HomeWork 0 - Alohomora

Gokul Hari
Directory ID: hgokul@umd.edu

***Abstract*—The abstract goes here.**

I. Pb-Lite Edge Detection

In this section, a detailed implementation of a simplified Pb Lite edge detection algorithm [1] is presented. This algorithm considers textural and color discontinuities in addition with intensity discontinuities considered by well regarded Sobel and Canny boundary detectors. The algorithm consists of 4 steps: 1) Generating Filter Banks, 2) Computing Texton, Brightness and Color Maps, 3) Computing Texton, Brightness and Color Gradients, 4) Generating pb-Lite Outputs.

A. Generating Filter Banks

Filter banks are necessary to obtain textural information of an image. In this subsection, three different filter banks are generated, namely, 1) Oriented DoG Filters, 2) Leung-Malik Filters, 3) Gabor Filters. Oriented DoG Filter Bank is a collection of Difference of Gaussian (DoG) filters across various scales and orientations. Leung-Malik Filters are a collection of 48 filters. They consist of first and second order derivatives of gaussian with an elongation factor of 3, across 3 scales and 6 orientations, resulting 36 filters. They also consist of 8 Laplacian of Gaussian (LoG) filters and 4 gaussian filters. Finally gabor filters are generated by modulating a gaussian kernel with sinusoidal plane wave. Figures 1illustrate these three filter banks.

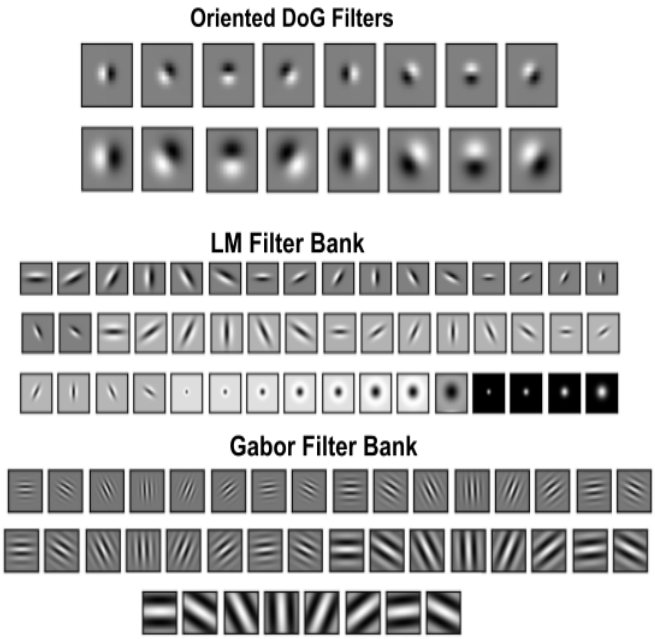

Fig. 1. 3 Filter Banks

B. Computing Texton, Brightness and Color Maps

Texton, brightness and color maps of a given image can be used to represent the textural, brightness and color information of that image.

The filters in the three generated filter banks are used to compute the texton map. Consider that there are a total of N filters in the filter banks. Applying N filters to a given image results in N filter responses. Thus, each input image pixel is now represented with a N dimensional vector. These N-dimensional vectors can be clustered using K-Means clustering and labelled to a discrete texton ID, thus "vector-quantizing" the N dimensional map to a 1D texton map. This is denoted as $\mathcal{T}$. Brightness maps $\mathcal{B}$ can also be generated by performing K means on the grayscaled version of the input image. Color maps $\mathcal{C}$ are generated by performing K means clustering to RGB pixel values. Texton, Brightness and Color Maps are shown in the first row of the output images in subsection I-D

C. Computing Texton, Brightness and Color Gradients

The texton $\mathcal{T}_g$, brightness $\mathcal{B}_g$ and color $\mathcal{C}_g$ gradients of the image can be computed using the respective maps generated in previous subsection. Instead of looping of over each pixel neighborhood, the gradients can be computed more efficiently using half disc masks and χ^2 metric. These masks are illustrated in figure 2.

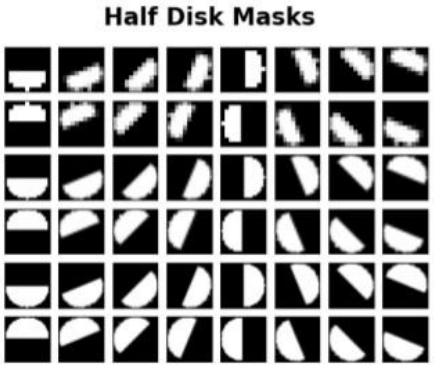

Fig. 2. Half Disk Masks

Notice that these half disk masks are left-right pairs. These left and right mask pairs can be convolved with the maps ($\mathcal{T}$,$\mathcal{B}$,$\mathcal{C}$) to obtain g_i and h_i histograms. The χ^2 value can be

calculated as

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

where K is a predefined number of histogram bins for which χ value was updated iteratively. This χ^2 output of the image is calculated for various orientations o and scales s of the half disk mask pairs and stacked depth wise. The mean of these $o \times s$ outputs of χ is calculated to obtain a 1D gradient map. This procedure is performed on $\mathcal{T}$,$\mathcal{B}$ and $\mathcal{C}$ to obtain $\mathcal{T}_g, \mathcal{B}_g$ and $\mathcal{C}_g$.

D. Generating pb-Lite Outputs

With the gradient maps $\mathcal{T}_g, \mathcal{B}_g$ and $\mathcal{C}_g$ generated, the pb Lite boundary detected output is calculated as,

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

where, $\odot$ is the Hadamard product (Element-wise matrix multiplication) applied on a weighted combination of Sobel and Canny baselines of the same image with w_1 and w_2 as their respective weights. The results of maps and gradients along with the final pb Lite outputs with sobel and canny are showed in the following figures.

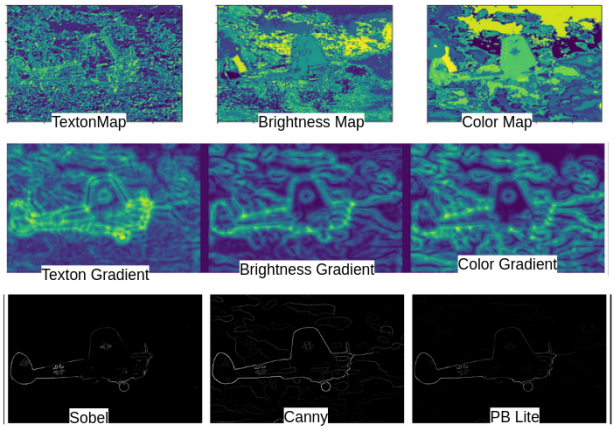

Fig. 3. Result of Image 1

II. Deep Dive on Deep Learning

In this section, several neural network architectures were implemented to solve an image classification problem using CIFAR-10 dataset, which consists of 50,000 training images and 10,000 testing images of size 32×32 with 10 different classes. The training and testing accuracies, with respect to epochs, during the training routines were recorded for analysis. The performance metrics to used to test these architectures is Confusion matrix and Accuracy Score. Categorical Cross entropy Loss was used. Adam optimizer was chosen with a learning rate of 0.001. Batch size was chosen to be 64 while the number of epochs were chosen either 25 or 50, depending on the model & experiment.

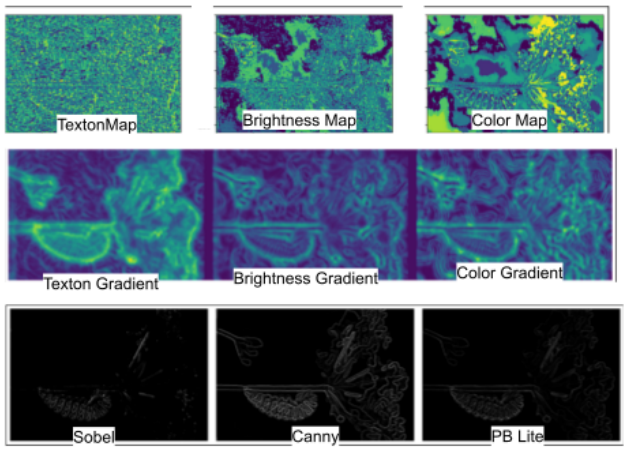

Fig. 4. Result of Image 2

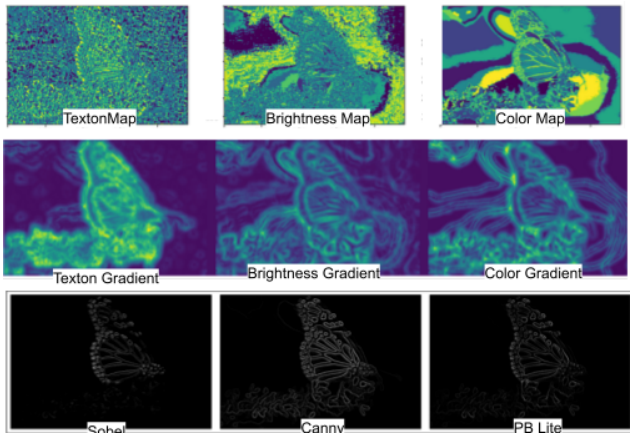

Fig. 5. Result of Image 3

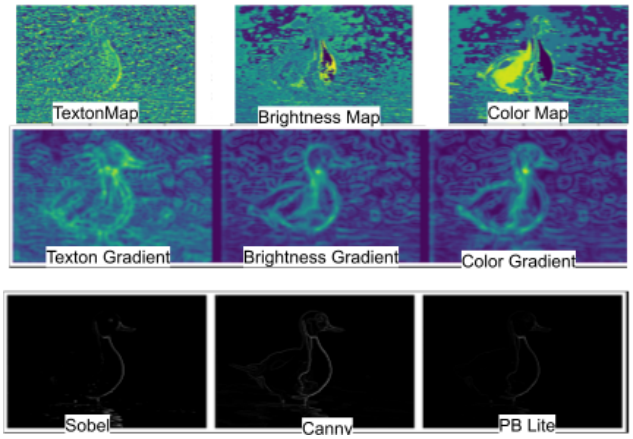

Fig. 6. Result of Image 4

A. Starter Neural Network

To begin with this section, I trained a very simple neural network to get a quick grasp on the ML model training routine. This unoptimised model has just 2 Convolution layers and 2 dense layers as depicted in 13. Unpreprcessed

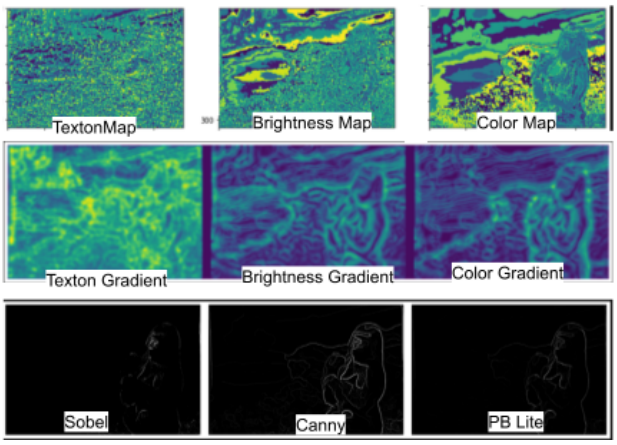

Fig. 7. Result of Image 5

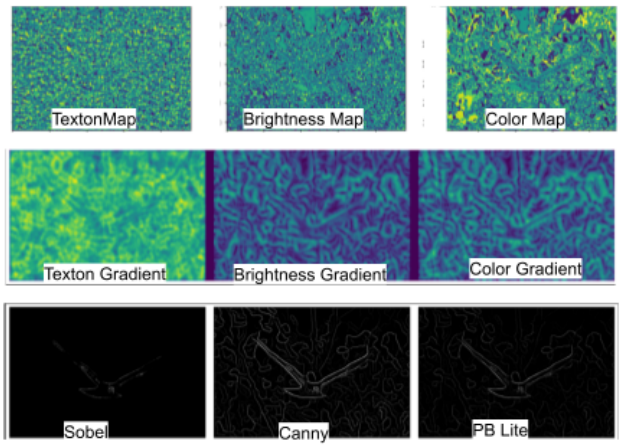

Fig. 8. Result of Image 6

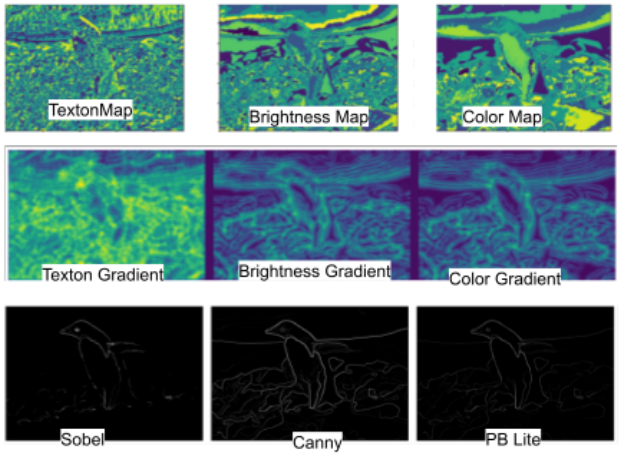

Fig. 9. Result of Image 7

CIFAR 10 Image data was fed to the model.

Results & Observations: While training, the model's training accuracy saturated quickly at just 5 epochs, while the test accuracy plateaued around 50%. This indicates over-fitting,

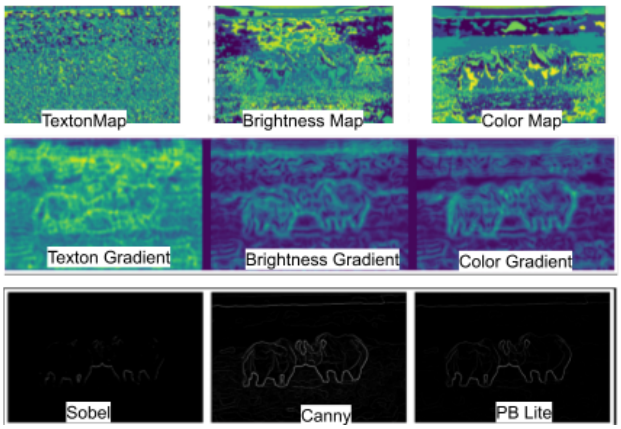

Fig. 10. Result of Image 8

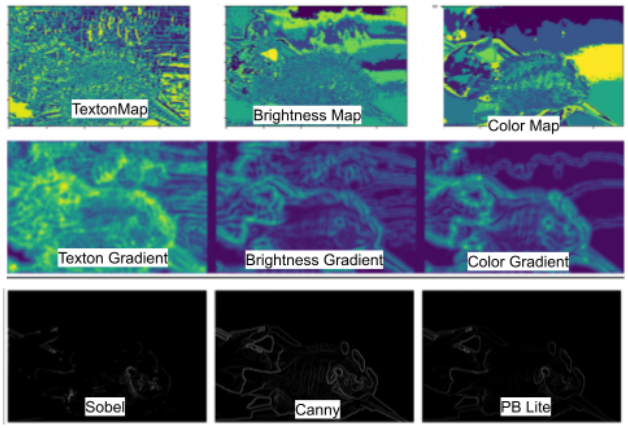

Fig. 11. Result of Image 9

where the model cannot generalise it's performance when tested with new data. This is seen in Figure 14

The Confusion matrix of the model is shown in 15. The overall test accuracy of the model is just 49.01%, for this model with an enormous of 8,409,418 parameters. The model's low accuracy can be addressed by adding more convolution layers followed by batch normalization, to make training faster. The excessive complexity of the model can be controlled by adding more pooling layers, like maxpooling. The problem of model overfitting can be addressed by adding dropout layers. Apart from modifying the model's architecture, data preprocessing methods like normalization and data augmentation will also effectively address the issues of low accuracy and overfitting.

B. Modified Architecutre

Addressing the shortcomings mentioned in previous section, I modified the architecture with more number of convolution layers along with max pooling and dropout layers. This modified architecture is shown in 16.

Results & Observations: In Figure 17, which corresponds to the Modified Architecture, we can see a decent improvement in test accuracy and relatively less over-fitting unlike

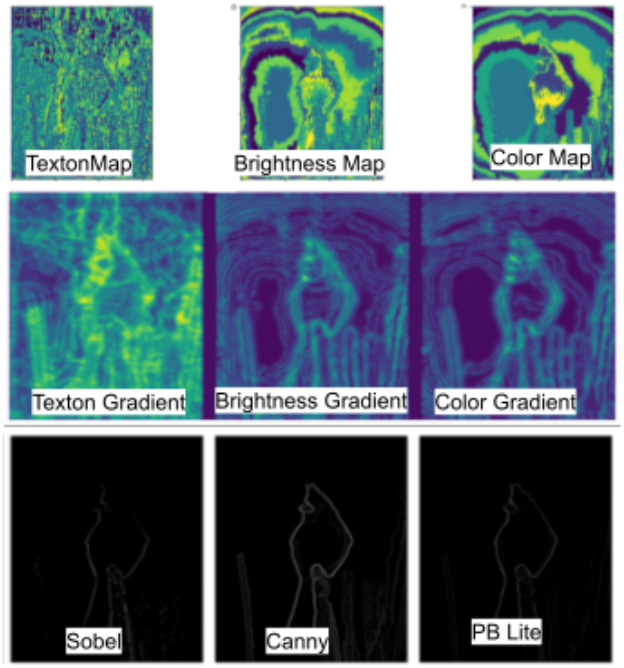

Fig. 12. Result of Image 10

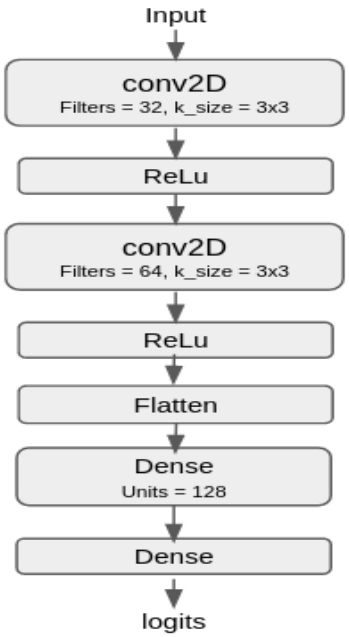

Fig. 13. Simple Neural Network

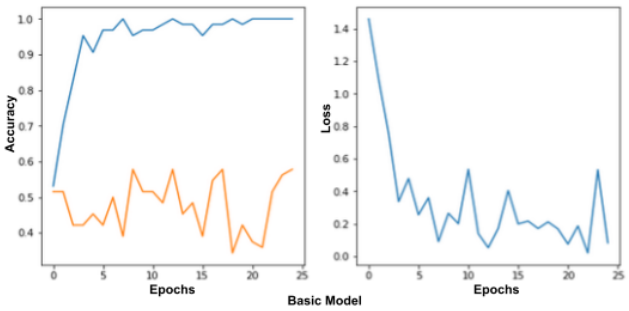

Fig. 14. Left: Train/Test Accuracy vs Epochs; Right: Loss vs Epochs

```
[580  33  63  44  32  11  17  24 136  60]
[ 45 661   5  20   8   6   8  15  62 170]
[118  19 329 142 102  80  80  62  45  23]
[ 46  31  68 360  80 182  89  78  28  38]
[ 47  11 127 119 325  90 110 125  22  24]
[ 29  35  82 195  87 377  52  93  27  23]
[ 26  39  60 134  59  61 543  27  20  31]
[ 35  18  47 102  68  93  12 548  16  61]
[165  63  22  29  14  10  11   7 623  56]
[ 47 215  11  27   9  10  10  29  87 555]
```

Fig. 15. Confusion matrix - Simple Neural Network

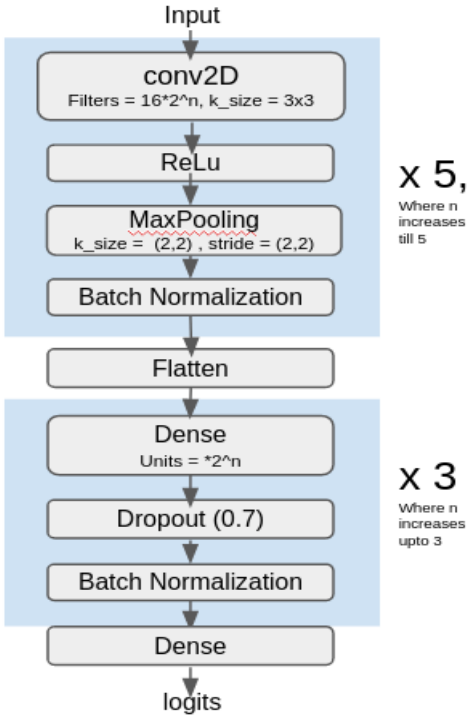

Fig. 16. Modified Architecture

the previous model. The model has a total of 1,807,754 parameters and resulted in a test accuracy of 76.56% after training for 50 epochs. The Confusion matrix of the model is shown in 18 However, we notice that the training accuracy and loss have saturated and hence, we cannot expect the model to perform any better by just training for more epochs. To achieve better performance, we experiment on other established state of the art models.

C. ResNet

ResNet [5], [**?**] introduces the idea of skip connections with identity mappings between the network's blocks of layers, to address the vanishing gradient problem that occurs in deeper models with more layers. I implemented a simple version of Resnet (termed as RESNET-2) with 20 layers,

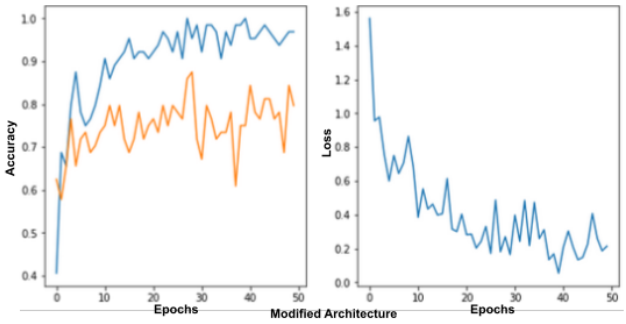

Fig. 17. Left: Train/Test Accuracy vs Epochs; Right: Loss vs Epochs

```
[[783  12  65  13   6   5   8   7  72  29](0)
 [ 10 884  13   5   0   5   6   2  19  56](1)
 [ 42   4 733  36  69  36  45  14  17   4](2)
 [ 22   7 119 461  50 205  67  36  18  15](3)
 [ 23   1  73  30 701  42  41  74  10   5](4)
 [  6   3  55 104  38 708  29  40   6  11](5)
 [  8   4  36  22  37  31 846   3   9   4](6)
 [ 19   2  33  29  36  51  11 797   5  17](7)
 [ 43  15  14   6   0   7   1   4 883  27](8)
 [ 21  66   7   9   0   4   5   6  22 860]](9)

 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
```

Fig. 18. Confusion matrix - Modified Network

as shown in fig 19, and trained it for 50 epochs. I also implemented a modified version where the identity mappings had additional average pooling layers with varying strides (termed as RESNET-1). The ResNet models were referred from the block diagram seen in [2]

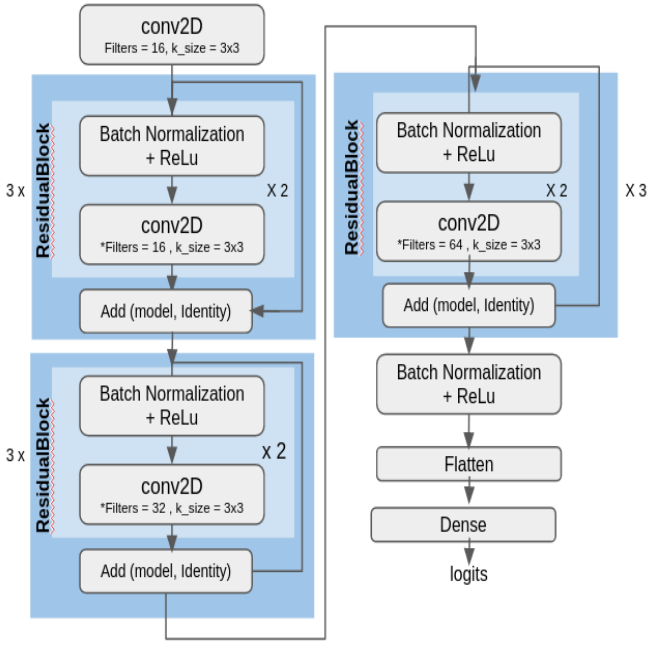

Fig. 19. ResNet-20

1) Results & Observations: Figure 21, corresponds to ResNet-20 with simple identity mappings and Figure 21, corresponds to ResNet-20 with identity mappings containing average pooling and varying strides as seen in [2]. I noticed a slightly higher test accuracy score in the RESNET-1 (74.57%) compared to RESNET-2(74.16%), with both the models containing 795,466 parameters each. The Confusion matrix of the models is shown in 22,23. This implementation could be potentially flawed with my choice of hyperparameters, and is only built for the sake of experimental understanding of residual networks and might not reflect the actual performance of these architecture under optimal parameter choices. This applies for the ResNext and DenseNet model architectures too.

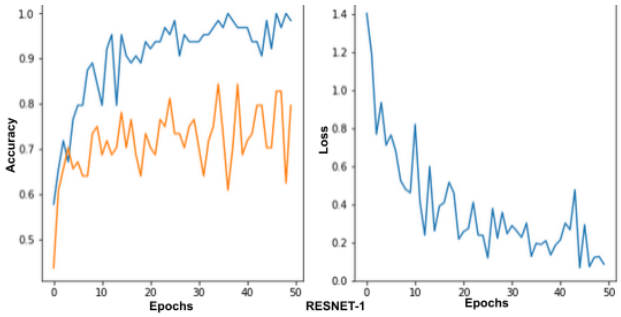

Fig. 20. Left: Train/Test Accuracy vs Epochs; Right: Loss vs Epochs

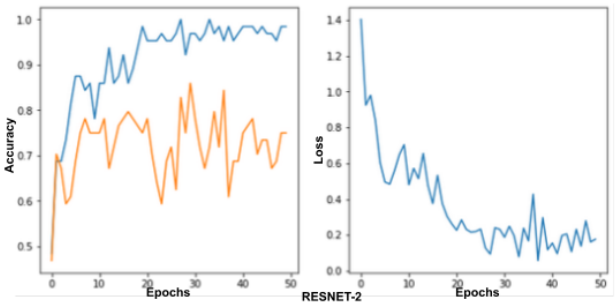

Fig. 21. Left: Train/Test Accuracy vs Epochs; Right: Loss vs Epochs

```
[[796  11  45  32  15   6   8  12  48  27]
 [ 14 881   5  10   1  13   4   2  14  56]
 [ 66   5 618  93  59  65  51  27  15   1]
 [ 14   6  54 598  42 179  54  40   3  10]
 [ 12   1  69  97 676  47  47  43   8   0]
 [  6   2  23 193  29 698  16  28   1   4]
 [  5   7  46  74  36  38 786   3   4   1]
 [ 11   1  30  59  66  64   6 752   4   7]
 [ 62  26  13  16   7  14   6   5 833  18]
 [ 19  78   8  26   3  11   4  14  18 819]]
```

Fig. 22. Confusion matrix - RESNET-2

D. ResNext

ResNext involves splitting the model to C number of bottle neck layers and adding the the results at the end of the block along with the identity mapping connection, as done in ResNet. ResNext introduces the hyperparameter C (cardinality) for model-splitting. The bottle neck layers contain 3 convolution layers, where the middle layer has a kernel size of 3 while the first and last layer are 1×1 convolutions. My model architecture is shown in figure 24

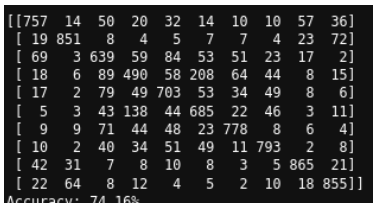

Fig. 23. Confusion matrix - RESNET-1

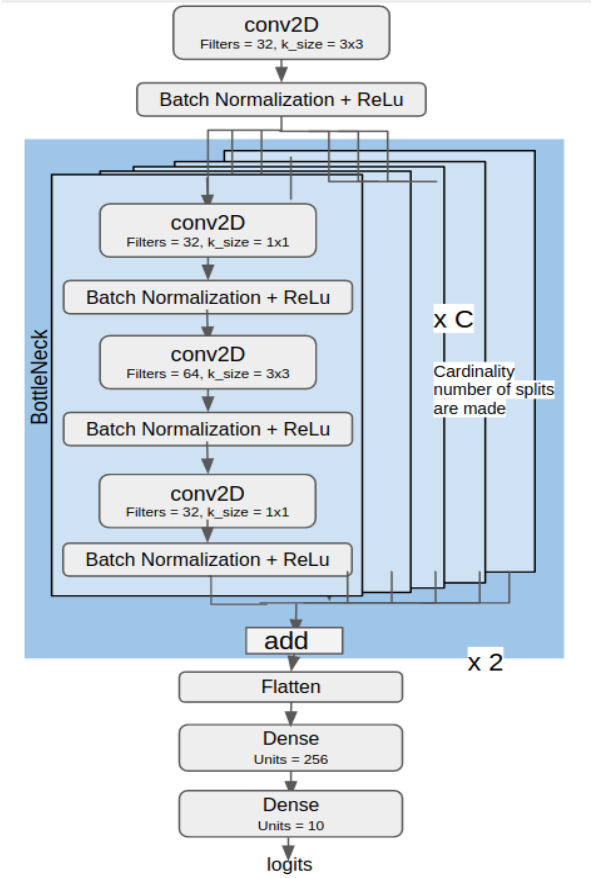

Fig. 24. ResNext

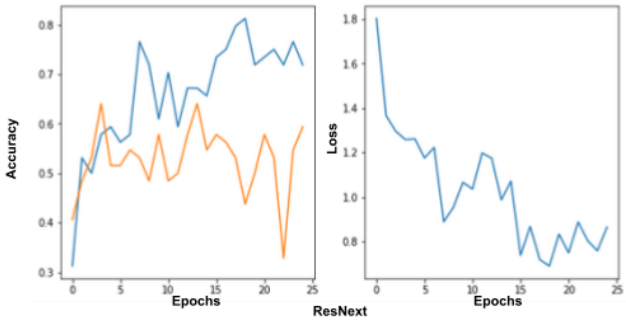

Fig. 25. Left: Train/Test Accuracy vs Epochs; Right: Loss vs Epochs

1) Results & Observations: Figure 25, corresponds to ResNext and it was run only for 25 epochs since my implementation of the model was heavy, un-optimised and unstable as the model's test accuracy and loss showed massive fluctuations. More research can be done from my part, to optimise the model and solve the instability to achieve better performance. The model's test accuracy was 54.7 % with 8,611,274 parameters in it. The Confusion matrix of the models is shown in 26

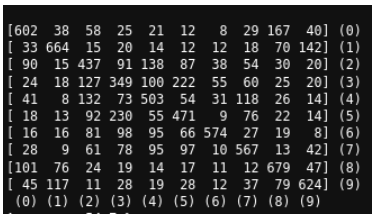

Fig. 26. Confusion matrix - ResNext

E. DenseNet

DenseNet is very similar to ResNet, except that, DenseNet model contains "Dense blocks" where the shortcut connection is concatentated to the end of the block, instead of addition done in ResNet. A DenseNet model usually contains 4 Dense blocks and each Dense Block is connected to one another using Transition Blocks, which contain 1×1 convolutions. My implementation of architecture with only two dense blocks connected by a transition block.is shown in Figure 27, also refered with [4]

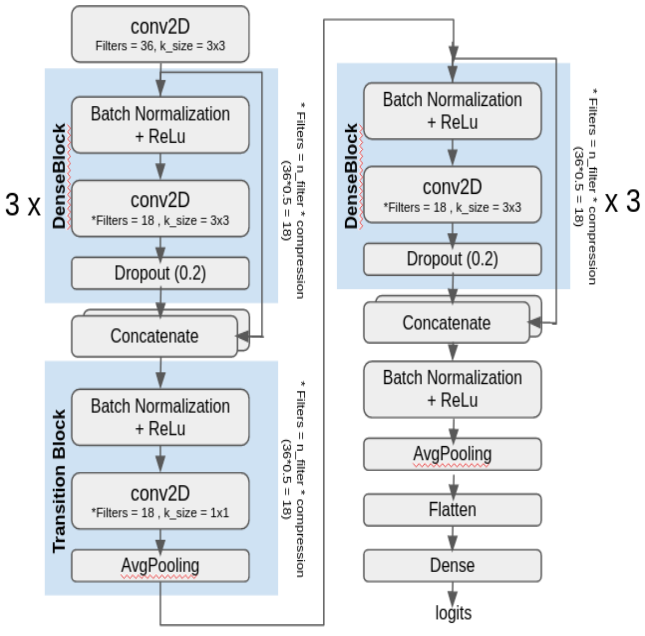

Fig. 27. DenseNet

1) Results & Observations: Figure 28, corresponds to DensNet was run for 50 epochs and the model's train/test accuracy shows a promising trend of increase in accuracy with epochs and no overfitting. The model's test accuracy

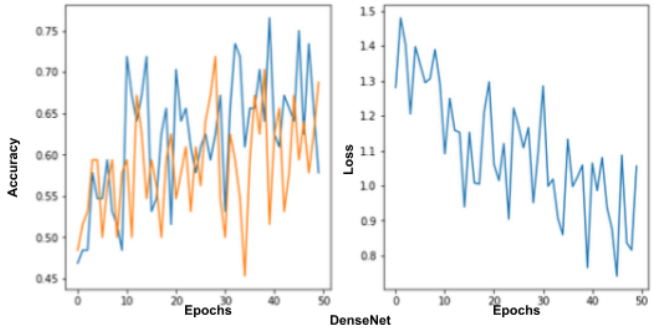

Fig. 28. Left: Train/Test Accuracy vs Epochs; Right: Loss vs Epochs

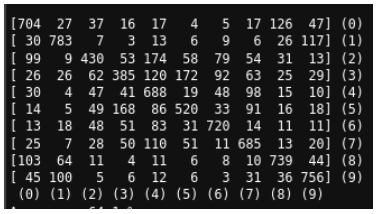

Fig. 29. Confusion matrix - DenseNet

at 50th epoch is 64.1% with 695,368 parameters in it. The Confusion matrix of the models is shown in 29

REFERENCES

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

[2] chao-ji/tf-resnet-cifar10, https://github.com/chao-ji/tf-resnet-cifar10/blob/master/files/ResNet20_CIFAR10.pdf.

[3] He, Kaiming Zhang, Xiangyu Ren, Shaoqing Sun, Jian. (2016). Identity Mappings in Deep Residual Networks. 9908. 630-645. 10.1007/978-3-319-46493-0_38.

[4] aayushs879/Densenet-on-CIFAR-10 ,https://www.kaggle.com/genesis16/densenet-93-accuracy.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun (2015). Deep Residual Learning for Image Recognition. https://arxiv.org/abs/1512.03385.