HW0: Alohomora

Venkateshkrishna
Worcester Polytechnic Institute
Worcester, MA 01609
Email: vparsuram@wpi.edu

***Abstract*—The assignment has 2 phases. Phase 1 is the implementation of simplified version of the pb, an algorithm that finds boundaries by examining brightness, color and texture of images across multiple scales. While Phase 2 involves the implementation of various neural networks to classify the images in the CIFAR-10 dataset.**

I. PHASE 1: SHAKE MY BOUNDARY

The goal of this Phase is to develop a simplifies version of the Pb (Probability of boundary) algorithm. It works by calculating the per pixel probability of a boundary by considering the texture, color discontinuities and intensity discontinuities. There are 4 major steps in the algorithm:

1) Generation of filter banks: Oriented DoG, LM and Gabor filters
2) Generation of Texton, Brightness and Color maps
3) Generation of Texton, Brightness and Color gradient maps
4) Boundary detection by combing the gradient maps with the outputs of Canny and Sobel

A. Generation of Filter Banks

In the first step of the pb lite boundary detection process, we start by running the image through some filter sets. We're going to have three different sets of filters for this. These are the Oriented DoG filters, Leung-Malik Filters and Gabor Filters. Once we've done that, we create a texton map, showing the texture in the image by grouping together the filter responses.

1) Oriented DoG filters: The Differnce of Gaussian filters are created by convolving a simple Sobel filter and a Gaussian kernel and then rotating the result. 2 scales with 16 orientations ranging from 0 to 360 degrees were used to obtain 32 filters. A Gaussian kernels of size 7 and standard deviation 0.7 and 1 were chosen to generate the filter bank. The generated filter bank can be seen in Fig 1.

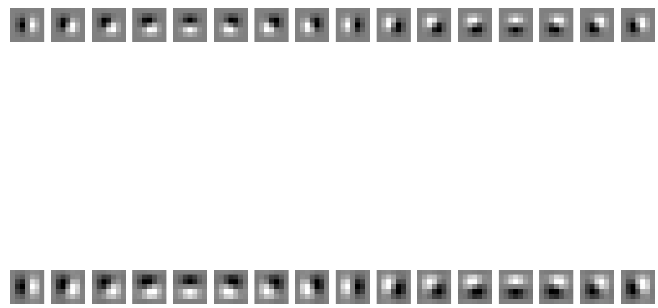

Fig. 1: Oriented Difference of Gaussian filter bank

2) Leung-Malik Filters: The Leung-Malik filters or LM filters are a set of multi scale, multi orientation filter bank with 48 filters. It consists of first and second order derivatives of Gaussians at 6 orientations and 3 scales making a total of 36; 8 Laplacian of Gaussian (LOG) filters; and 4 Gaussians. 2 Versions of this filter are generated LM small and LM large. LM small filters are generated using σ = [1,$\sqrt{2}$,2,2$\sqrt{2}$] and LM large filter are generated using σ = [$\sqrt{2}$,2,2$\sqrt{2}$,4].

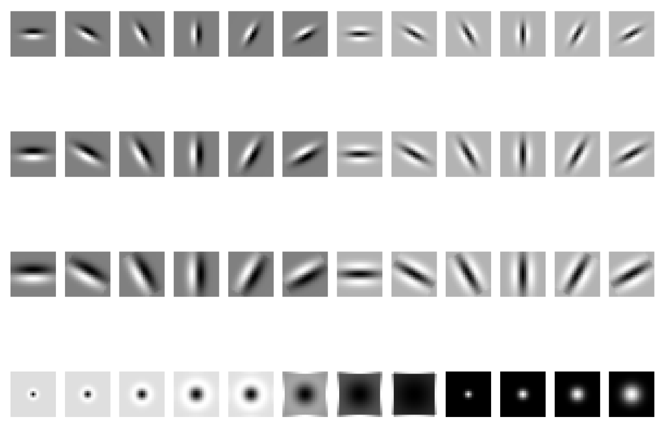

Fig. 2: LM small filter bank

The first and second derivatives of Gaussian occur at the first three scales with $\sigma_x = \sigma$ and $\sigma_y = 3\sigma_x$, whereas the Gaussians occur at all the basic scales, and the Laplacian of Gaussian (LOG) occurs at σ and 3σ. A kernel size of 21 was used to generate these filters. The generated LM small and LM large filter banks can be seen in Fig 2 and Fig 3 respectively.

3) Gabor filters: Gabor filters are designed based on how the human visual system works. It is formed by a Gausian Kernel modulated by a sinusoidal wave. A filter size of 21 was used to generate this filter bank at scales= [4,5,6,8,10]. The generated filter bank can be seen in Fig 4.

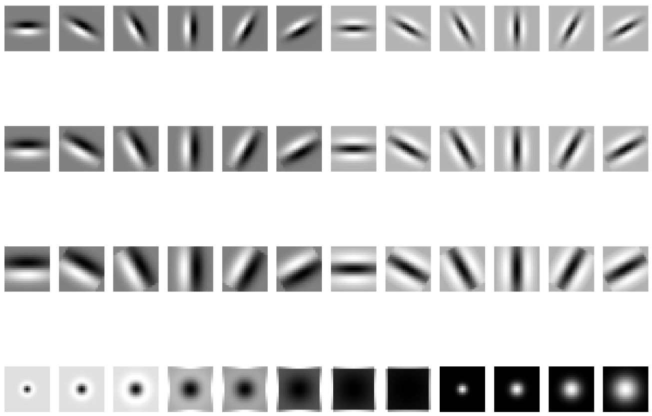

Fig. 3: LM large filter bank

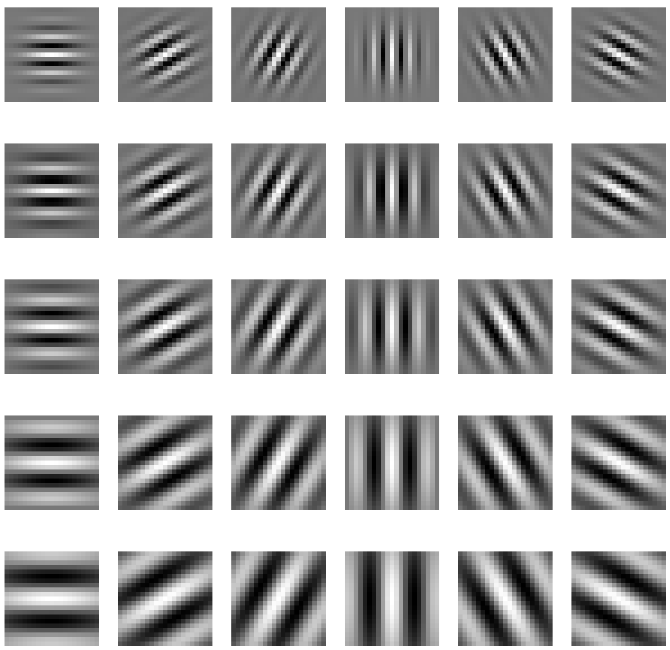

Fig. 4: Gabor filter bank

B. Texton, Brigtness and Color Map

1) Texton Map: To generate a texton maps, all the 168 filters generated are convolved over the image. This reuslts in a vector of filter responses centered around each pixel. The collection of N-dimensional filter responses can be thought of as encoding texture characteristics. To simplify this representation, we replace each N-dimensional vector with a discrete Texton ID. This simplification involves clustering the filter responses at every pixel in the image into K Textons using K-means clustering. Consequently, each pixel is represented by a one-dimensional, discrete cluster ID instead of a vector with high-dimensional, real-valued filter responses. The outcome is presented as a single-channel image with values ranging from 1 to K. A K value of 64 was chosen for K mean clustering.

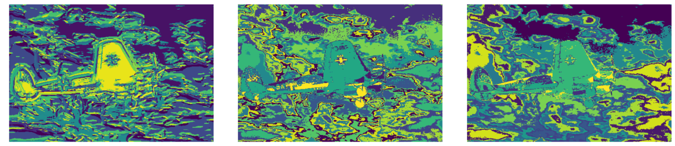

Fig. 5: Texton, Brightness, and Color map for Image 1

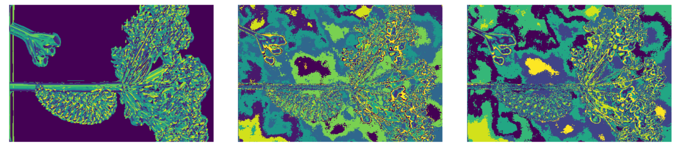

Fig. 6: Texton, Brightness, and Color map for Image 2

2) Brightness Map: The notion of the brightness map involves capturing variations in brightness within the image. Once more, we employ k-means clustering to group the brightness values (equivalent to the grayscale representation of the color image) into 16 clusters.

3) Color Map: The concept of the color map is used to capture the color changes or chrominance content in the image. Here, again we cluster the RGB color values using kmeans clustering into 16 clusters.

The generated Texton, Brightness and Color map for the 10 test images can be seen in Fig 5 through Fig 14.

C. Texton, Brightness and Color Gradient

The Texture, Brightness, and Color gradients (Tg, Bg, Cg) are computed to analyze the changes in the distributions of Texture, Brightness, and Color maps at each pixel. These gradients are determined by convolving half-disk masks of various orientations and scales with the previously generated maps. The use of half-disk masks facilitates the evaluation of gradient maps at different scales and angles, allowing us to capture variations in texture, brightness, and color across different orientations and scales. The half disks masks generated can be seen in Fig 15. This approach aids in

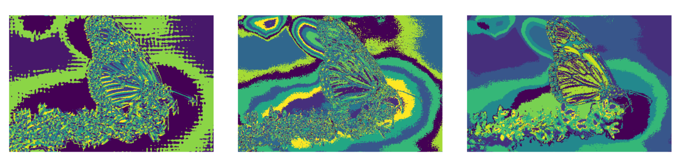

Fig. 7: Texton, Brightness, and Color map for Image 3

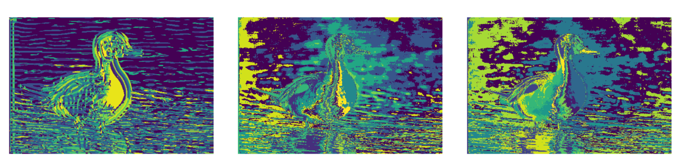

Fig. 8: Texton, Brightness, and Color map for Image 4

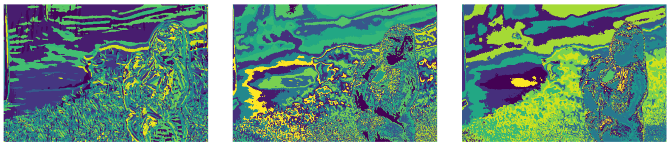

Fig. 9: Texton, Brightness, and Color map for Image 5

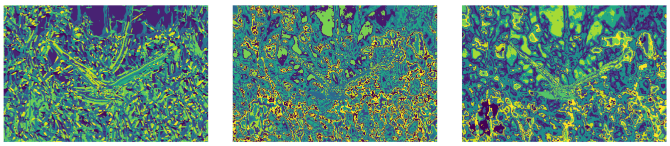

Fig. 10: Texton, Brightness, and Color map for Image 6

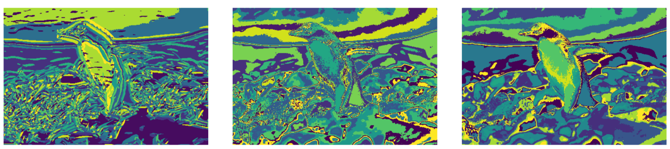

Fig. 11: Texton, Brightness, and Color map for Image 7

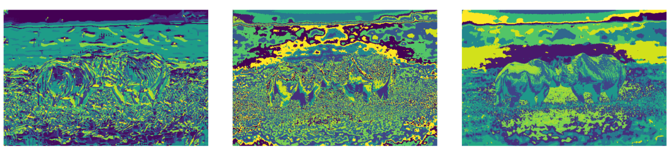

Fig. 12: Texton, Brightness, and Color map for Image 8

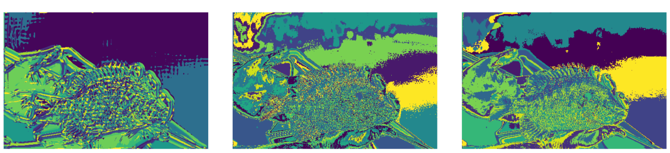

Fig. 13: Texton, Brightness, and Color map for Image 9

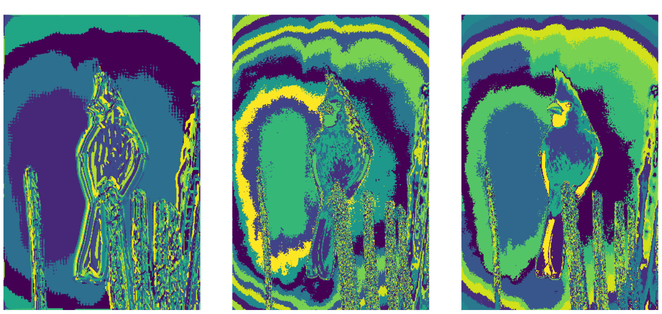

Fig. 14: Texton, Brightness, and Color map for Image 10

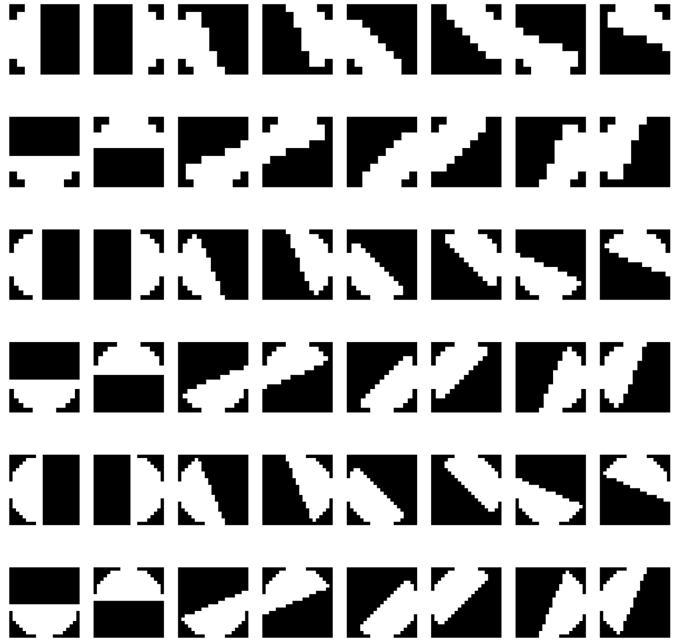

Fig. 15: Half disk masks

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

Fig. 16: Chi-square distance formula

calculating the chi-square distance between the filtered left and right parts around each image pixel. The chi-square distance, commonly employed for comparing histograms, quantifies the similarity or dissimilarity between the filtered left and right portions of each image pixel. The chi-square distance formula can be seen in Fig 16.

The generated Texton, Brightness and Color gradients for the 10 test images can be seen in Fig 17 through Fig 26.

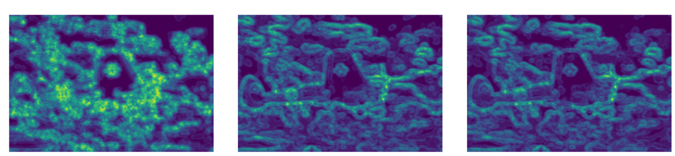

Fig. 17: Texton, Brightness, and Color gradients for Image 1

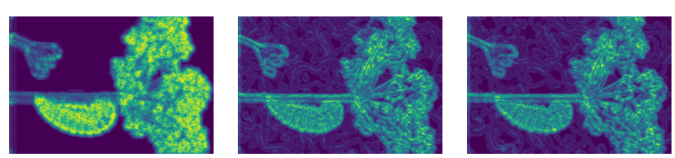

Fig. 18: Texton, Brightness, and Color gradients for Image 2

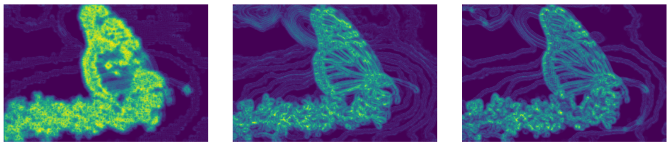

Fig. 19: Texton, Brightness, and Color gradients for Image 3

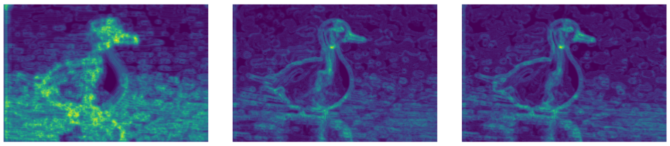

Fig. 20: Texton, Brightness, and Color gradients for Image 4

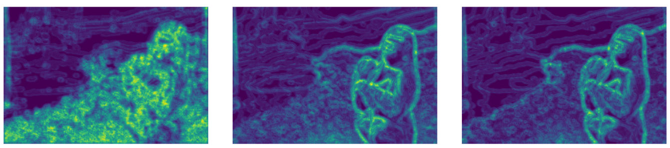

Fig. 21: Texton, Brightness, and Color gradients for Image 5

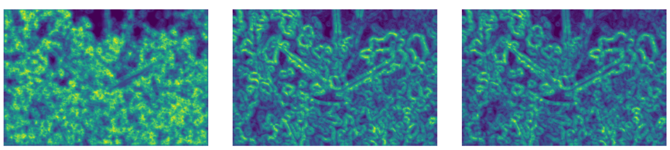

Fig. 22: Texton, Brightness, and Color gradients for Image 6

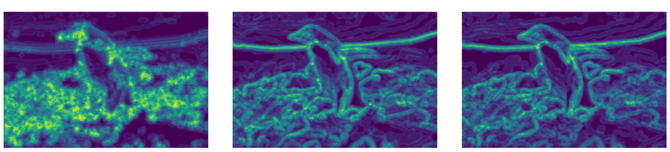

Fig. 23: Texton, Brightness, and Color gradients for Image 7

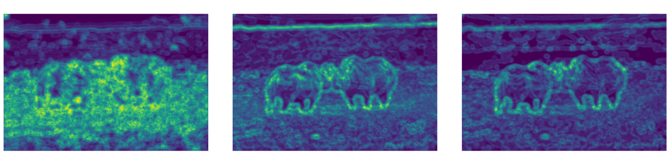

Fig. 24: Texton, Brightness, and Color gradients for Image 8

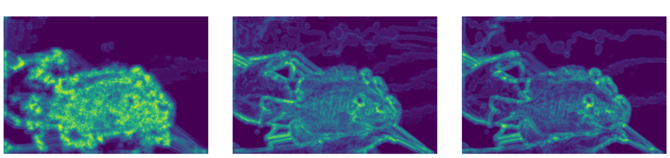

Fig. 25: Texton, Brightness, and Color gradients for Image 9

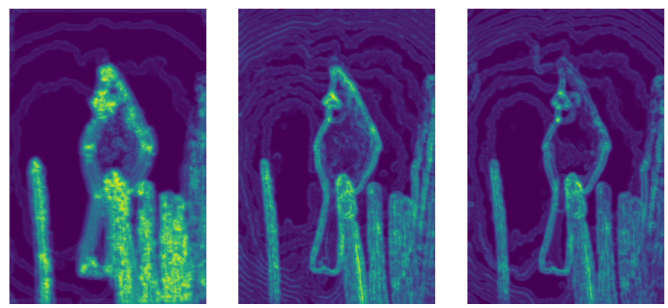

Fig. 26: Texton, Brightness, and Color gradients for Image 10

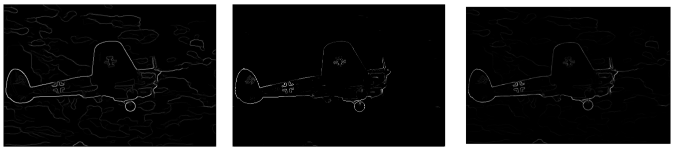

Fig. 27: Canny, Sobel and Pb lite outputs for Image 1

Fig. 28: Canny, Sobel and Pb lite outputs for Image 2

D. Pb lite Output

The final output of the pb lite algorithm is achieved by averaging the gradients of Tg, Bg, and Cg. In a similar manner, a weighted average is done for the Sobel and Canny baselines, and the resulting map is element-wise multiplied with the previous average. This process yields a comprehensive map that incorporates Texton, brightness, and color features, along with the features present in the Canny and Sobel baselines. w1=0.5 and w2 =0.5 were chosen for this operation.This is done using the equation:

$$\mathrm{PbEdges} = \frac{(\mathrm{Tg} + \mathrm{Bg} + \mathrm{Cg})}{3} \odot (w1 \cdot \mathrm{cannyPb} + w2 \cdot \mathrm{sobelPb})$$

A comparison between the Canny, Sobel and Pb lite outputs can be seen in Fig 27 through Fig 36.

Fig. 29: Canny, Sobel, and Pb lite outputs for Image 3

Fig. 30: Canny, Sobel, and Pb lite outputs for Image 4

Fig. 31: Canny, Sobel, and Pb lite outputs for Image 5

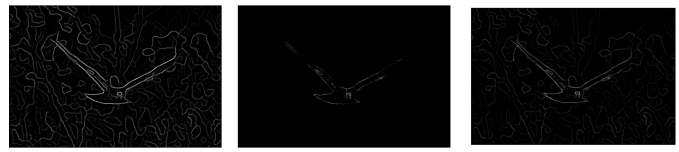

Fig. 32: Canny, Sobel, and Pb lite outputs for Image 6

Fig. 33: Canny, Sobel, and Pb lite outputs for Image 7

Fig. 34: Canny, Sobel, and Pb lite outputs for Image 8

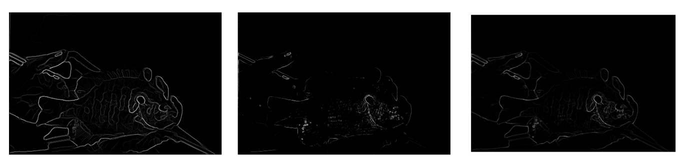

Fig. 35: Canny, Sobel, and Pb lite outputs for Image 9

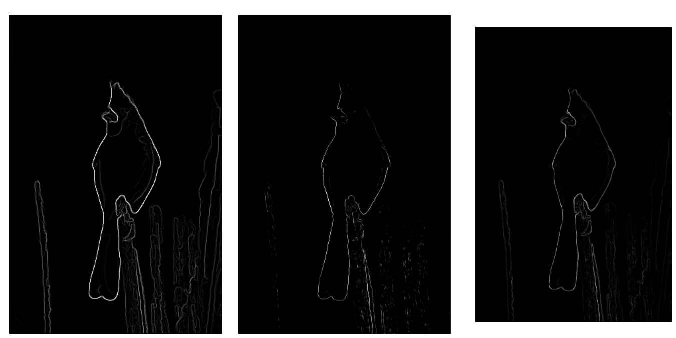

Fig. 36: Canny, Sobel, and Pb lite outputs for Image 10

E. Conclusion

Upon comparing with the baselines, it becomes evident that Pb lite has successfully eliminated numerous incorrect edges identified by Canny, while also incorporating many edges overlooked by Sobel. The results strongly suggest that Pb lite outperforms the standalone Canny and Sobel algorithms.

II. Phase 2: Deep Dive on Deep Learning

The goal of this phase is to implement and train various neural networks to classify the images of the CIFAR-10 dataset. The CIFAR-10 dataset consists of 60,000 (50,000 training and 10,000 testing) 32x32 images belonging to 10 classes. The neural networks implemented are:

1) A Simple CNN
2) An Improved CNN
3) ResNet
4) ResNeXt
5) DenseNet

These networks were further compared on various performance criteria such as accuracy, speed, loss and number of parameters.

A. A simple CNN

A simple convolution neural network with 2 convolution layers and 2 fully connected layers was implemented. The outputs of the convolution layers were activated using RelU activation then passed through max pooling layers. After passing through both the convolution layers the outputs were reshaped and passed through the 2 fully connected layers. The second fully connected layer outputs 10 values. The predicted class is obtained by taking argmax of these 10 values. The architecture of this network can be seen in Fig 37.

The parameters used for training this network are:

- Learning rate=0.001
- Number of epochs= 20
- Batch Size= 32
- Optimizer = Adam
- Loss function = Cross Entropy loss

The accuracy and loss per epoch for the training and test set can be seen in Fig 38. While the confusion matrix for the training and test set can be seen in Fig 39 and 40 respectively.

The Model has an 545098 trainable parameters. It has inference time of 0.00023 seconds per image and has an accuracy of 68.89% on the test set and an accuracy of 81.2% on the train set.

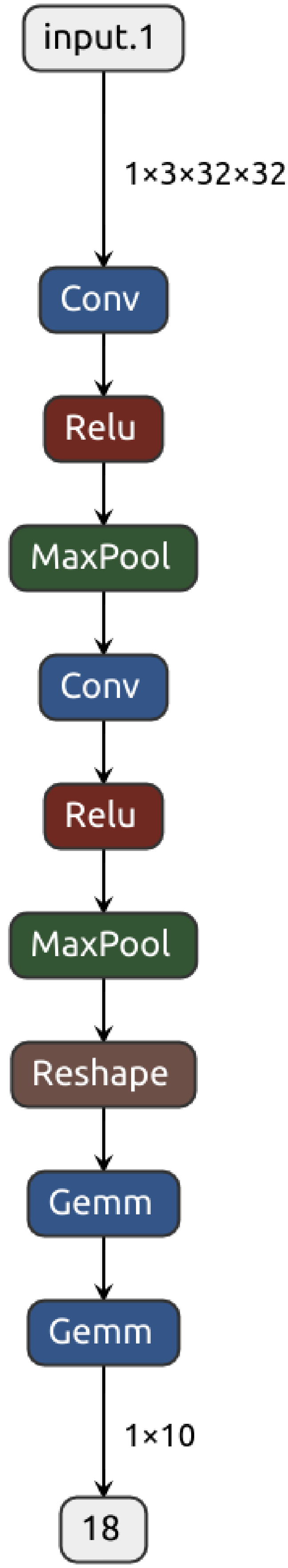

Fig. 37: Architecture of Simple CNN

Fig. 38: Accuracy and Loss per Epoch for the Simple CNN

```
[4339   27  181   68   19   12    8   41  261   44] (0)
[  95 4499   33   33    8   10   18   12  132  160] (1)
[ 216    4 3846  273  165   85  185  137   66   23] (2)
[  96   12  236 3746  101  325  193  190   70   31] (3)
[ 135   14  425  367 3361   82  252  298   50   16] (4)
[  53    7  307 1090  128 2964   82  318   31   20] (5)
[  34    4  212  265   56   31 4342   26   16   14] (6)
[  46    4  126  150  113   60   18 4436   24   23] (7)
[ 141   38   39   30    8    3   13   10 4697   21] (8)
[ 137  205   33   50    9   12   15   50  119 4370] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 81.2 %
```

Fig. 39: Train set Confusion Matrix for the Simple CNN

```
[746  11  66  29   5   5  10  15  83  30] (0)
[ 44 778   9  17   2   4  10   7  52  77] (1)
[ 81   3 618  74  61  41  57  49   9   7] (2)
[ 34   7  94 548  46 131  58  43  27  12] (3)
[ 29   4 112 113 548  17  72  87  16   2] (4)
[ 23   5  82 245  24 514  23  71   9   4] (5)
[ 11   2  73  95  16  11 771   8   9   4] (6)
[ 22   4  52  55  33  33  12 768   7  14] (7)
[ 68  23  11  15   4   7   2   6 847  17] (8)
[ 59  82  11  22   2   5   5  23  40 751] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 68.89 %
```

Fig. 40: Test set Confusion Matrix for the Simple CNN

B. An Improved CNN

The simple CNN was improved by adding another convolution layer bringing the number of convolution layers to 3 and by normalizing the output of the convolution layers by passing it through a batch normalization layer before sending it to the ReLU activation function. The architecture of this network can be seen in Fig 41.

The parameters used for training this network are:

- Learning rate=0.001
- Number of epochs= 20
- Batch Size= 32
- Optimizer = Adam
- Loss function = Cross Entropy loss

The accuracy and loss per epoch for the training and test set can be seen in Fig 42. While the confusion matrix for the training and test set can be seen in Fig 43 and 44 respectively.

The Model has an 357258 trainable parameters. It has inference time of 0.0004 seconds per image and has an accuracy of 76.58% on the test set and an accuracy of 96.0% on the train set.

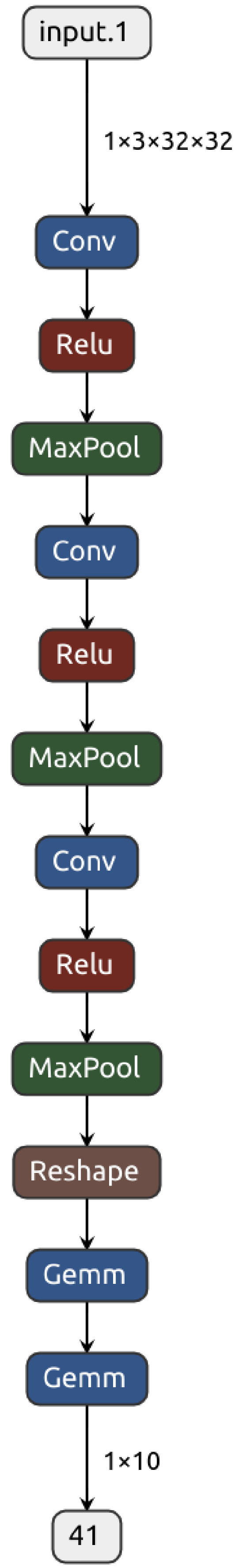

Fig. 41: Architecture of Improved CNN

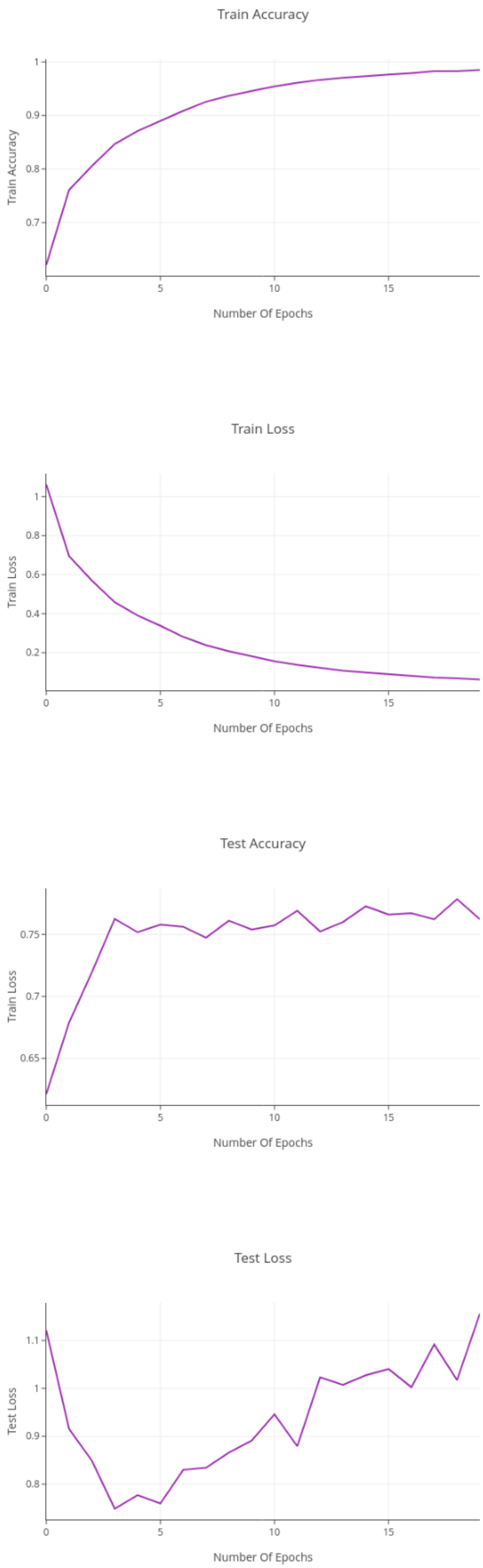

Fig. 42: Accuracy and Loss per Epoch for the Improved CNN

```
[4748   44   28   22   12    7    5   10   66   58] (0)
[   1 4986    2    0    0    0    0    0    4    7] (1)
[  73   14 4721   51   21   43   29   16   20   12] (2)
[   8   12   53 4653   45  137   38   23   12   19] (3)
[  27   11  113   39 4662   53   12   72    6    5] (4)
[   2   10   35   89   11 4783   13   45    2   10] (5)
[   0   24   40   37   18   20 4838    7   12    4] (6)
[   3   10   10   24   13   14    1 4914    1   10] (7)
[  29   27    1    4    2    1    1    0 4915   20] (8)
[   2  193    1    1    0    0    1    8   12 4782] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 96.004 %
```

Fig. 43: Train set Confusion Matrix for the Improved CNN

```
[773  34  30  20   7   6   7  15  66  42] (0)
[  5 926   1   6   2   3   1   3   8  45] (1)
[ 59  13 681  58  44  55  43  32   9   6] (2)
[ 15  16  65 589  58 143  50  28  18  18] (3)
[ 17   3  95  42 689  43  33  64  10   4] (4)
[ 10   8  38 138  24 699  16  47   8  12] (5)
[  5  15  44  51  28  29 812   5   6   5] (6)
[ 13  11  26  33  21  54   3 821   3  15] (7)
[ 50  38   4  12   0   6   4   3 871  12] (8)
[ 11 116   3  15   1   4   3   5  18 824] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 76.85 %
```

Fig. 44: Test set Confusion Matrix for the Improved CNN

C. ResNet

The main innovation of ResNet lies in the use of residual blocks, which contain shortcut connections or skip connections. These connections allow the information from the input of a certain layer to be directly propagated to the output of a deeper layer. The fundamental idea is that instead of trying to learn the mapping directly, the network learns the residual or the difference between the input and the output.

The residual blocks mitigate the vanishing gradient problem, which is a common issue in training deep networks. As networks get deeper, it becomes more challenging for the gradients to flow back through the layers during backpropagation, leading to slow or stalled learning. The skip connections in ResNet help gradients to easily propagate through the network, enabling the training of very deep models.

The ResNet-50 model was implemented which has a total of 50 layers.The architecture of this network can be seen in Fig 45.

The parameters used for training this network are:

- Learning rate=0.001
- Number of epochs= 20
- Batch Size= 32
- Optimizer = Adam
- Loss function = Cross Entropy loss

The accuracy and loss per epoch for the training and test set can be seen in Fig 46. While the confusion matrix for the training and test set can be seen in Fig 47 and 48 respectively.

The Model has an 13970442 trainable parameters. It has inference time of 0.005 seconds per image and has an accuracy of 39.68% on the test set and an accuracy of 42.32% on the train set.

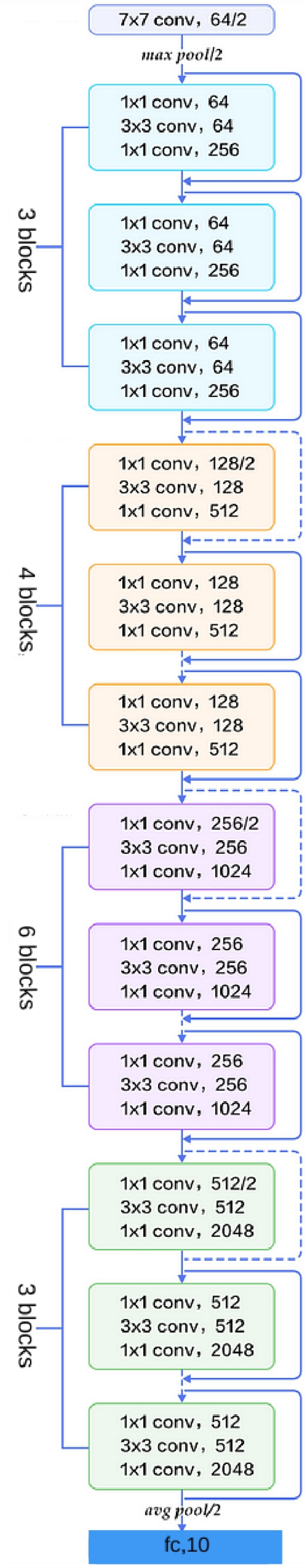

Fig. 45: Architecture of ResNet

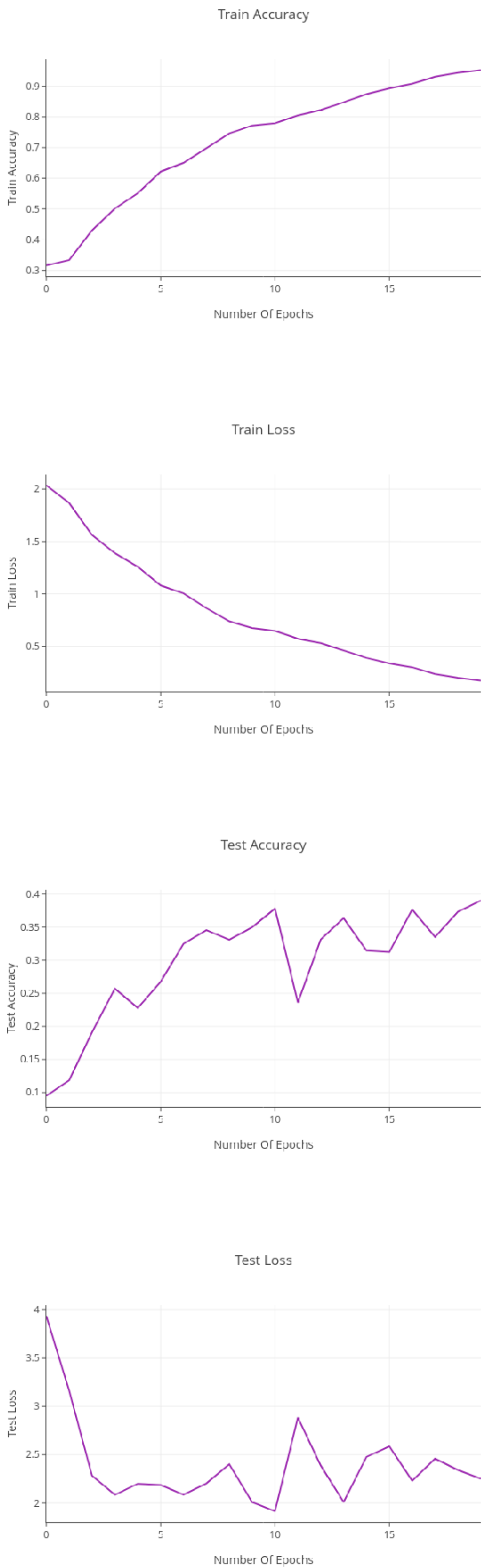

Fig. 46: Accuracy and Loss per Epoch for ResNet

```
[1653  116  392 1306   15    4    4  188    8 1314] (0)
[   4 2794   58  686    7    4    8  164    1 1274] (1)
[  66   51 1734 2563   17   41   37  324    0  167] (2)
[   6   21   48 4469    4   54   10  246    0  142] (3)
[  33   22  434 3406  272   44   58  563    1  167] (4)
[   1   23  166 3157    8  847   14  658    1  125] (5)
[   1   65  185 3192   17   43 1088  214    1  194] (6)
[  12   22   79 1184   14   77    6 3358    0  248] (7)
[ 249  302  179 1381    8    1    2  142  854 1882] (8)
[   2  146   25  554    3   11    3  164    0 4092] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 42.322 %
```

Fig. 47: Train set Confusion Matrix for ResNet

```
[333  25  84 278   4   1   0  22   1 252] (0)
[  2 532  13 163   0   4   1  36   1 248] (1)
[ 20  10 306 515   8  20  16  70   0  35] (2)
[  2  18  16 836   3  19   6  65   0  35] (3)
[  6   3  94 689  48  11  17 109   1  22] (4)
[  3   5  36 660   0 131   3 142   0  20] (5)
[  0   7  37 642   5  18 214  31   1  45] (6)
[  4  11  12 254   5  21   2 619   0  72] (7)
[ 59  66  35 252   2   0   0  18 171 397] (8)
[  3  50   5 119   1   1   1  42   0 778] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 39.68 %
```

Fig. 48: Test set Confusion Matrix for ResNet

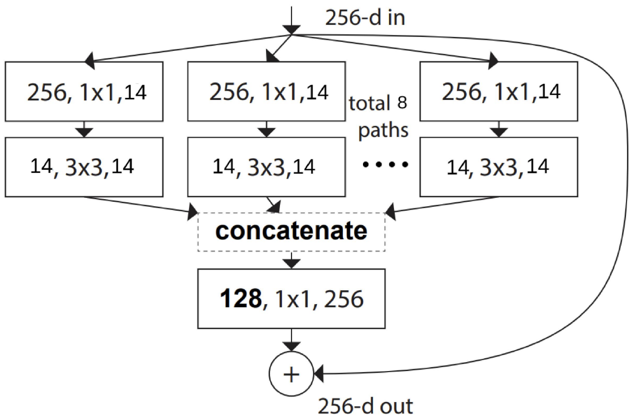

Fig. 49: Architecture of ResNeXt block

D. ResNeXt

The key innovation in ResNeXt is the introduction of a new block called a "cardinality bottleneck," which replaces the traditional bottleneck structure found in ResNet. The cardinality bottleneck involves grouping the channels within a layer into multiple independent paths or "cardinalities." This allows ResNeXt to capture a diverse set of features across different paths, promoting richer representations and improving model generalization.

The cardinality concept provides a flexible way to scale up the network's capacity without significantly increasing the number of parameters, making ResNeXt more computationally efficient compared to traditional approaches.
The ResNet model created earlier was modified to create the ResNeXt model. Each ResNet block was replaced with a ResNeXt block. Initially a cardinality of 32 was chosen but this led to extremely high computation time. To reduce the computation time a cardinality of 8 was chosen along with a bottleneck width of 14.

The architecture of one of the ResNext blocks can be seen in Fig 49.

The parameters used for training this network are:

- Learning rate=0.001
- Number of epochs= 20
- Batch Size= 32
- Optimizer = Adam
- Loss function = Cross Entropy loss

The accuracy and loss per epoch for the training and test set can be seen in Fig 50. While the confusion matrix for the training and test set can be seen in Fig 51 and 52 respectively.

The Model has an 3780202 trainable parameters. It has inference time of 0.0243 seconds per image and has an accuracy of 28.39% on the test set and an accuracy of 29.482% on the train set.

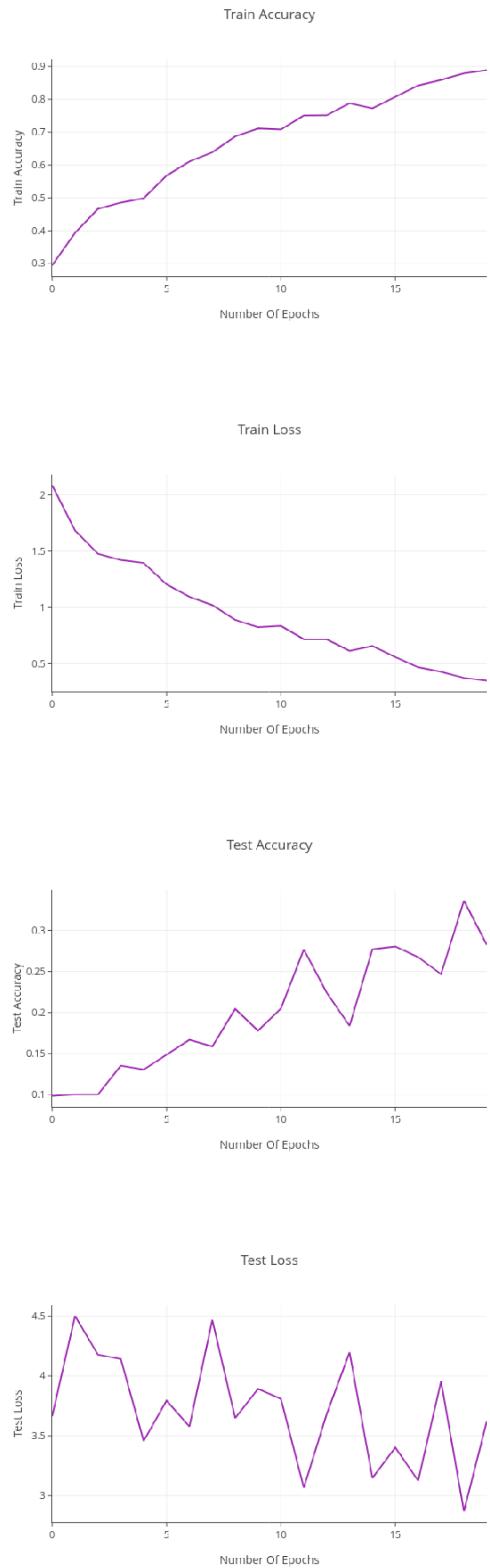

Fig. 50: Accuracy and Loss per Epoch for ResNeXt

```
[2129    2 1792    0  476    0   34  567    0    0] (0)
[ 153 1446 1996   17  532    0  121  724    0   11] (1)
[ 124    0 4289    1  294    0   57  235    0    0] (2)
[ 126    0 3056  159  568    7  275  809    0    0] (3)
[  41    0 2604    3 2019    0   58  275    0    0] (4)
[  66    1 3161   64  420   77  288  923    0    0] (5)
[  58    1 2519    1  744    0 1479  198    0    0] (6)
[   7    0 1153    0  810    0   35 2995    0    0] (7)
[1533   86 2163   17  631    0   72  490    6    2] (8)
[ 173  335 1408    3  695    0  126 2118    0  142] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 29.482 %
```

Fig. 51: Train set Confusion Matrix for ResNeXt

```
[421   2 343   0 110   0   9 115   0   0] (0)
[ 31 284 393   3 116   1  24 145   0   3] (1)
[ 34   1 810   1  78   0  18  58   0   0] (2)
[ 27   0 600  30 126   3  69 145   0   0] (3)
[  6   0 545   1 361   0  20  67   0   0] (4)
[ 21   0 633   8  75  13  60 190   0   0] (5)
[ 12   0 495   1 155   0 284  53   0   0] (6)
[  3   0 229   1 154   0   9 604   0   0] (7)
[328  19 414   3 126   0  16  93   0   1] (8)
[ 41  69 281   0 128   0  28 421   0  32] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 28.39 %
```

Fig. 52: Test set Confusion Matrix for ResNeXt

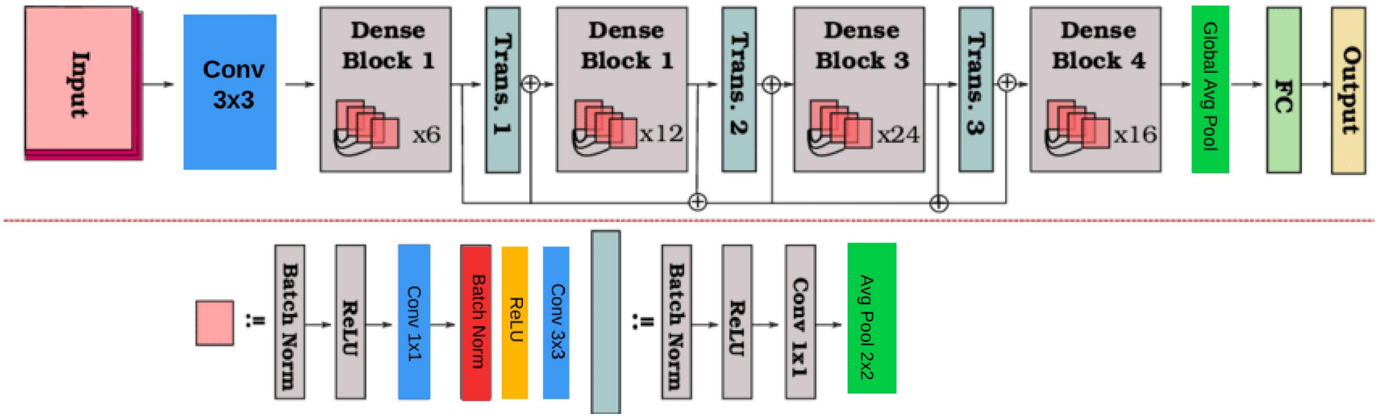

Fig. 53: Architecture of DenseNet

E. DenseNet

In a DenseNet, each layer receives direct input from all preceding layers in the block, and its own feature maps are passed to all subsequent layers. This dense connectivity fosters feature reuse, allowing the network to efficiently leverage information from different scales and abstraction levels. Additionally, DenseNet's dense connectivity addresses the vanishing gradient problem by providing shorter paths for gradients to propagate during training.

DenseNet architectures are characterized by dense blocks, transition layers, and a global average pooling layer at the end.
The DenseNet-121 model was implemented which has a total of 121 layers. A growth rate of 16 was chosen. The architecture of this network can be seen in Fig 53.

The parameters used for training this network are:

- Learning rate=0.001
- Number of epochs= 20
- Batch Size= 32
- Optimizer = Adam
- Loss function = Cross Entropy loss

The accuracy and loss per epoch for the training and test set can be seen in Fig 38. While the confusion matrix for the training and test set can be seen in Fig 54 and 55 respectively.

The Model has an 1739448 trainable parameters. It has inference time of 0.0116 seconds per image and has an accuracy of 85.52% on the test set and an accuracy of 98.388% on the train set.

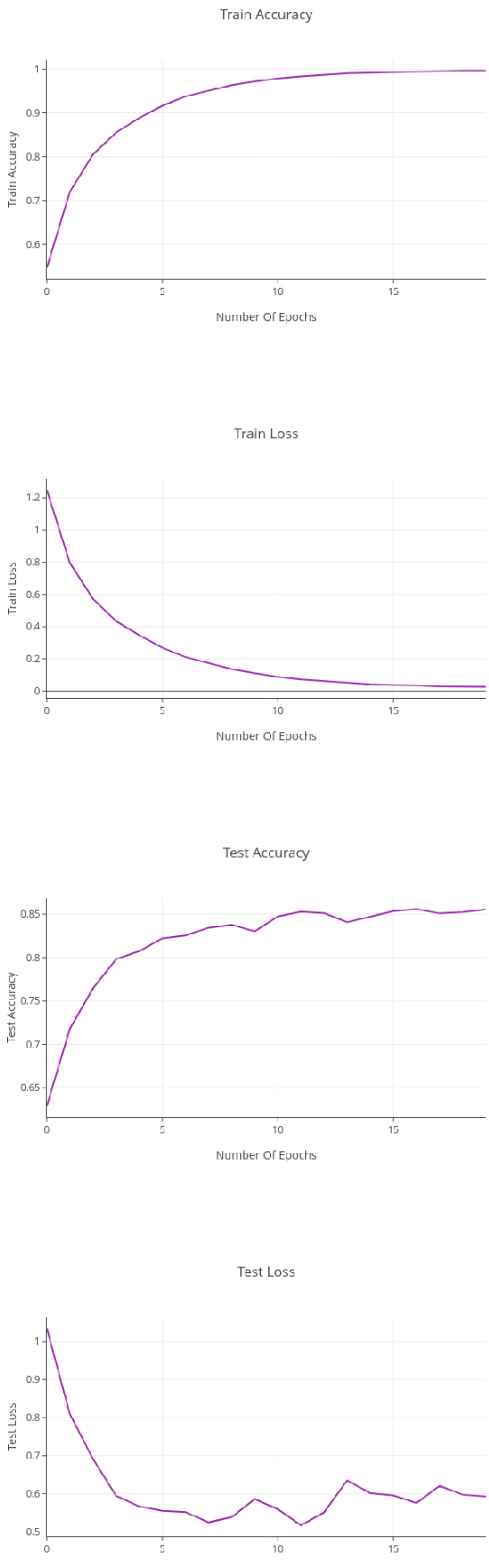

Fig. 54: Accuracy and Loss per Epoch for DenseNet

```
[4958    0   17    2   14    2    2    0    4    1] (0)
[   4 4964    1    1    0    0    2    0   11   17] (1)
[  28    0 4919    8   11   24    8    1    0    1] (2)
[  23    1   40 4730   40  121   17   11   12    5] (3)
[   8    0   25    6 4906   26    5   23    0    1] (4)
[   2    0    9   18   12 4953    1    4    1    0] (5)
[   3    1   29   20    6   28 4910    2    1    0] (6)
[  13    0    8   11   14   26    1 4923    1    3] (7)
[  15    2    2    0    3    1    1    0 4975    1] (8)
[  34    5    1    0    0    0    3    0    1 4956] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 98.388 %
```

Fig. 55: Train set Confusion Matrix for DenseNet

```
[896   6  36   5   9   3   3   3  27  12] (0)
[ 17 936   2   3   1   2   1   0  10  28] (1)
[ 42   1 801  25  44  39  29  10   6   3] (2)
[ 23   4  42 672  39 147  30  21  12  10] (3)
[ 11   1  48  29 831  38  13  25   3   1] (4)
[  6   3  41  76  27 825   4  16   1   1] (5)
[  8   1  36  27  19  18 878   7   4   2] (6)
[ 14   0  15  25  22  31   3 883   1   6] (7)
[ 47   8  10   0   1   5   4   1 916   8] (8)
[ 26  33   3   3   0   1   3   2  15 914] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 85.52 %
```

Fig. 56: Test set Confusion Matrix for DenseNet

F. Discussion and Conclusion

Table 1 provides a comprehensive overview of how the various models perform across multiple criteria.

TABLE I: Comparison of Different Models

Model	Number of Parameters	Train Accuracy (%)	Test Accuracy (%)	Inference Time
Simple CNN	545,098	81.2	68.89	0.00023
Improved CNN	357,258	96	76.58	0.0004
ResNet	13,970,442	42.32	39.68	0.005
ResNext	3,780,202	29.48	28.39	0.0243
DenseNet	1,739,448	98.388	85.52	0.0116

From the data it is evident that DenseNet performs the best, followed by the improved CNN and the Simple CNN. It is also worth noting that ResNet and ResNext do not perform very well, infact even worse than the Simple CNN. This is possibly due to the architecture chosen, ResNet-50 is a fairly large model that requires a lot of training. Perhaps changing the architecture slightly or by tuning the hyper parameters a bit more the performance of these models can be improved. ResNext performs even worse than ResNet possibly due to the low cardinality chosen due to hardware limitations. The impressive performance of the DenseNet shows how well these models can perform and highlights the advantage of using skip connections.

References

[1] Pb lite: link
[2] ResNet: link
[3] ResNeXt: link
[4] DenseNet: link