Computer Vision Homework 0 - Alohomora

Kaushik Kavuri Subrahmanya
Robotics Engineering
Worcester Polytechnic Institute
Worcester, Massachusetts 01609
Email: ksubrahmanya@wpi.edu
Using 1 Late Day

***Abstract*—In the second phase, multiple neural network architectures were implemented to train on CIFAR-10 and their accuracies are compared. The neural networks include an initial simple neural network implemented and more effecient ResNet, ResNeXt, DenseNet architectures were implemented.**

I. Phase 1

In this phase, edges are detected using the pb-lite algorithm which is an upgrade over the standard Canny and Sobel algorithms. To achieve this better edge detection, we first construct various filter banks. Then we filter the images using these filter banks. Then we build texture, brightness and colour maps for each image. Then we take texture, brightness and colour gradients for each image. Finally, we combine these gradients with baseline edges obtained from from Canny and Sobel algorithms. We shall go through these steps with their outputs below.

A. Filter Banks

These filter banks help identify edges and boundaries by checking for rapid changes in intensity.

1) Oriented DoG Filter: These filters are created by convolving a simple Sobel filter and a Gaussian kernel and rotating the result. The filter bank looks as shown in fig.1

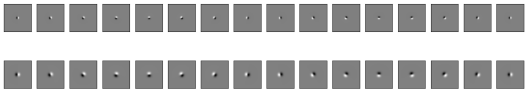

Fig. 1: Oriented DoG

2) Leung-Malik Filters: LM filter bank consists of 48 filters. They consist first and second order derivatives of Gaussians at 6 orientations and 3 scales for a total of 36. Other 12 consists 8 Laplacian of Gaussians and 4 Gaussians. The scales used for the filters are [1, $\sqrt{2}$, 2, $2\sqrt{2}$] for LM small, and [$\sqrt{2}$, 2, $2\sqrt{2}$, 4] for LM Large. The result is shown in fig.2,3

3) Gabor Filters: These filters are based on human visual system and they are obtained by modulating a Gaussian kernel by a sinusoidal plane wave. The result is as follows in fig.4

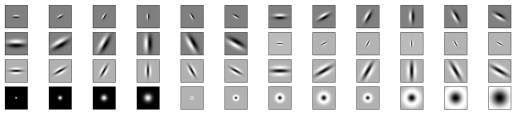

Fig. 2: LM Small

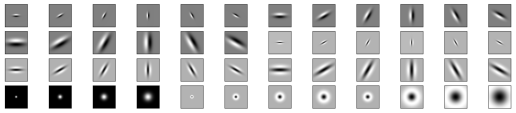

Fig. 3: LM Large

B. Texton, Brightness and Colour Maps

Here we use KMeans clustering to cluster the respective property of each image to create image maps. The results of this is presented in fig.5-15

C. Texton, Brightness and Colour Gradients

We use Half disk maps (shown in fig.15) to create colour gradients for each image. These maps help compute chi-square distances using a filtering operation which is faster than looping over each pixel neighborhood and aggregating counts for histograms.

The results of these are shown in fig. 16 - 25

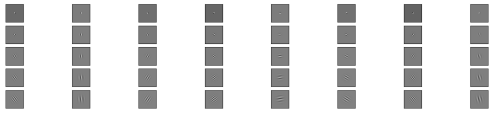

Fig. 4: Gabor Filter bank

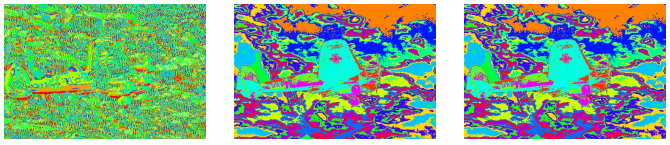

Fig. 5: $(\mathcal{T}, \mathcal{B}, \mathcal{C})$ for Image 1

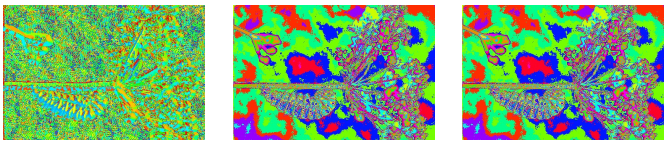

Fig. 6: $(\mathcal{T}, \mathcal{B}, \mathcal{C})$ for Image 2

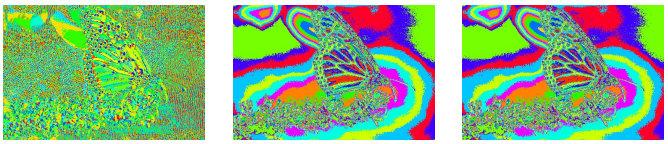

Fig. 7: $(\mathcal{T}, \mathcal{B}, \mathcal{C})$ for Image 3

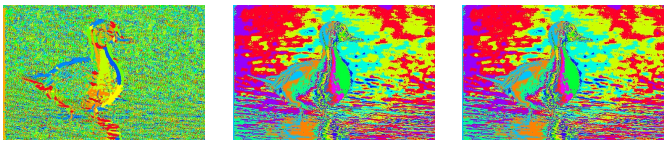

Fig. 8: $(\mathcal{T}, \mathcal{B}, \mathcal{C})$ for Image 4

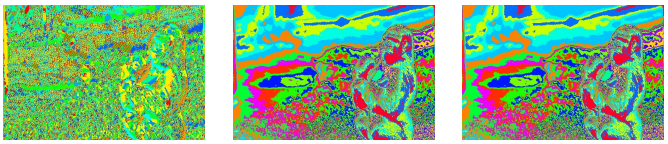

Fig. 9: $(\mathcal{T}, \mathcal{B}, \mathcal{C})$ for Image 5

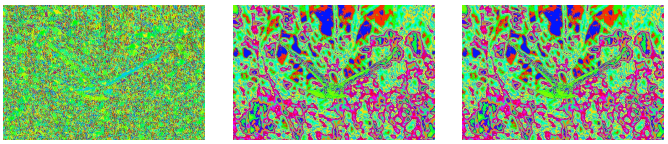

Fig. 10: $(\mathcal{T}, \mathcal{B}, \mathcal{C})$ for Image 6

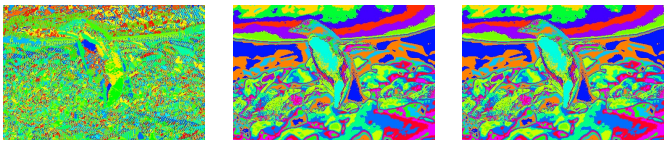

Fig. 11: $(\mathcal{T}, \mathcal{B}, \mathcal{C})$ for Image 7

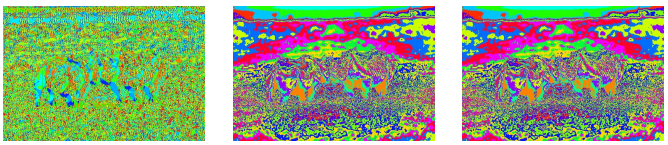

Fig. 12: $(\mathcal{T}, \mathcal{B}, \mathcal{C})$ for Image 8

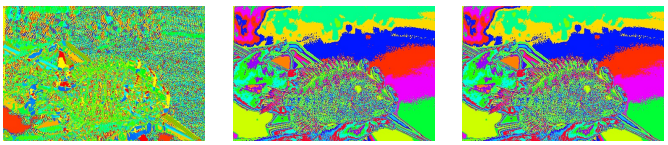

Fig. 13: $(\mathcal{T}, \mathcal{B}, \mathcal{C})$ for Image 9

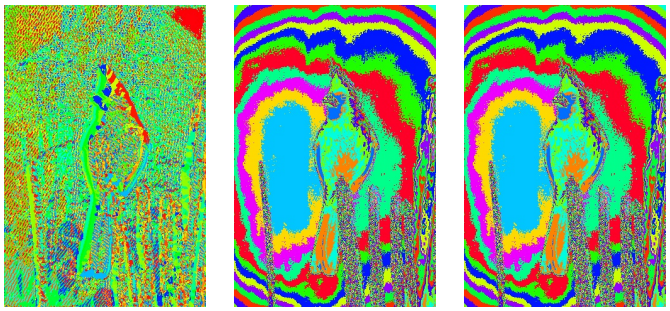

Fig. 14: $(\mathcal{T}, \mathcal{B}, \mathcal{C})$ for Image 10

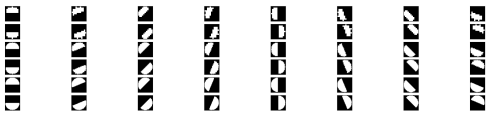

Fig. 15: Half Disk Maps

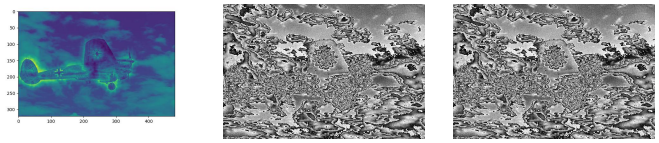

Fig. 16: $(\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g)$ for Image 1

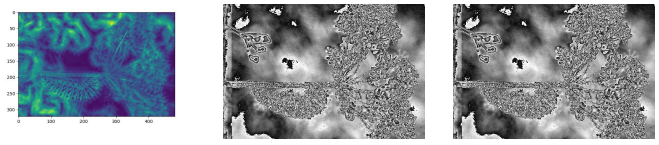

Fig. 17: $(\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g)$ for Image 2

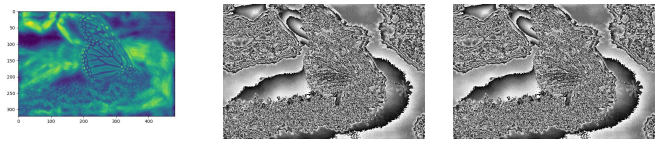

Fig. 18: $(\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g)$ for Image 3

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

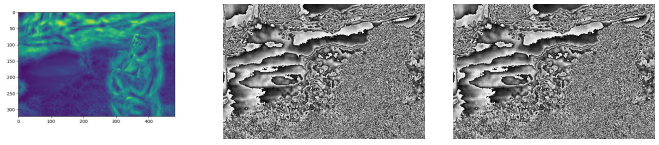

Fig. 20: $(\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g)$ for Image 5

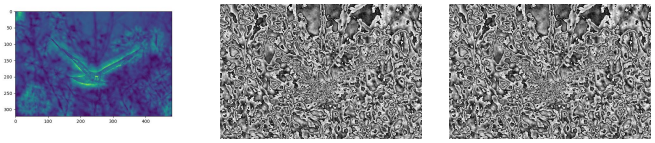

Fig. 21: $(\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g)$ for Image 6

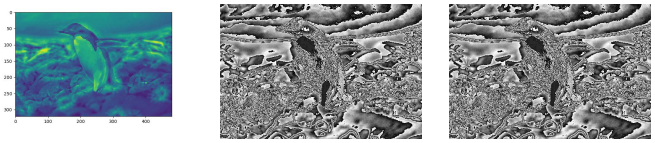

Fig. 22: $(\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g)$ for Image 7

D. PB Lite Output

Using and combining all the above filters, we obtain the following results in figs. 26-35

II. Phase 2

In the second phase of the homework, a simple neural network was implemented, and then, ResNet, ResNeXT, and DesnseNet neural network architectures were implemented. Accuracy and Loss of these architectures over Epochs are reported. The heat maps of the confusion matrix for each architecture's trained model are also reported.

A. Simple Neural Network

A simple neural network has been implemented with a single batch of sequential layers between Input and Output. The tensorboard snaphot of the network is shown in fig. Adamw optimizer and Batch normalization was used to optimize this network. The number of parameters in this model are 18,896,490, and the model has been trained for 50 epochs which took around 19 mins to train. The mini batch size used for training was 256.

1) Results: The trained model had an accuracy of 63% on the test data set and 99% on the trained data set.

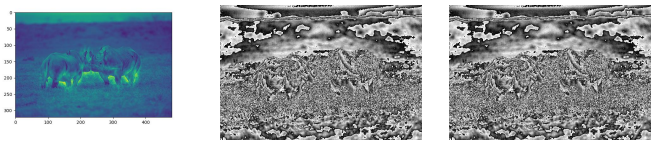

Fig. 23: $(\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g)$ for Image 8

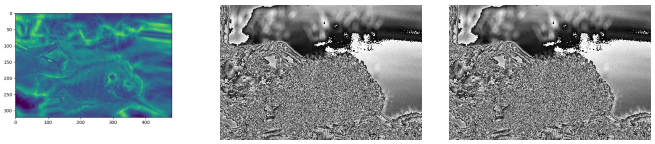

Fig. 24: $(\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g)$ for Image 9

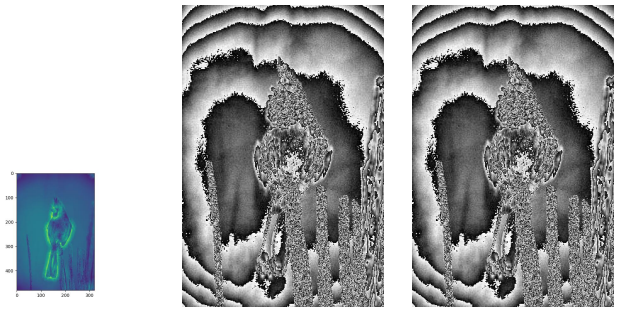

Fig. 25: $(\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g)$ for Image 10

Fig. 26: Canny, Sobel baselines and PB-Lite for Image 1

Fig. 27: Canny, Sobel baselines and PB-Lite for Image 2

Fig. 28: Canny, Sobel baselines and PB-Lite for Image 3

Fig. 29: Canny, Sobel baselines and PB-Lite for Image 4

Fig. 30: Canny, Sobel baselines and PB-Lite for Image 5

Fig. 31: Canny, Sobel baselines and PB-Lite for Image 6

Fig. 32: Canny, Sobel baselines and PB-Lite for Image 7

Fig. 33: Canny, Sobel baselines and PB-Lite for Image 8

Fig. 34: Canny, Sobel baselines and PB-Lite for Image 9

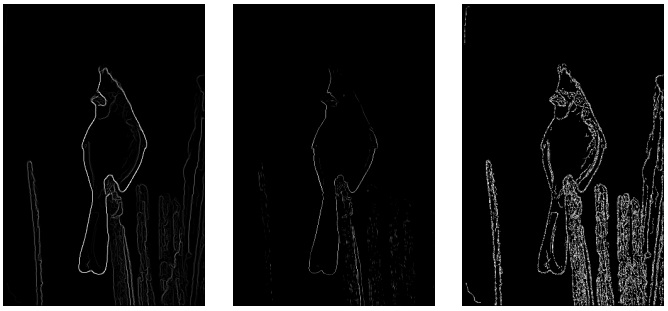

Fig. 35: Canny, Sobel baselines and PB-Lite for Image 10

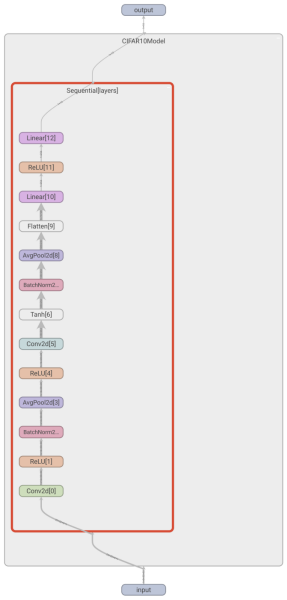

Fig. 36: Simple Neural Network

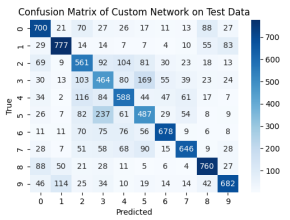

Fig. 37: Test CM

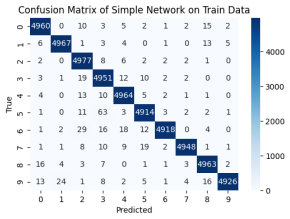

Fig. 38: Train CM

B. ResNet

Implemented the ResNet architecture. The loss, accuracy plots and the confusion matrix are shown in the figure.39-44

C. ResNext

Implemented the ResNext architecture. The loss, accuracy plots and the confusion matrix are shown in the figure.45-50

D. DenseNet

Implemented the ResNet architecture. The loss, accuracy plots and the confusion matrix are shown in the figure. 51-55

III. Conclusion

Here we detected edges and then trained neural networks to classify images using the above mentioned neural architectures

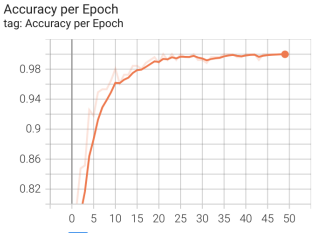

Fig. 39: training accuracy per epoch

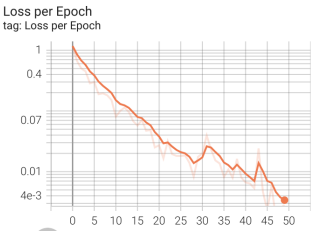

Fig. 40: loss per epoch

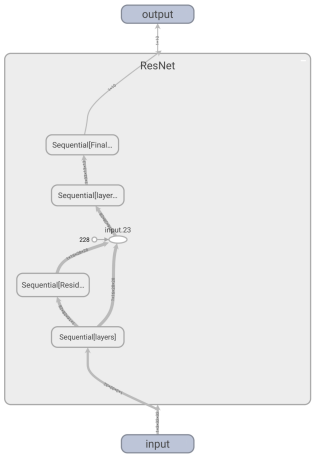

Fig. 41: ResNet Network

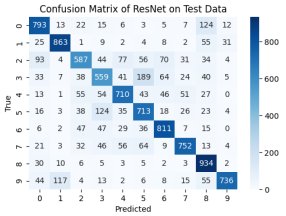

Fig. 42: Simple Neural Network

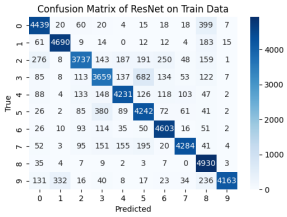

Fig. 43: Simple Neural Network

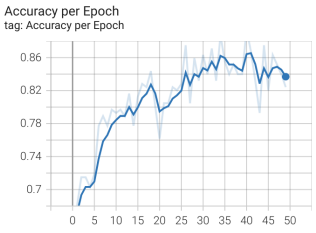

Fig. 44: training accuracy per epoch

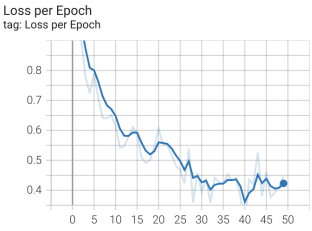

Fig. 45: loss per epoch

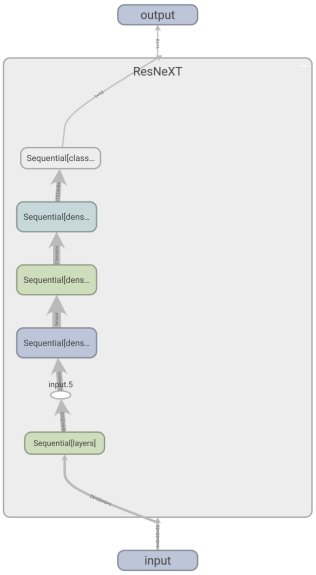

Fig. 46: ResNet Network

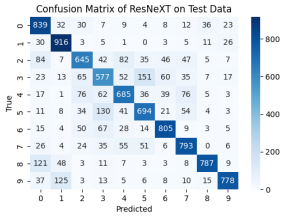

Fig. 47: Simple Neural Network

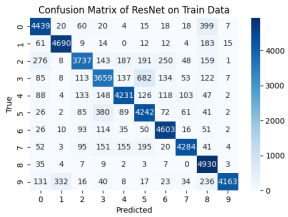

Fig. 48: Simple Neural Network

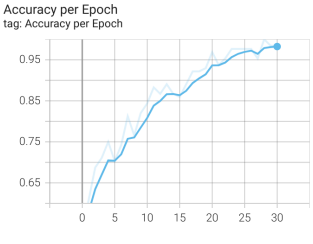

Fig. 49: training accuracy per epoch

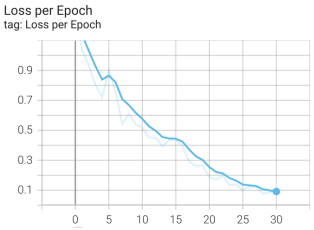

Fig. 50: loss per epoch

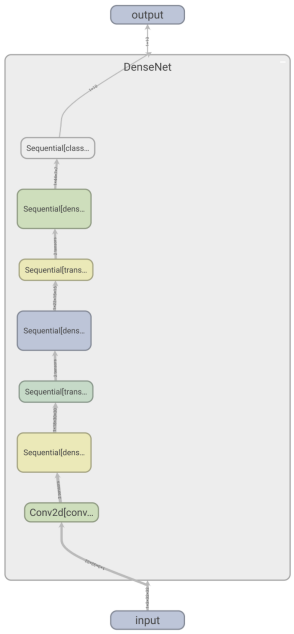

Fig. 51: DenseNet Network

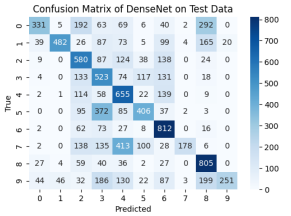

Fig. 52: DenseNet

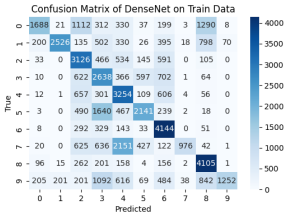

Fig. 53: DenseNet train

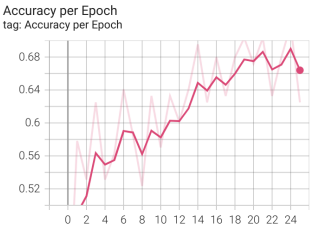

Fig. 54: training accuracy per epoch

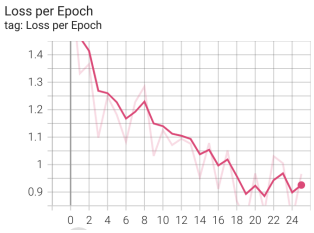

Fig. 55: loss per epoch

ACKNOWLEDGMENT

The authors would like to thank...

REFERENCES

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.

Model Name	No. Parameters	Total Size(MB)	Params Size (MB)
Custom CNN	18,896,490	75.73	72.08
ResNet	197274	3.47	0.75
ResNext	3270794	42.50	12.48
DenseNet	10832	3.45	0.04

TABLE I: Size of the Models