Homework 0 - Alohomora - CMSC733

Vinay Lanka
University of Maryland,
College Park, Maryland 20742
Email: vlanka@umd.edu

***Abstract*—This report outlines the results and analysis gained from completing Homework 0. Phase 1 of the homework focuses on boundary detection based on the Pb Lite (Probability of Boundary) algorithm. Phase 2 deals with an introduction to Deep Learning Architectures in Computer Vision, in the field of Image classification. We implement multiple popular architectures on the CIFAR10 Dataset.**

I. Phase 1: Boundary Detection

A. Introduction

Phase 1 deals with boundary detection, which is a very fundamental Computer Vision problem. We're first introduced to traditional Sobel and Canny edge detection algorithms, which look for change in intensities between pixels a gray-scale map or checking the similarity of the gradient at the pixel with neighbouring pixels.

These two algorithms only focus on intensity differences, and do not consider texture and colour changes between pixels. We now attempt to implement a simplified version of the Pb Algorithm, where we consider the texture, colour and brightness between pixels to assign a probability of pixel lying on an edge.

B. Pb Lite Edge Detection Algorithm

The Pb algorithm [1] outlines the various steps we need to follow to perform edge detection on an image. We first need to generate a set of filter banks which can capture various texture information in the image. We then compute the Texton, Brightness and Color maps with K-Means clustering. We then generate Texture, Brightness and Color gradient maps using half disk masks and chi square distances. Finally we combine the maps with a Sobel and Canny baseline and produce our final boundary detection result.

We now go over each step

1) Filter Bank Generation: The first step is to generate a set of filter banks to recover texture information. This step only needs to be done once and they are reused for each image. They are generated with different sizes, orientations and characteristics to identify various types of textures. We generate 3 types of filter banks in this version of the algorithm.

a) Oriented DoG Filters: The Oriented DoG filters, as the name suggests, is the first derivative of the 2D Gaussian function. This can be achieved with the convolution of the Gaussian kernel with a Sobel filter and rotate the result to the desired orientations. For this algorithm, 2 scales and 16 orientations were used, creating a total of 32 filters.

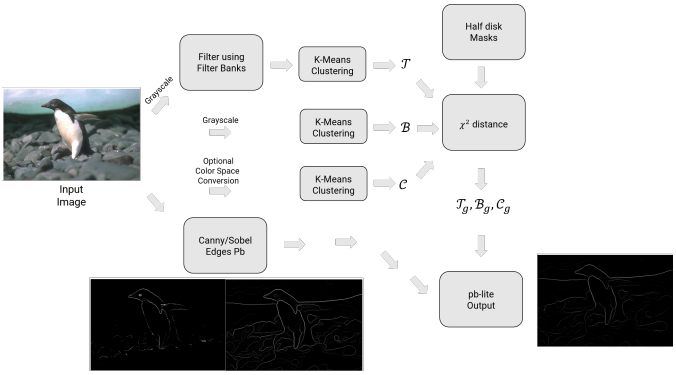

Fig. 1. Overview of the Pb Lite Algorithm

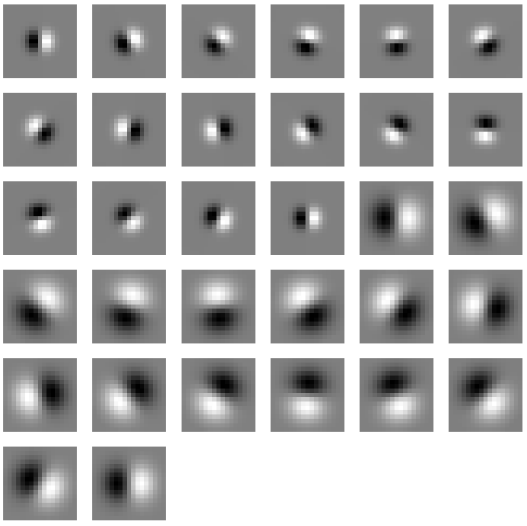

Fig. 2. Derivative of Gaussian Filter Bank - with 2 different scales and 16 different orientations for each scale

b) Leung-Malik Filters: There are 2 * 48 Leung-Malik Filters generated across 4 scales of various orientations. There are 4 scales of Gaussian filters. To get the LoG filter, we convolve the Laplacian filter with the Gaussian kernel in the

various scales. We also generate first and second derivative of the Gaussian kernel for the first 3 scales with different standard deviations across the X and Y axes. All these filters have 8 different orientations generating a total of 48 filters.
We generate 2 separate filter banks with 2 scales, LM small and LM large.

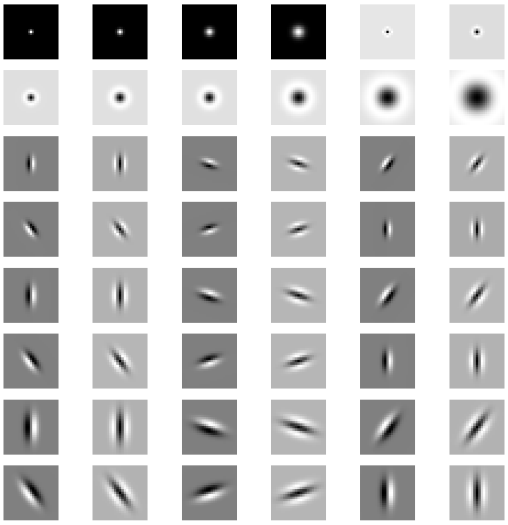

Fig. 3. Leung-Malik Filter Bank Small - with 4 different scales and 8 different orientations for each first and second derivative. Standard deviation values of 1, $\sqrt{2}$, 2 and $2\sqrt{2}$ were used

c) Gabor Filters: The Gabor filter is obtained by modulating a Gaussian filter with a sinusoidal wave. We can obtain this by multiplying the Gaussian function with a sinusoidal wave equation. The Gabor filters generated are generated with 2 scales, 3 frequencies and 6 orientations each, generating a total of 36 filters.

2) Texton, Brightness, and Color Maps: After obtaining the filter bank with all the filters, we use them on the input image. The texton map encodes the texture information of the pixels in the image, and the brightness and color maps encode intensity and color values respectively. There is a dimensionality reduction performed using the k-means clustering algorithm. The Texton, Brightness and Color maps are shown for each image in Figure 5.

3) Texton, Brightness, and Color Gradients: To compute the gradient maps for texture, brightness and color, we use half disk masks along with the chi-square distance. The half disk masks are generated in different scales by setting values that lie in a radius as 1 and the rest 0. This generates a half disk which an then be rotated to produce various orientations. We use 5 scales with 6 orientations. These are generated in opposite pairs as shown in Figure 7.

We then apply this opposite masks and then compute the difference between the results with a chi-square distance. This gives a gradient map for all texture, brightness and color information. This is shown in Figure 6.

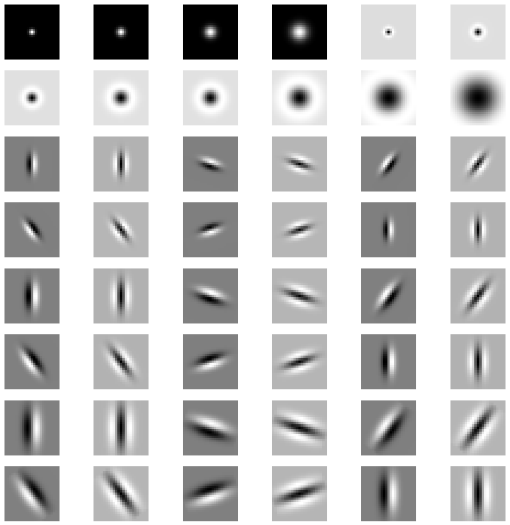

Fig. 4. Leung-Malik Filter Bank Large - with 4 different scales and 8 different orientations for each first and second derivative. Standard deviation values of $\sqrt{2}$, 2, $2\sqrt{2}$ and 4 were used

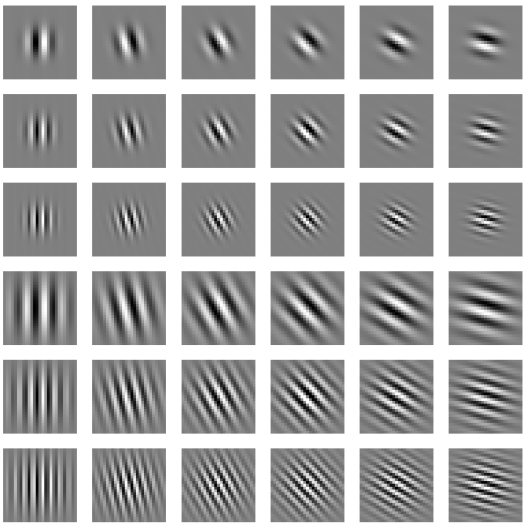

Fig. 5. Gabor Filter Bank - with scales [5,10] and 6 different orientations, generated by frequencies [4,6,8]

4) Pb Lite Boundary Detection: We finally find the boundaries by combining information from the features obtained with baselines based on the Sobel and Canny methods using

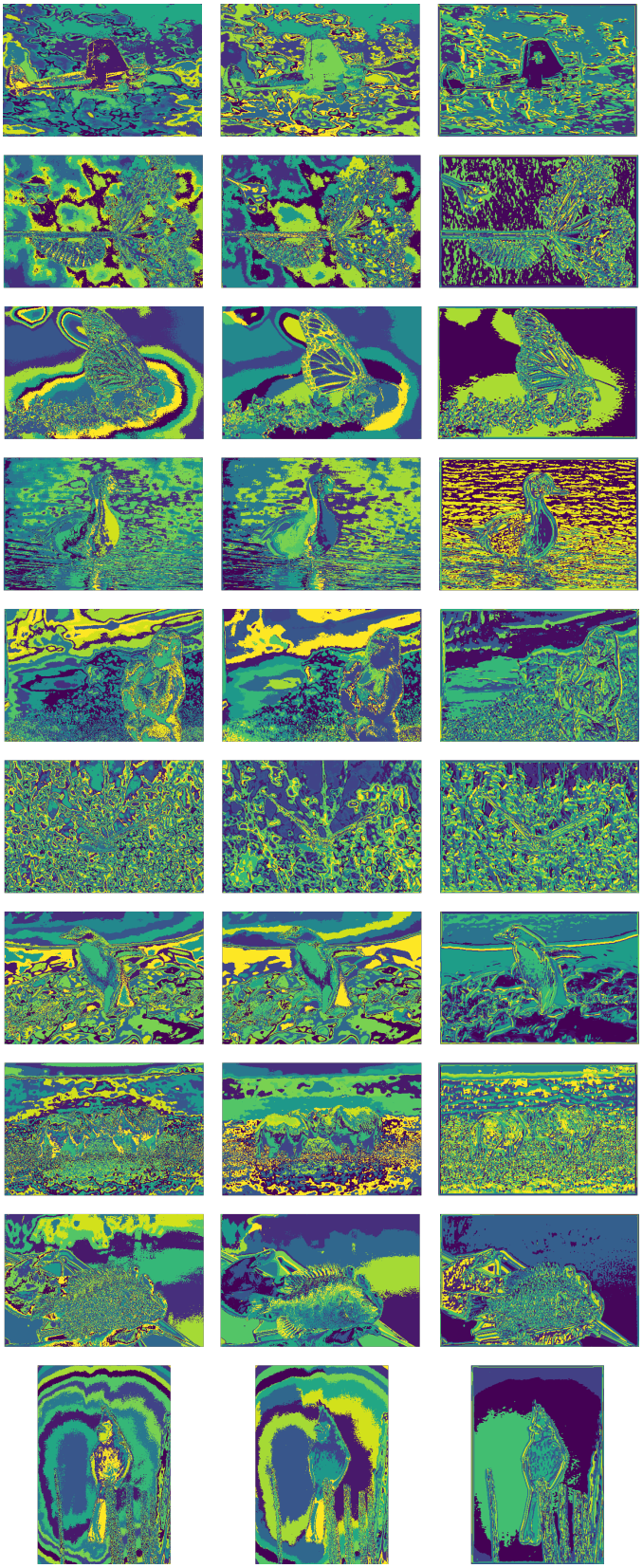

Fig. 6. Brightness, Color and Texton Maps

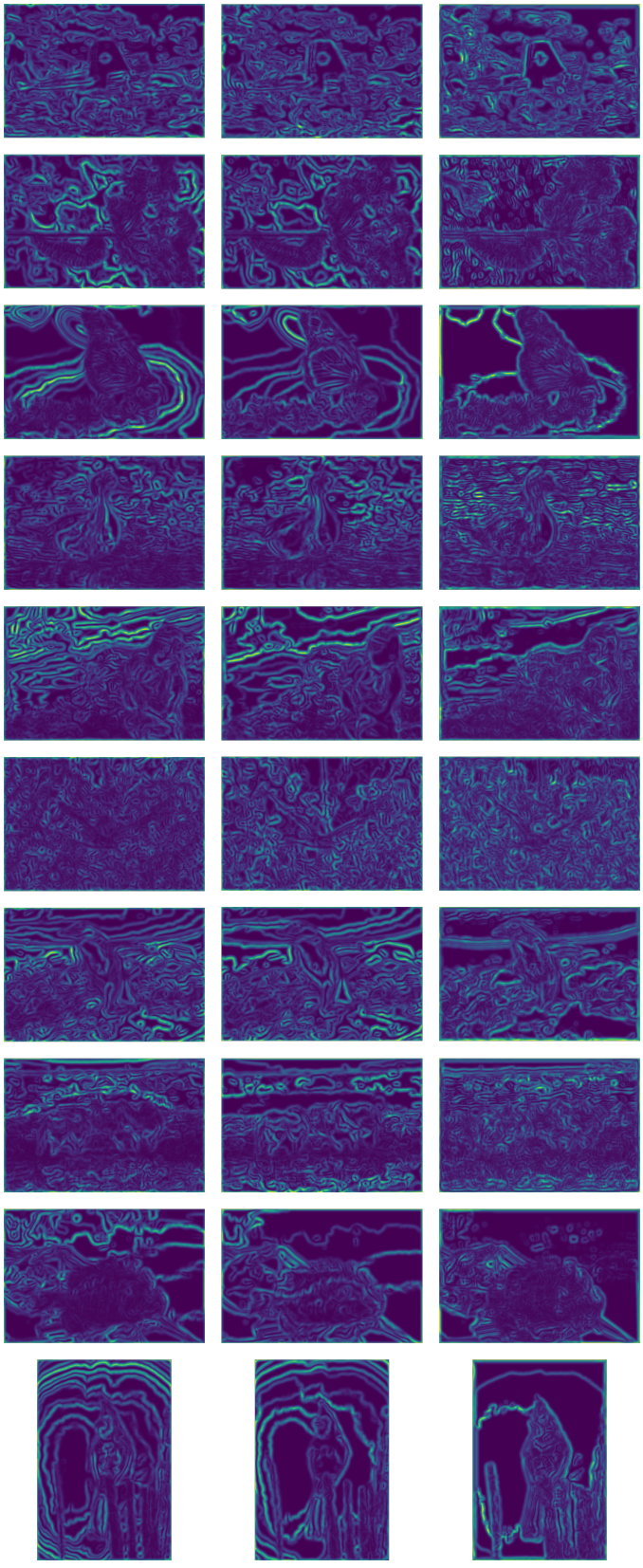

Fig. 7. Brightness, Color and Texton Gradients

Fig. 8. Half Disk Masks generated with scales 2,4,6,8 and 10

the formula.

$$PbLiteEdges = ((T_g + B_g + C_g)/3) \odot (w_1 * CannyBaseline + w_2 * SobelBaseline) \quad (1)$$

The $\odot$ is the element wise multiplication of the arrays, and the w_1 and w_2 is the weights associated with the Canny and Sobel baselines which add up to 1.

The Pb Lite outputs are shown in the Figures 8 - 10.

Fig. 9. Pb Lite outputs for images 1-4

II. Phase 2: Deep Learning Introduction

A. Introduction

Phase 2 deals introduces us to Deep Learning Architectures for Image classification. The Classification models use the CIFAR10 dataset, with 60,000 images belonging to 10 classes. We first start out with a baseline Convolutional Neural Network and then improve it by taking inspiration from various architectures like ResNet, ResNeXt, and DenseNet.

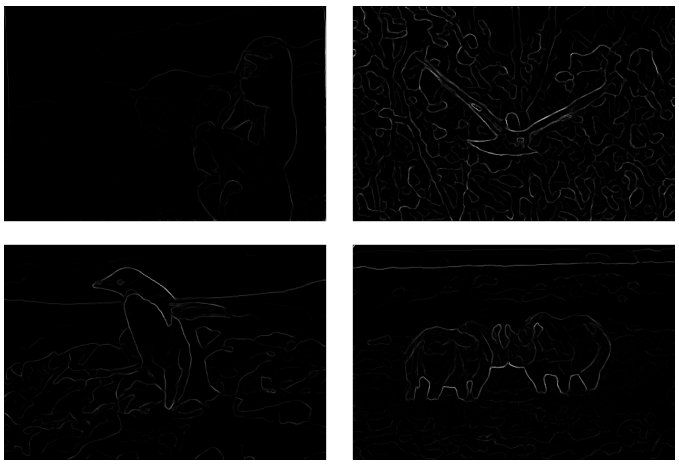

Fig. 10. Pb Lite outputs for images 5-8

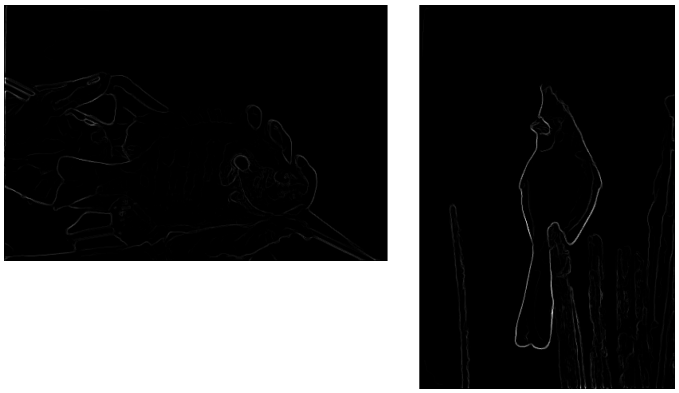

Fig. 11. Pb Lite outputs for images 9-10

Architectures, hyper-parameters, data-augmentations, accuracy and loss plots, and confusion matrices are shown for each model implemented.

1) Simple Neural Network: To establish a baseline, a CNN inspired by the LeNet-5 [4] was implemented. It consists of 2 convolution layers with MaxPooling in between followed by 2 fully connected layers. The last layer is a soft-max output layer. This architecture did not use any data standardization or any augmentation. Containing 315722 parameters, this network was trained over 25 epochs and got a 95.5% accuracy over the training data and a 61.3% over the validation or testing set.

Base Network Summary

Parameters: 315722
Epochs: 25
Train Accuracy: 95.51
Test Accuracy: 61.36
Optimizer: Adam Optimizer
Learning Rate: 0.001

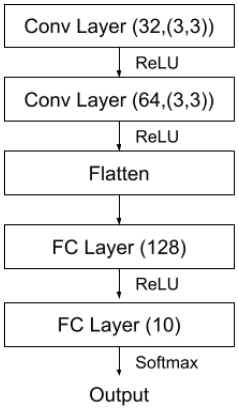

Fig. 12. Architecture of the Simple Neural Network

Batch Size: 64

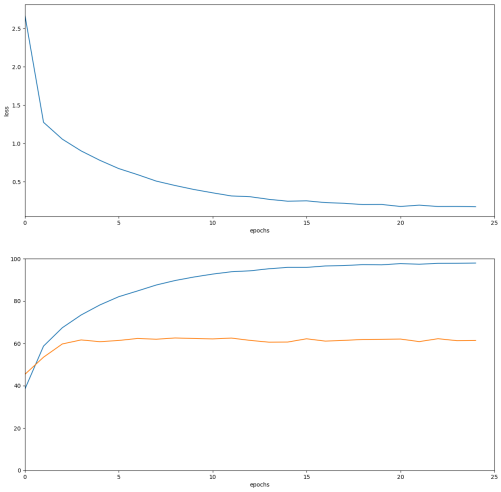

Fig. 13. Training Loss and Training and Test Accuracy Plot - Simple Neural Network

The Test/Train Accuracy, Training loss over epochs, Test/Train Confusion Matrices for the Simple Neural Network are plotted in figures 13-15.

The Softmax Cross Entropy loss function was used as it was a good choice for the classification task. The Adam optimizer was used with a learning rate of 0.001. Observations show that this network was very shallow and overfit very easily to the training data.

2) Modified Neural Network: With just a few tweaks to the architecture and some data augmentation, we get much better results. We now standardize the data, by re-scaling the distribution of values to obtain a mean of 0 and a standard deviation of 1. Due to restrictions with the starter code being written in TensorFlow 1, under the compatibility layer, we

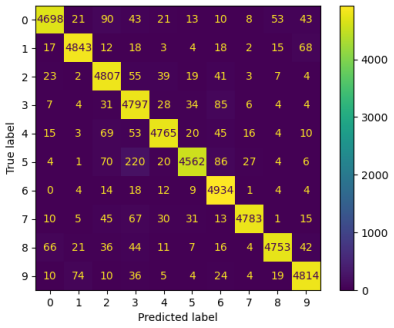

Fig. 14. Confusion Matrix Train - Simple Neural Network

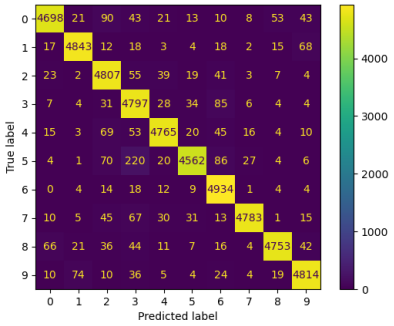

Fig. 15. Confusion Matrix Test - Simple Neural Network

could not perform decent data augmentation techniques using tf.image . Still we did implement random horizontal image flipping from scratch.

To the architecture, we add 2 more convolution layers, bringing the total to 4 convolution layers, with 2x32 filters and 2x64 filters. Between each layer we include the ReLU activation function (Not shown in figure). We also perform batch normalisation, a technique to reduce internal co-variate shift to ensure roughly the same distribution every training step. This technique has worked extremely well. We also add dropout after every fully connected layer. We use the Adam optimizer and the Softmax Cross Entropy loss function. The compatibility layer also meant we could not use good learning rate decay, although we did tweak the β_1 and β_2 values to 0.9 and 0.99 respectively.

Modified Network Summary

Parameters: 509994
Epochs: 15
Train Accuracy: 93.52
Test Accuracy: 78.26
Optimizer: Adam Optimizer
Learning Rate: 0.001
Batch Size: 64

The Test/Train Accuracy, Training loss over epochs, Test/Train Confusion Matrices for the Modified Neural Net-

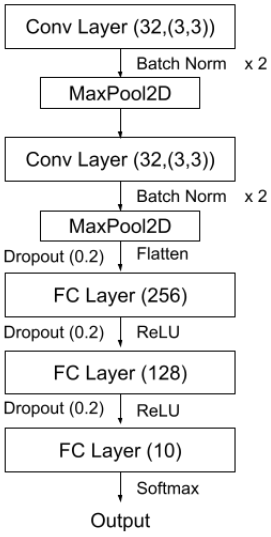

Fig. 16. Architecture of the Modified Neural Network

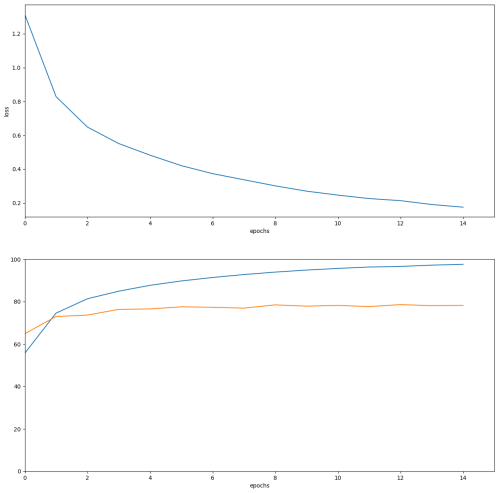

Fig. 17. Training Loss and Training and Test Accuracy Plot - Modified Neural Network

work are plotted in figures 17-19. We find that this Modified Neural Network performs the best, with a validation accuracy of 78.2%. This is mainly credited to the network being slightly deeper with batch normalisation and dropout.

3) ResNet: The ResNet paper [2] reports that it works very well over the CIFAR10 dataset. We do not implement the exact ResNet architecture, but take inspiration and build our own mini Residual block. The Residual block is shown in the Fig. It consists of a skip connection to solve the problem of the vanishing gradient when the gradients flow backward while training the model.

Our Residual block implementation consists of a convolution layer directly being added to a set of 3 convolution

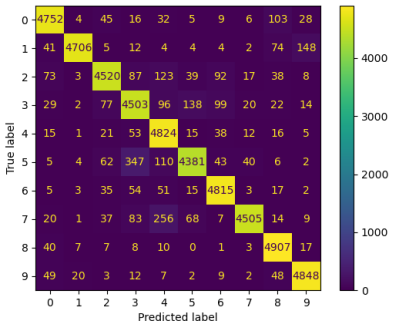

Fig. 18. Confusion Matrix Train - Modified Neural Network

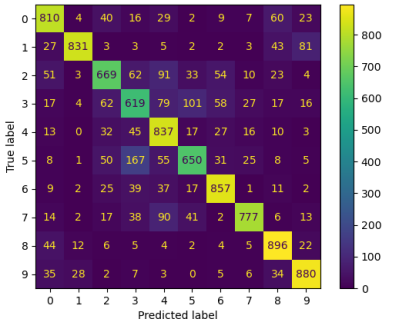

Fig. 19. Confusion Matrix Test - Modified Neural Network

layers with ReLU in between them. Each path has a batch normalisation layer in between layers.

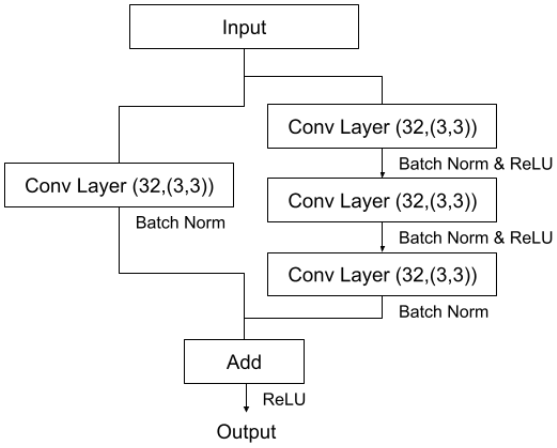

Fig. 20. ResNet inspired residual block

ResNet Network Summary

Parameters: 8518090
Epochs: 15
Train Accuracy: 97.85
Test Accuracy: 69.47

Optimizer: Adam Optimizer
Learning Rate: 0.001
Batch Size: 64

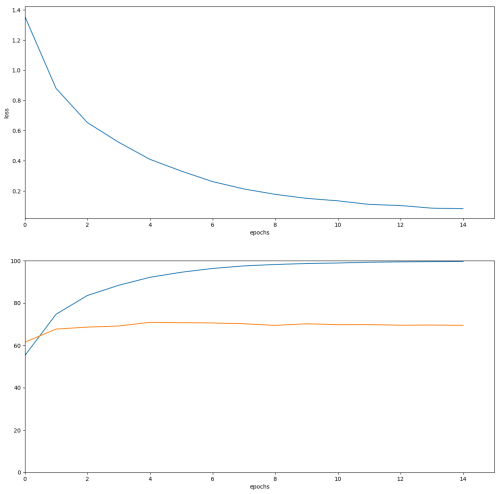

Fig. 21. Training Loss and Training and Test Accuracy Plot - ResNet Inspired Network

The Test/Train Accuracy, Training loss over epochs, Test/Train Confusion Matrices for the ResNet Model are plotted in figures 21-23. We find that this ResNet inspired Network performs poorly, with a validation accuracy of 69.47%. The performance doesn't improve that much in the validation set and plateaus as the training accuracy just begins to start overfitting the train set.

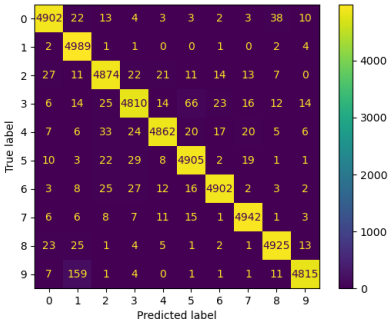

Fig. 22. Confusion Matrix Train - ResNet Inspired Network

The Model consists of 3 such residual blocks, and 3 fully connected layers. This implementation is extremely heavy with over 8 million parameters, which explains the sub-optimal performance. We use the Adam optimizer and the Softmax Cross Entropy loss function.

4) ResNeXt: The ResNeXt architecture [5] builds upon the ResNet by introducing the concept of cardinality. Multiple

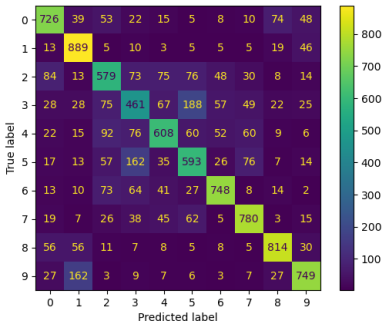

Fig. 23. Confusion Matrix Test - ResNet Inspired Network

blocks of smaller filter sizes are added in parallel, and the outputs of these parallel units are added in parallel. The paper says that increasing cardinality instead of making the network deeper has a greater effect on accuracy.

We take inspiration from the ResNeXt block by initially placing a convolution layer and adding it's output at the end. Now we implement 8 parallel convolution paths of 2 layered 4 filter convolution blocks. Each layer in this network has batch normalisation and a ReLU, except for the layers facing the addition block, for which the non-linearity is applied after addition.

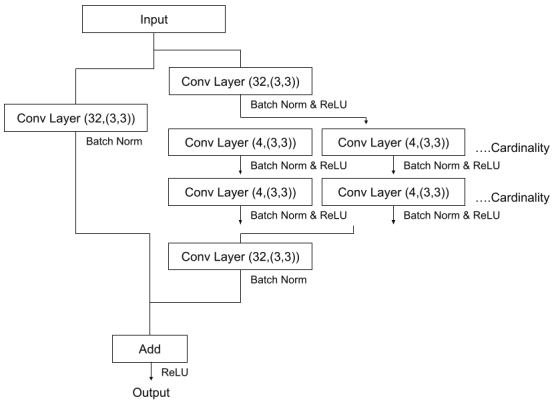

Fig. 24. ResNeXt inspired residual block

ResNeXt Network Summary
Parameters: 8520650
Epochs: 15
Train Accuracy: 97.77
Test Accuracy: 68.53
Optimizer: Adam Optimizer
Learning Rate: 0.001
Batch Size: 64

The Test/Train Accuracy, Training loss over epochs, Test/Train Confusion Matrices for the ResNeXt Model are plotted in figures 25-27. We find that this ResNeXt inspired Network performs poorly, with a validation accuracy

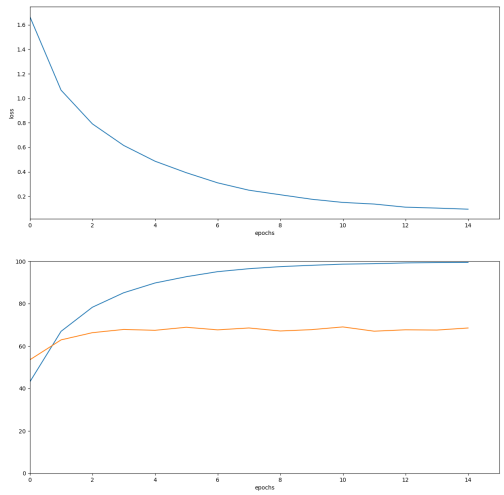

Fig. 25. Training Loss and Training and Test Accuracy Plot - ResNeXt Inspired Network

of 68.53%. The performance doesn't improve that much in the validation set and plateaus early as the training accuracy just begins to start over-fitting the train set.

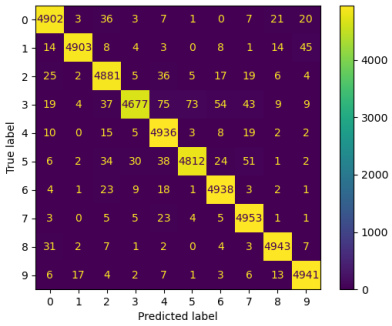

Fig. 26. Confusion Matrix Train - ResNeXt Inspired Network

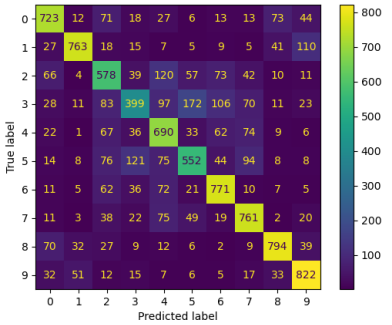

Fig. 27. Confusion Matrix Test - ResNeXt Inspired Network

The Model consists of 2 such residual blocks, and 2 fully connected layers. Like the ResNet implementation, this is also extremely heavy with over 8 million parameters, which explains the sub-optimal performance. We use the Adam optimizer and the Softmax Cross Entropy loss function.

5) DenseNet: With knowledge from the previous networks, we can conclude that the accuracy of networks can be improved by making shorter connections skipping blocks to keep integrity till the output. DenseNet [3] takes the same approach with each block feeding the following blocks ahead of it. The output is a concatenation of all the previous layers and ends the DenseNet block.

Our DenseNet inspired inception block contains 4 parallel connections, the input is fed to a 1x1 convolution layer and a MaxPool2D layer. The output of the 1x1 convolution layer leads to the concatenation and fed forward to the 3x3 and 5x5 convolution layers. We do not perform batch normalisation in the block. Every convolution layer output is fed to the ReLU activation function and finally all the outputs are concatenated and given as the output.

The DenseNet also consists of a transition block which lies in between sucessive inception blocks and performs a convolution and a MaxPool operation.

The Model consists of 2 such inception blocks, separated by 2 transition blocks all with filter size 32. It is then followed by 2 fully connected layers. It's quite a shallow network which defeats the purpose of the DenseNet so that explains the suboptimal performance. We also did not implement any growth rate as its such a shallow network taking only the CIFAR's 32x32 as an input. This model is just meant to show the use of the DenseNet inspired block. We use the Adam optimizer and the Softmax Cross Entropy loss function.

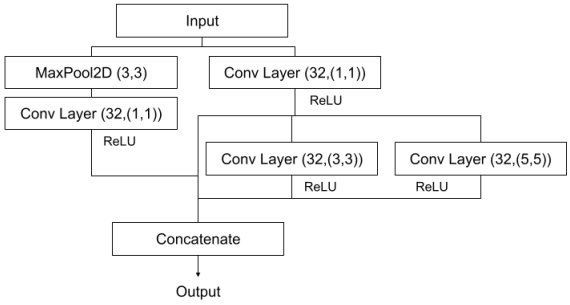

Fig. 28. DenseNet inspired inception block

DenseNet Network Summary

Parameters: 607498
Epochs: 15
Train Accuracy: 94.91
Test Accuracy: 75.94
Optimizer: Adam Optimizer
Learning Rate: 0.001
Batch Size: 64

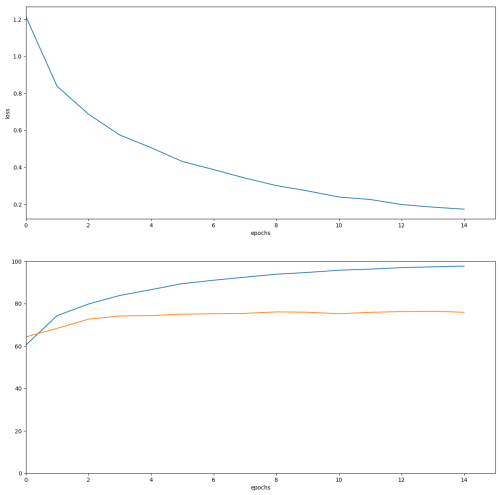

Fig. 29. Training Loss and Training and Test Accuracy Plot - DenseNet Inspired Network

The Test/Train Accuracy, Training loss over epochs, Test/Train Confusion Matrices for the DenseNet Model are plotted in figures 29-31. We find that this ResNet inspired Network performs pretty decently, with a validation accuracy of 75.94%. The performance actually was higher and dropped at the final few epochs, probably due to over-fitting.

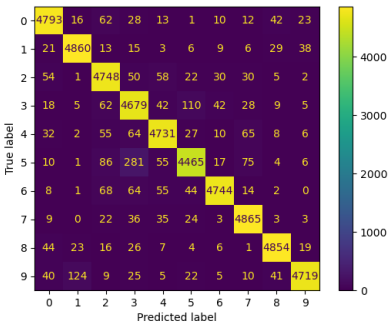

Fig. 30. Confusion Matrix Train - DenseNet Inspired Network

6) Summary: The Models are all summarised in Table 1. We see that the Modified Network performs the best as its simple and straightforward with neat tricks that help it's accuracy. The ResNet and ResNeXt implementations that were made in this paper were unnecessarily heavy and weren't optimized and were just to test out the blocks. The DenseNet performs decently but would perform better if it was a deeper network.

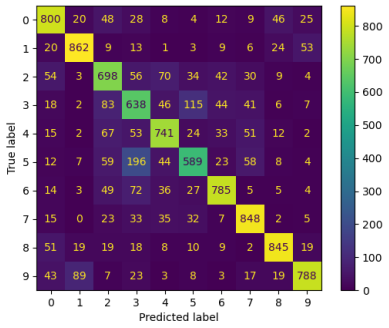

Fig. 31. Confusion Matrix Test - DenseNet Inspired Network

Model	Parameters	Train Acc	Test Acc
BaseModel (25 ep)	315722	95.512	61.36
Modified	509994	93.522	78.26
ResNet	8518090	97.85	69.47
ResNeXt	8520650	97.77	68.53
DenseNet	607498	94.91	75.94

TABLE I
SUMMARY OF ALL NETWORKS.

REFERENCES

[1] Pablo Arbeláez, Michael Maire, Charless Fowlkes, and Jitendra Malik. Contour detection and hierarchical image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(5):898–916, 2011.

[2] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2015.

[3] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016.

[4] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[5] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *CoRR*, abs/1611.05431, 2016.