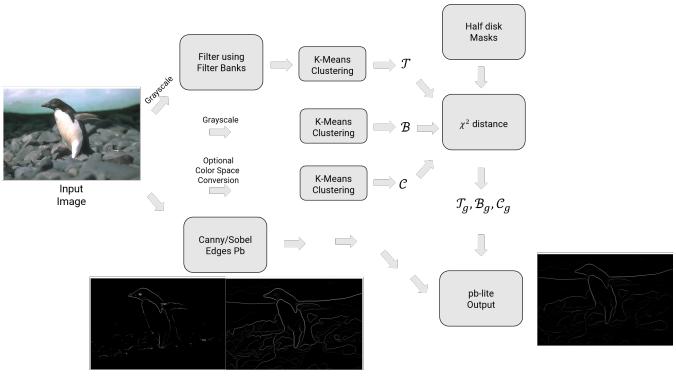

Fig. 1. Overview of the Pb lite Pipeline

B. Filter Banks

This step is all about building filters which helps us build low-level features. These features helps us to understand regional texture and brightness properties. Once I create these filter bank I can proceed further building a Texton map which captures the texture properties of the image. Here I build primary three different sets of filter banks which are explained below:

1) Oriented Derivative of Gaussian (DoG) Filters: This is a simple filter which involves convolving a simple Sobel filter and a Gaussian Kernel and finally rotating the result. Selecting different standard deviations for the Gaussian Kernel and rotating at random positions helps us to get a wide variety of filters. (See Fig.2)

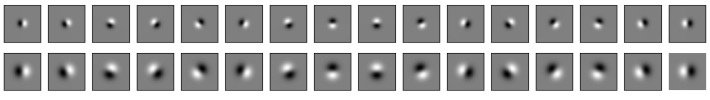

Fig. 2. DoG Filters

2) Leung-Malik Filters: The Leung-Malik (LM) filters are a set of multi scale, multi orientation filter bank with 48 filters. It consists of the below filters. (See Fig.3 and Fig.4)

1) 36 Filters - First and Second Order derivatives of Gaussians at 6 orientations and 3 Scales. These first and second order derivatives occur at the 3 scales with a elongation factor α, which means having different

standard deviations along x and y axes (i.e, $\sigma_x = \sigma, \sigma_y = \alpha \times \sigma$.

2) Four Gaussians at four difference scales.
3) 8 Laplacian of Gaussian (LOG) Filters which occur at two scales σ and 3σ.

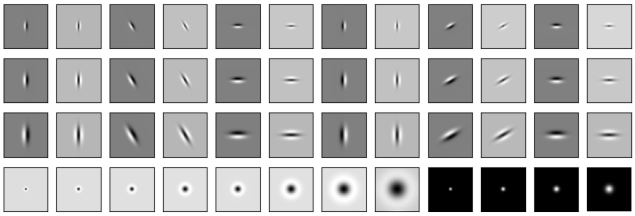

Fig. 3. LM Small Filters

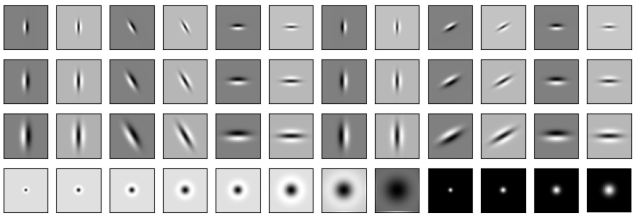

Fig. 4. LM Large Filters

Overall, the scales used for the filters are $[1, \sqrt{2}, 2, 2\sqrt{2}]$. And the orientations are in the range of $(0, 180)$ degrees. Here Laplacian of Gaussian filters are constructed by convolving a laplacian filter on the Gaussians at different scales.

3) Gabor Filters: Gabor filters are very interesting as their inspiration is based from the human visual system. A gabor filter is a gaussian kernel function modulated by a sinusoidal plane wave. The Gabor filter is primarily used for Texture analysis and is a linear filter that analyzes whether there is any specific frequency content in the image in specific directions in a localized region around the region of analysis. (See Fig.5)

C. Texture, Brightness and Color Maps $(\mathcal{T}, \mathcal{B}, \mathcal{C})$

1) Texton Map: Filtering an input image with each element of the filter bank created above results in a vector of filter responses centered on each pixel. For instance, if your filter bank has N filters, you'll have N filter responses at each pixel. A distribution of these responses could be thought of as encoding texture properties. This representation is simplified with a discrete texton ID. I will do this by clustering the filter responses into K Textons which makes the pixels having unique texton ID. Here I used the cluster size as K = 64.

2) Brightness Map: Here I take the image and cluster the brightness values using KMeans. This map is called Brightness Map. Here I used the cluster size as K = 16.

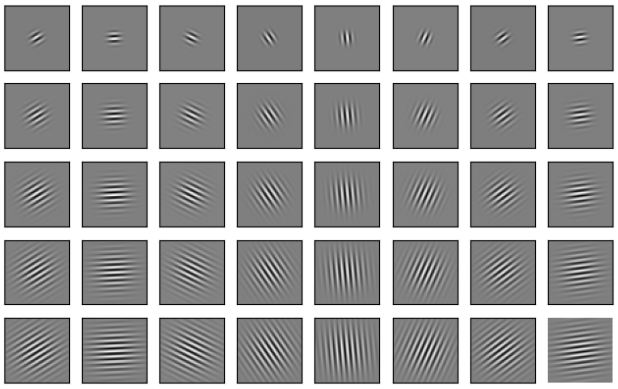

Fig. 5. Gabor Filters

3) Color Map: Here I take the image and capture the color changes or chrominance content in the image. I cluster the color values using Kmeans. I can also cluster each color channel separately here. Please find the texton, brightness and Color Maps of the given BSDS500 Dataset in the figures (6-15).

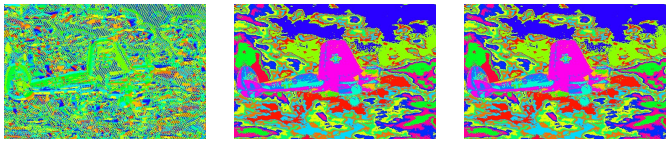

Fig. 6. $(\mathcal{T}, \mathcal{B}, \mathcal{C})$ for Image 1

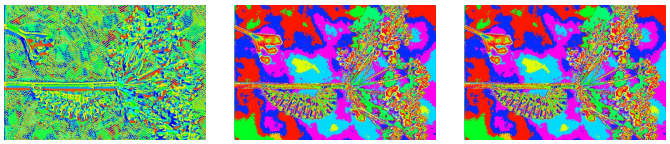

Fig. 7. $(\mathcal{T}, \mathcal{B}, \mathcal{C})$ for Image 2

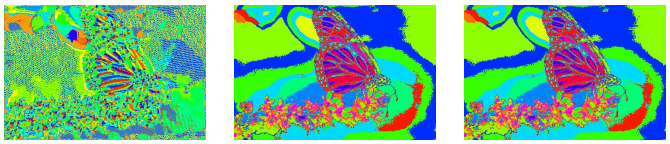

Fig. 8. $(\mathcal{T}, \mathcal{B}, \mathcal{C})$ for Image 3

D. Texture, Brightness and Color Gradients $(\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g)$

To obtain gradients the simple step is to compute differences of values across different shapes and sizes. But doing that on an image is a hard problem as it involves looping over the pixels multiple times and computational inefficient. To solve this problem, we leverage the concept of half-disc masks.
Before going there, why gradients are needed ?
Because gradients helps us to understand changes in the

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

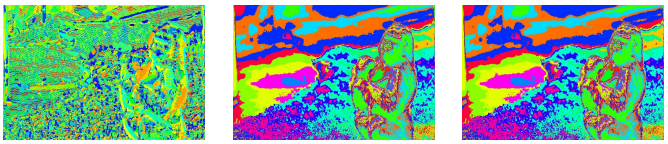

Fig. 10. $(\mathcal{T}, \mathcal{B}, \mathcal{C})$ for Image 5

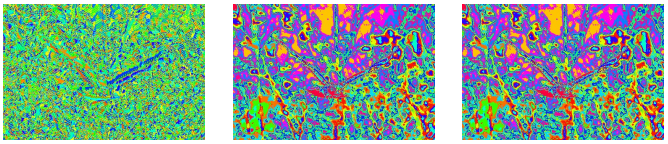

Fig. 11. $(\mathcal{T}, \mathcal{B}, \mathcal{C})$ for Image 6

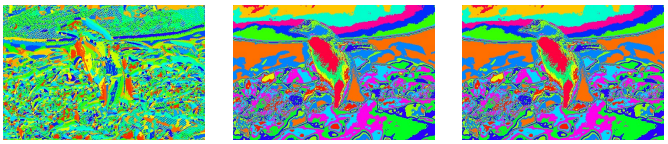

Fig. 12. $(\mathcal{T}, \mathcal{B}, \mathcal{C})$ for Image 7

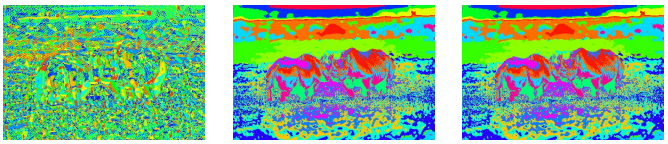

Fig. 13. $(\mathcal{T}, \mathcal{B}, \mathcal{C})$ for Image 8

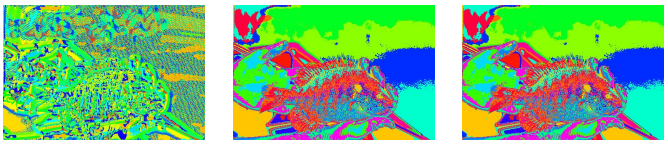

Fig. 14. $(\mathcal{T}, \mathcal{B}, \mathcal{C})$ for Image 9

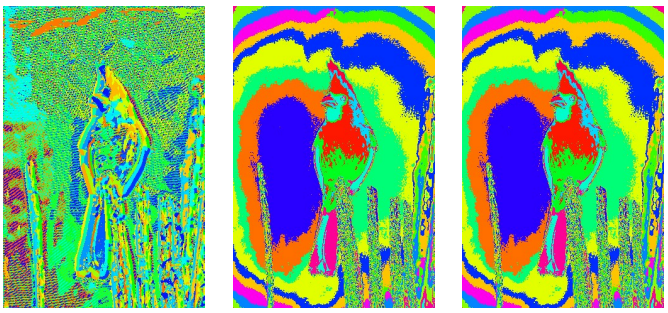

Fig. 15. $(\mathcal{T}, \mathcal{B}, \mathcal{C})$ for Image 10

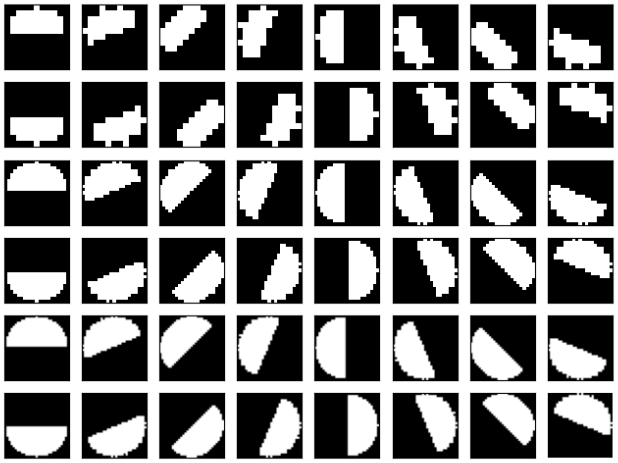

Fig. 16. Half Disk Masks at Radius $[5, 10, 15]$

images which gives the primary idea of a edge as edge is a location where a change is happening in a 2D image whether it is texture, color or brightness. And most of the traditional filters also work on this concept. Please find the texton, brightness and Color gradients of the given BSDS500 Dataset in the figures (17-26).

1) Half Disc Masks: The half-disc masks are simply (pairs of) binary images of half-discs. This helps us to compute the χ^2 (chi-square) distances using a filtering operation (Similar to Convolution). This enables us to loop over each pixel faster than looping over each pixel neighborhood and aggregating counts for histograms. I have formed these scales at 8 orientations and 3 scales. The primary idea behind this method is that if the distributions are similar the gradient should be small and if the distributions are dissimilar then the gradients should be large. Because our half discs span multiple scales and orientations, I will end up with a series of local gradient measurements encoding. Please find the Half Disc Masks filter Fig.16

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

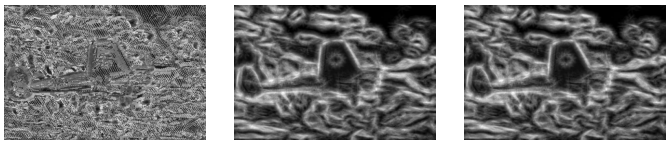

Fig. 17. $(\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g)$ for Image 1

E. Pb Lite Output

The final step of getting the Pb lite output involves combining the information from the features with a baseline method

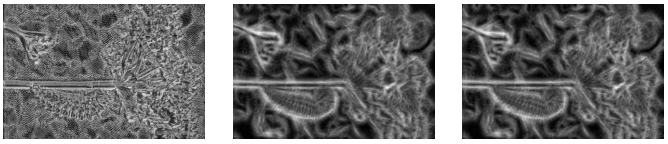

Fig. 18. $(\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g)$ for Image 2

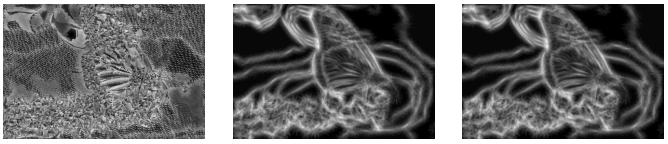

Fig. 19. $(\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g)$ for Image 3

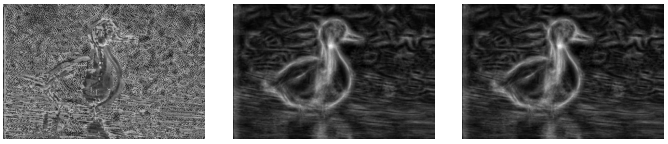

Fig. 20. $(\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g)$ for Image 4

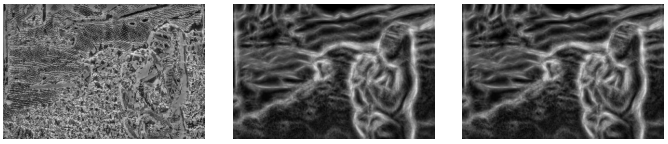

Fig. 21. $(\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g)$ for Image 5

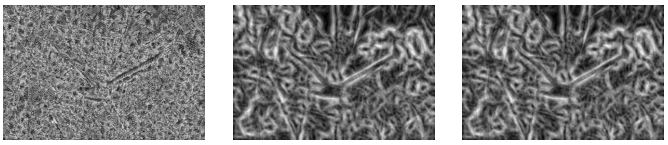

Fig. 22. $(\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g)$ for Image 6

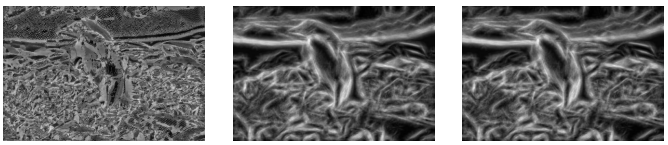

Fig. 23. $(\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g)$ for Image 7

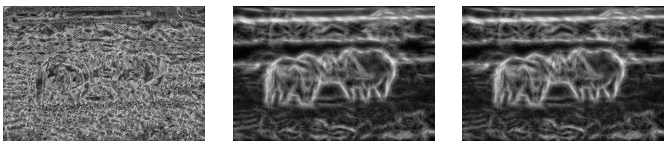

Fig. 24. $(\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g)$ for Image 8

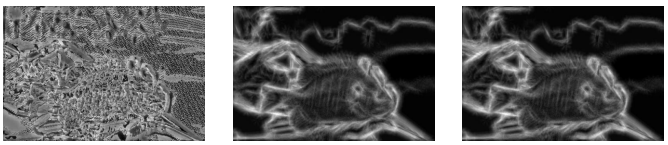

Fig. 25. $(\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g)$ for Image 9

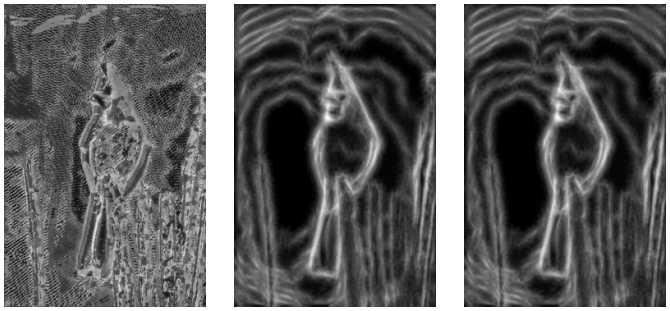

Fig. 26. $(\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g)$ for Image 10

(based on Sobel or Canny or some Iighted average of both) using the below equation.

$$PbEdges = \frac{(\mathcal{T}_g + \mathcal{B}_g + \mathcal{C}_g)}{3} \odot (w_1 * cannyPb + w_2 * sobelPb) \tag{2}$$

Here, $\odot$ is the Hadamard product operator. And the w_1 and w_2 are chosen as 0.5 in the implementation. This can be varied and tunable and can get better results in different scenarios.

The Outputs of Pblite can be seen in the figures (27-36).

Fig. 27. Comparison of Pblite (1) with Sobel(2) and Canny(3) for Image 1

Fig. 28. Comparison of Pblite (1) with Sobel(2) and Canny(3) for Image 2

Fig. 29. Comparison of Pblite (1) with Sobel(2) and Canny(3) for Image 3

F. Discussion

From the above results we can understand that the Pb lite algorithm is able to reduce some of the false positive in Canny and Sobel baselines with respect to texture. Even though they look a little light , the edges in the Pblite are almost outlining the object in the image. And Pb lite gives you more flexibility in terms of tunable parameters of filter banks which can almost customise most of the images. Being said that using this algorithm in many cases in real life can be tricky and needs adaptation to the scenario. Even though it works , it needs a bit more time to tune it produce richer outputs.

Fig. 30. Comparison of Pblite (1) with Sobel(2) and Canny(3) for Image 4

Fig. 31. Comparison of Pblite (1) with Sobel(2) and Canny(3) for Image 5

Fig. 32. Comparison of Pblite (1) with Sobel(2) and Canny(3) for Image 6

Fig. 33. Comparison of Pblite (1) with Sobel(2) and Canny(3) for Image 7

Fig. 34. Comparison of Pblite (1) with Sobel(2) and Canny(3) for Image 8

Fig. 35. Comparison of Pblite (1) with Sobel(2) and Canny(3) for Image 9

Fig. 36. Comparison of Pblite (1) with Sobel(2) and Canny(3) for Image 10

III. Phase II: Deep Dive on Deep Learning - CIFAR10

A. Introduction

In this Phase, I will be working on the implementation of multiple neural network architectures on CIFAR-10 dataset and their performance based on loss and accuracy will be compared. Various methods to improve the performance such as weight decay(regularization), data augmentation, standardization and learning rate decay are explored.

One of the primary approach to train on something visual data is to use a class of neural networks called Convolution Neural networks. CNN's are quite capable as they almost work on the same principle as filters. The Traditional filters which I have discussed are hand designed to understand specific feature. CNN's also can use different scales and can capture different feature with the filters it have. The only trick is that these features are learnable. Some of the common problems which are faced by Normal Custom CNN's might be:

1) Vanishing Gradients: During Backprop, Gradients can become extremely small as they Ire propogated back through many layers, leading to very slow or halted learning in early layers.
2) Exploding Gradients: In Backprop, Gradients can become excessively large during backpropagation, causing instability and making it difficult to find meaningful updates to the model parameteres.
3) Computational Efficiency: Deep Neural networks with a large number of parameters can be computationally expensive and even though they are accuracte, in real world they are not much useful especially in resource-constrained environments.
4) Feature propogation:Feature propogation is the process by which features are extracted at different layers of a neural network are passed or propagated through the network.

B. CIFAR10 - Custom Network

Here I start with a very basic Custom network with three convolutional layers with Batch norm and Maxpool and Dropout which is shown in Fig.37.

The Loss and accuracy curves are shown in Fig.38 and Fig.39 respectively and Confusion Matrix - Fig (40 and 41). The train and test accuracy are 99.30% and 68.67%. This model is further trained and the best performance is indicated in Table I.

C. ResNet 34

The primary idea behind the ResNet architecture is to introduce residual learning by using skip connections or shortcuts, allowing the network to learn residual functions. So, Resnet solves the primary problem which is Vanishing Gradients. For several years, it used to be the best concept till I Ire faced with Visual transformers. Even now, the concepts such as Residual and Identity mapping are used in several popular architectures. A block of how a Residual mapping looks is shown in Figure 42. This is from the ResNet 34 layers implementation I did.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 32, 32, 32]             896
              ReLU-2          [-1, 32, 32, 32]               0
       BatchNorm2d-3          [-1, 32, 32, 32]              64
            Conv2d-4          [-1, 16, 32, 32]           4,624
              ReLU-5          [-1, 16, 32, 32]               0
       BatchNorm2d-6          [-1, 16, 32, 32]              32
            Conv2d-7           [-1, 8, 32, 32]           1,160
              ReLU-8           [-1, 8, 32, 32]               0
       BatchNorm2d-9           [-1, 8, 32, 32]              16
        MaxPool2d-10           [-1, 8, 16, 16]               0
       Dropout2d-11           [-1, 8, 16, 16]               0
           Linear-12                 [-1, 512]       1,049,088
             ReLU-13                 [-1, 512]               0
       Dropout2d-14                 [-1, 512]               0
           Linear-15                  [-1, 10]           5,130
================================================================
Total params: 1,061,010
Trainable params: 1,061,010
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 1.36
Params size (MB): 4.05
Estimated Total Size (MB): 5.41
----------------------------------------------------------------
```

Fig. 37. Custom Neural Network

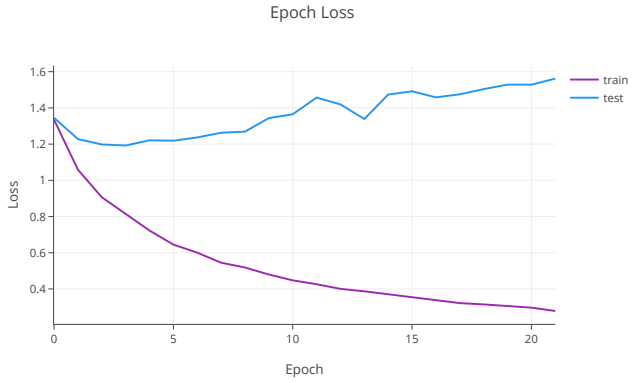

Fig. 38. Custom Neural Network -Loss

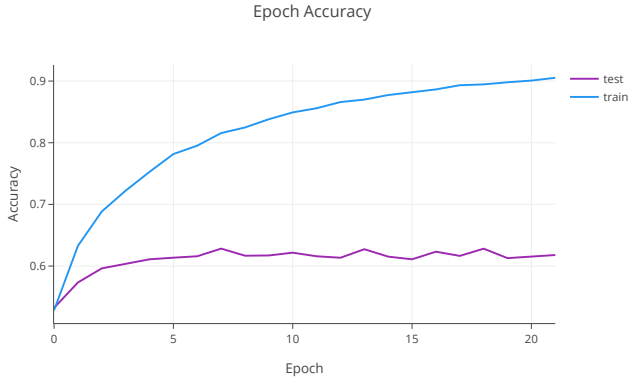

Fig. 39. Custom Neural Network -Accuracy

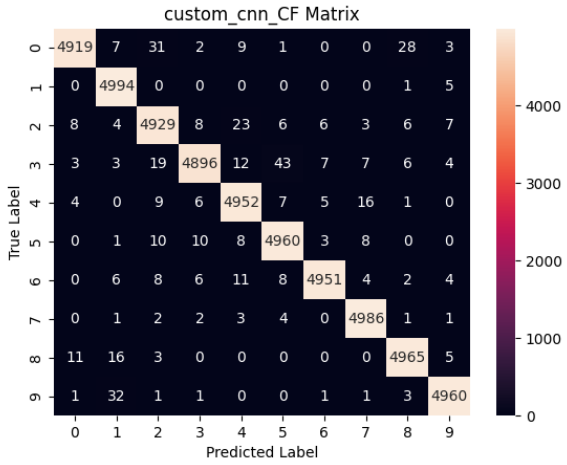

Fig. 40. Custom Neural Network - CF Matrix (Train)

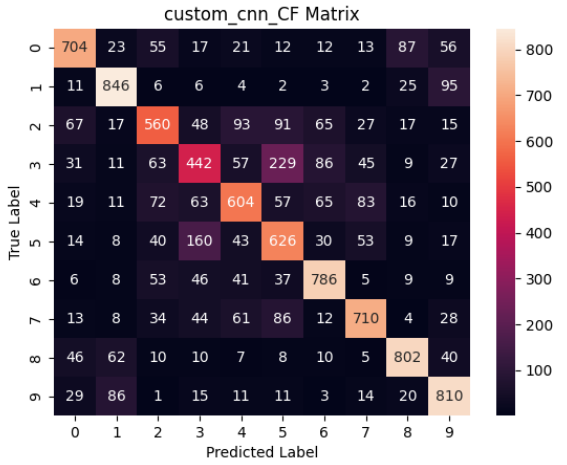

Fig. 41. Custom Neural Network -CF Matrix (Test)

The Loss and accuracy curves are shown in Fig.43 and Fig.44 respectively and Confusion Matrix - Figures (45 and 46).The train and test accuracy are 98.18% and 76.47%. This model is further trained and the best performance is indicated in Table I.

D. ResNext

ResNext builds on the foundations of ResNet and introduces a new concept of a cardinality parameter, which primarily controls the size of the set of the transformations a block can perform. This helps the model to be more efficient by enabling multiple paths for information flow within each block. A block of how a Residual mapping as a result of Cardinality looks like is shown in Figure 47. This is from the Custom ResNext layers implementation I did. The Loss and accuracy curves are shown in Fig.48 and Fig.49 respectively and Confusion matrix - Figures (50 and 51).The train and test accuracy are

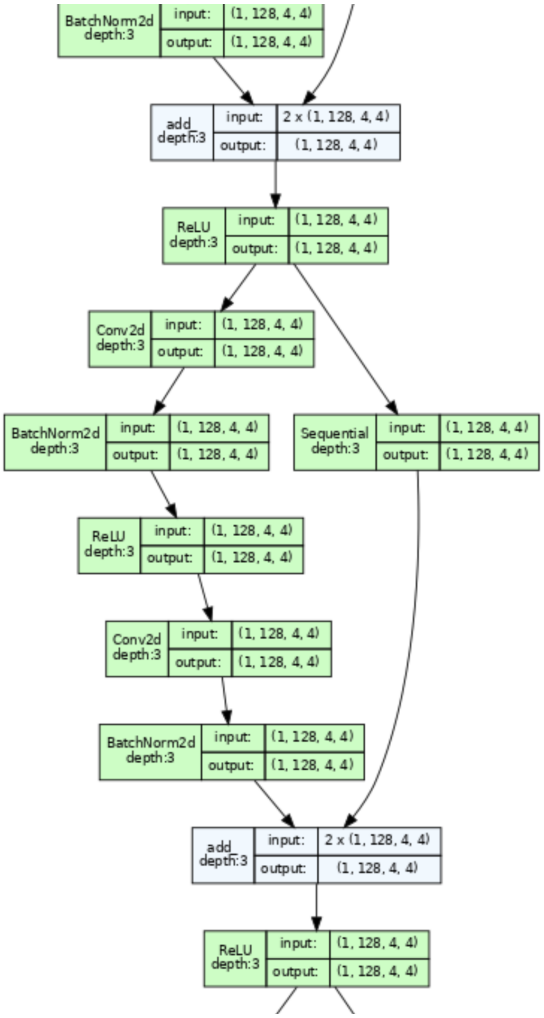

Fig. 42. ResNet34 - Sample ResNet Block

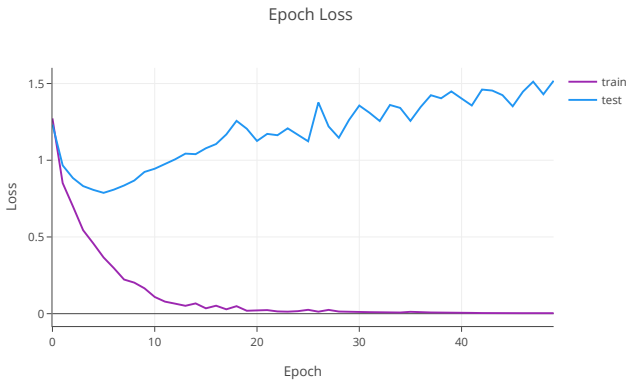

Fig. 43. ResNet34 -Loss

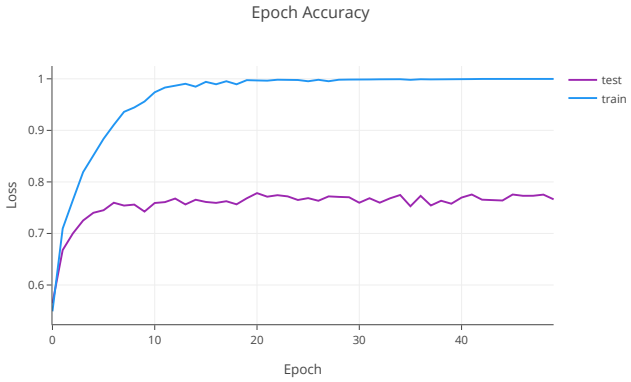

Fig. 44. ResNet34 -Accuracy

98.43% and 74.67%. This model is further trained and the best performance is indicated in Table I.

E. DenseNet 121

DenseNet works on the concept that each layer in the future receives feature maps from all preceding layers in a feedforward fashion. A block of Bottleneck layers and the connections are shown in Figure 52. The picture is from the DenseNet 121 architecture I have implemented. The Loss and accuracy curves are shown in Fig.53 and Fig.54 respectively and Confusion Matrix - Figures (55 and 56). The train and test accuracy are 95.996% and 87.14%. This model is further trained and the best performance is indicated in Table I.

F. MobileNet V2

Mobilenet V2 uses depthwise separable convolutions to reduce computation and parameters. This network is essentially light but having good performance when compared to ResNet not close but accurate. The primary advantage of using Mobilenet V2 is that it can be used on Resource-constrained devices. It also uses a concept called Inverted

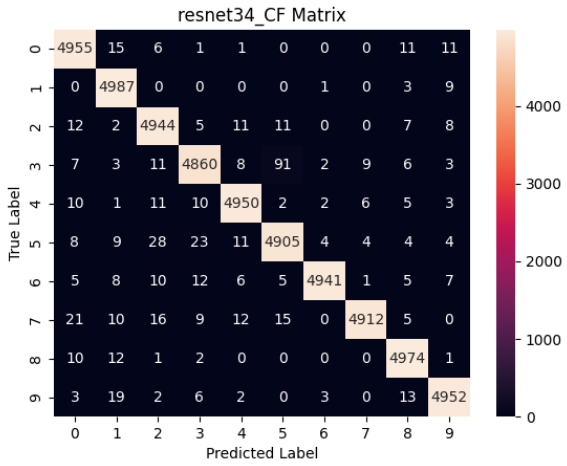

Fig. 45. ResNet 34 - CF Matrix (Train)

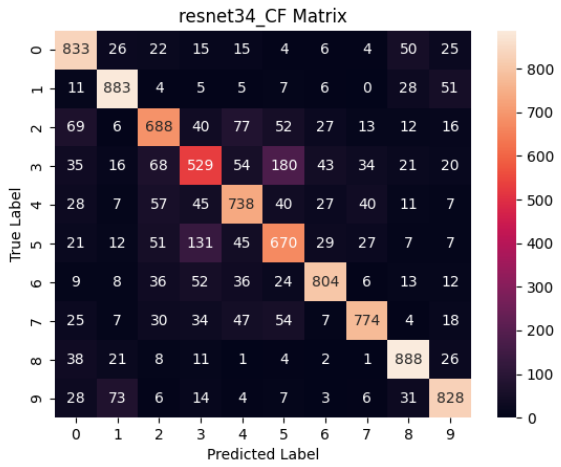

Fig. 46. ResNet 34 -CF Matrix (Test)

Residual networks which are a type of ResNet Implementation which have three primary concepts:

1) Linear Bottleneck: A light Iight linear layer that expands the number of channels
2) Depthwise Convolution: A depthwise separable convolution that captures spatial dependencies efficiently.
3) Projection: One more linear layer that reduces the number of channels back to the original size. The only difference between the Residual networks and Inverted Residual design is that the skips here pass through depthwise separable convolutions.

A sample block of Architecture of MobileNet v2 is shown in Figure 57. The Loss and accuracy curves are shown in Fig.58 and Fig.59 respectively and the Confusion Matrix - Figures (60 and 61).The train and test accuracy are 92.158% and 72.46%. This model is further trained and the best performance is indicated in Table I.

G. Comparison

The graphs which are shown above are for relative comparison of the variance of train and test accuracy and loss, some of the models I have trained overnight for good number of epochs generated decent accuracy which you can find below. Those checkpoints are shared along with the code. The models we have used are heavy as I wanted to test standard Architectures such as ResNet34, Variant of it as ResNext, DenseNet 121. The architectures that I made are generalized which can be changed to get different variants of the networks. For these networks, which I have trained please refer to the table below.

All the models are trained on RTX3080 16GB VRAM with i9 11th gen CPU (48GB RAM) and the inference is also performed on the same device.

IV. Conclusion

Phase I: Phase I involved dealing with a lot of traditional Image processing techniques to apply traditional filters on the image. One of the parts where I spent a lot of time

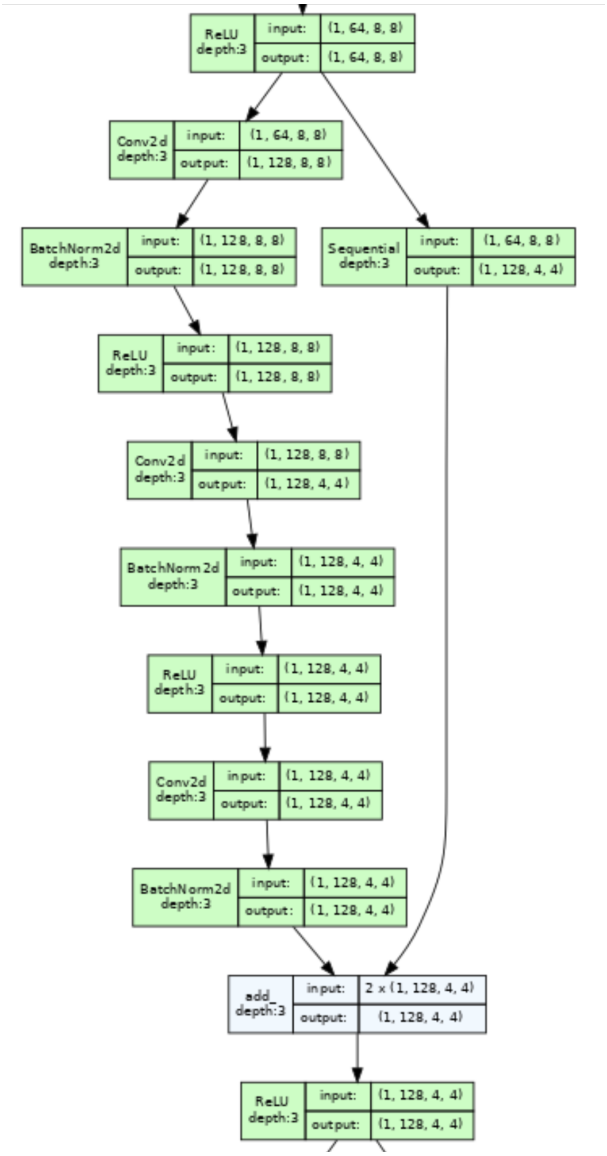

Fig. 47. ResNext - Sample ResNet Block

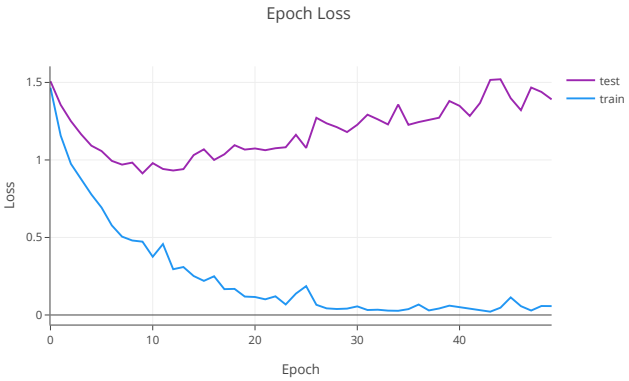

Fig. 48. ResNext -Loss

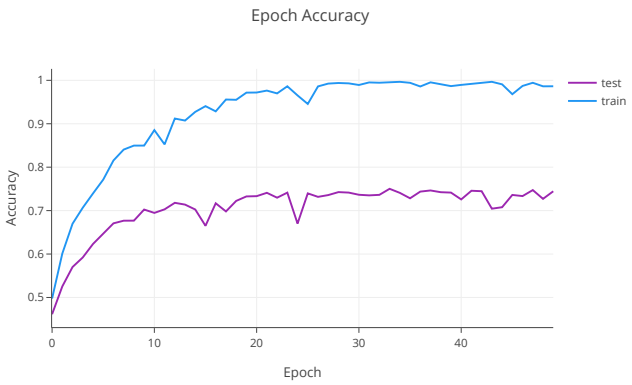

Fig. 49. ResNext -Accuracy

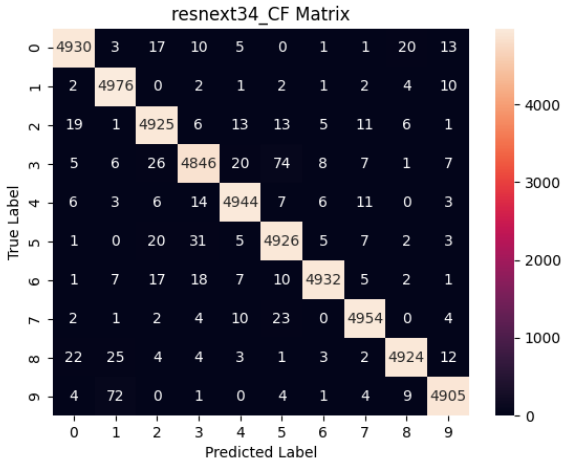

Fig. 50. ResNext - CF Matrix (Train)

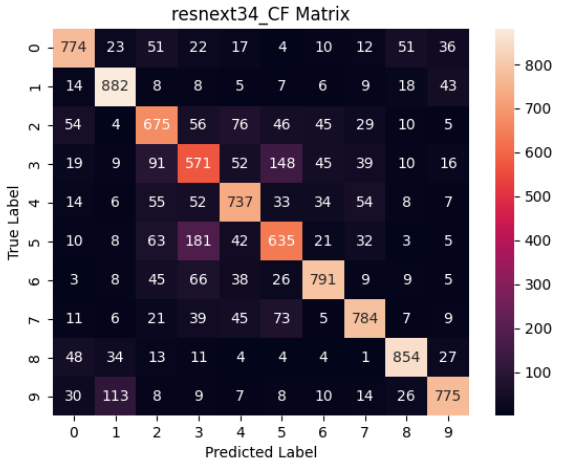

Fig. 51. ResNext -CF Matrix (Test)

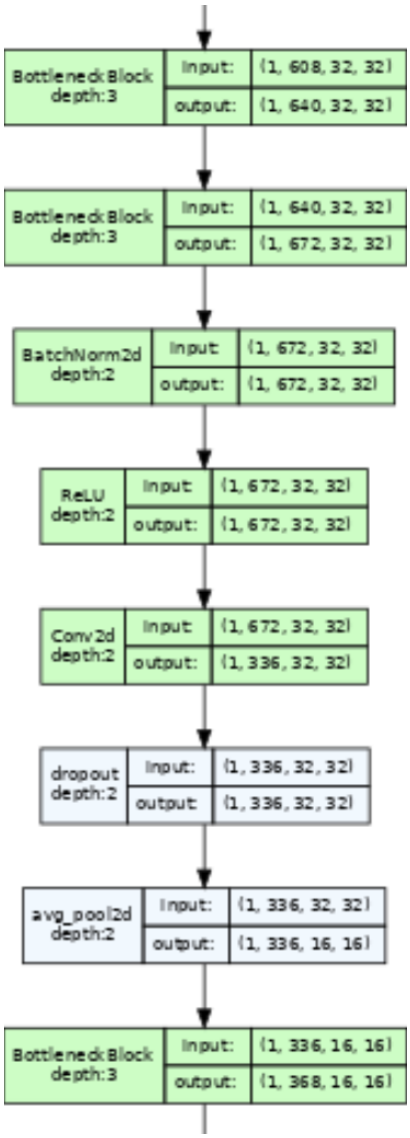

Fig. 52. DenseNet-Sample Bottleneck Block

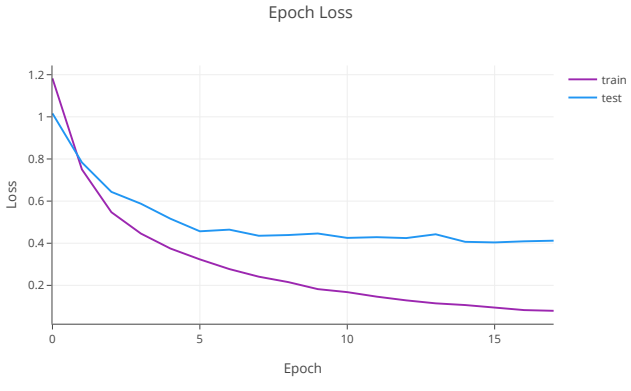

Fig. 53. DenseNet -Loss

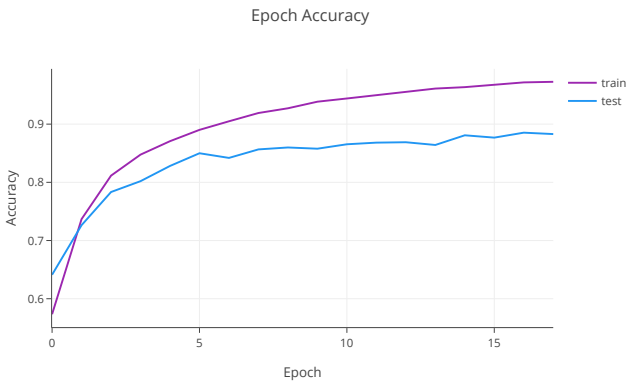

Fig. 54. DenseNet -Accuracy

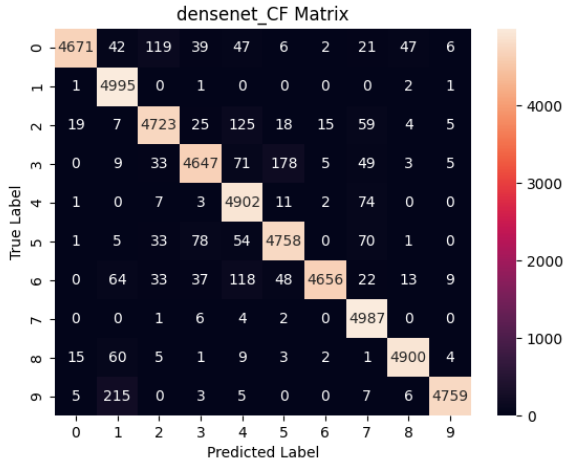

Fig. 55. DenseNet - CF Matrix (Train)

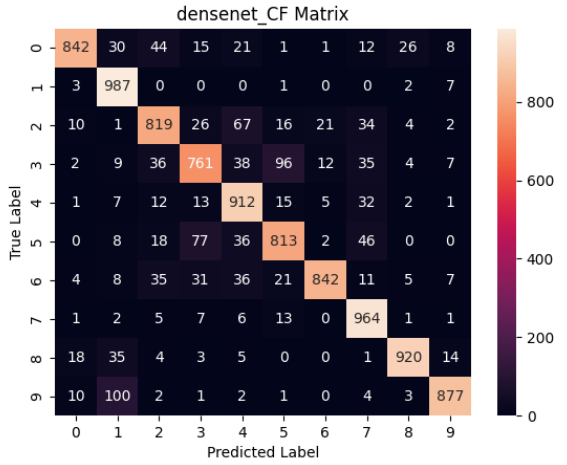

Fig. 56. DenseNet -CF Matrix (Test)

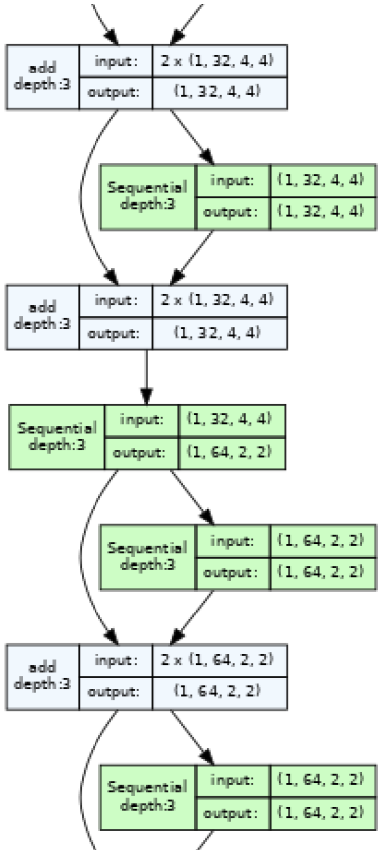

Fig. 57. MobileNet V2 Sample Block

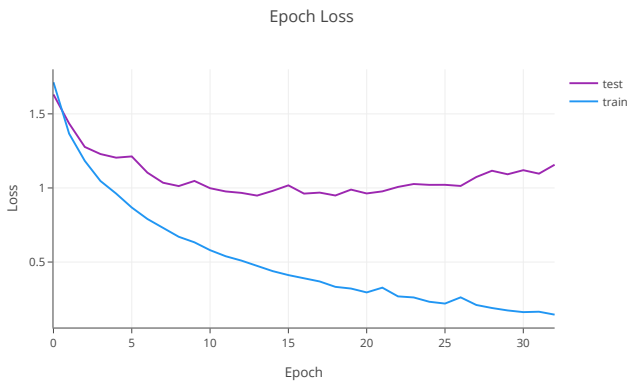

Fig. 58. MobileNetV2 -Loss

Model Name	Parameters	Accuracy (%)		Inference Time(ms)
		Train	Test	
Custom CNN	1061010	99.48	69.41	0.52
ResNet34	21289802	91.90	77.03	3.64
ResNext	2907722	97.938	72.43	3.91
DenseNet	7093050	99.75	90.48	11.75
MobileNetv2	2236682	89.31	67.66	3.56

TABLE I
BEST PERFORMANCE METRICS FOR DIFFERENT ARCHITECTURES

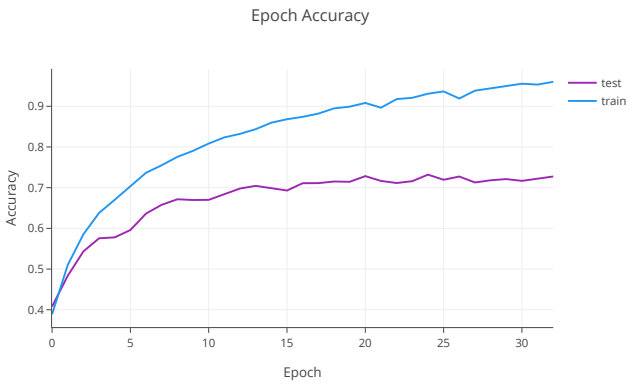

Fig. 59. MobileNetV2 -Accuracy

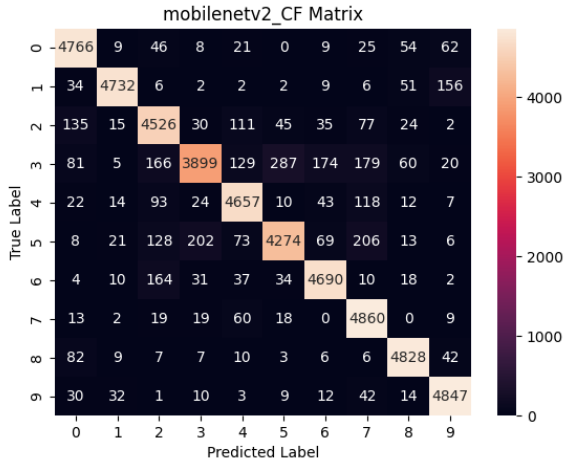

Fig. 60. MobileNet V2 - CF Matrix (Train)

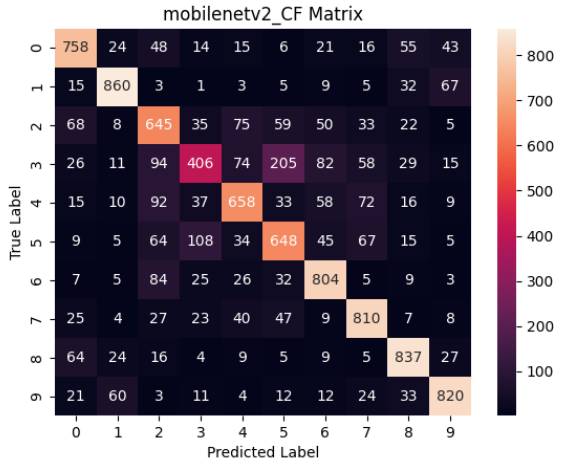

Fig. 61. MobileNet V2 -CF Matrix (Test)

Model Name	No. Parameters	Total Size(MB)	Params Size (MB)
Custom CNN	1061010	5.41	4.05
ResNet34	21289802	83.19	81.21
ResNext	2907722	13.96	11.09
DenseNet	7093050	389.52	27.06
MobileNetv2	2236682	11.68	8.53

TABLE II
SIZE OF THE MODELS

Hyperparam - Setting	
Optimizer	AdamW
LR Scheduler	Step
Learning Rate	1e-3
Weight Decay	5e-4
MiniBatchSize	64
Max Epochs Trained	50

TABLE III
HYPERPARAMETERS / SELECTIONS MADE FOR TRAINING

debugging is I was not able to figure out why the Gaussian first derivative or second derivative is taken along axis. Even though changing whether we want to take a First or Second derivative along both axes or different axis did not change the result much. But the filters that are shown in the reference [1] are not same as mine. But this was a learning as to how much impact changing these parameters can make.

Phase II: Phase II involved solving some of the issues with the code that are solvable. Going through the papers at least to have a skim through takes a bit of time, now making me realize to spend more time refreshing the concepts. On the Custom Convolution network and the other networks we can see there is a lot of overfitting. This can be further reduced by using Augmentation techniques and more better training techniques to choose the best hyperparameters. Currently, we used only Random Crop Augmentation which provided a bit better result in some cases.

ACKNOWLEDGMENT

I would like to thank Prof. Nitin J Sanket for giving this assignment as this was a lot of learning and the clean theory and instructions provided on the Course website.

REFERENCES

[1] Homework 0 - Course Website - https://rbe549.github.io/spring2024/hw/hw0/
[2] Arbeláez, P., Maire, M., Fowlkes, C. & Malik, J. Contour Detection and Hierarchical Image Segmentation. *IEEE Transactions On Pattern Analysis And Machine Intelligence*. **33**, 898-916 (2011)
[3] He, K., Zhang, X., Ren, S. & Sun, J. Deep Residual Learning for Image Recognition. (2015)
[4] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A. & Chen, L. MobileNetV2: Inverted Residuals and Linear Bottlenecks. (2019)
[5] Xie, S., Girshick, R., Dollár, P., Tu, Z. & He, K. Aggregated Residual Transformations for Deep Neural Networks. (2017)
[6] Huang, G., Liu, Z., Maaten, L. & Weinberger, K. Densely Connected Convolutional Networks. (2018)