Homework 0: Alohomora!

Marco Huang
Department of Computer Science
University of Maryland, College Park
hhuang01@umd.edu

I. Phase-1

"Shake My Boundary," ventures into the realm of boundary detection by implementing a simplified Pb detection algorithm that examines brightness, color, and texture gradients. It also compares the results against traditional edge detection methods.

A. Filter Banks

DoG, Leung-Malik, and Gabor Filters are commonly used methods for feature extraction in image processing and computer vision. These filter banks help in detecting and representing different orientations and scales in images, which are essential for tasks such as boundary detection and texture analysis.

1) Oriented DoG filters: Created by convolving a simple Sobel filter and a Gaussian kernel with sigma = [5, 10] kernel size = 50 orientations = 16

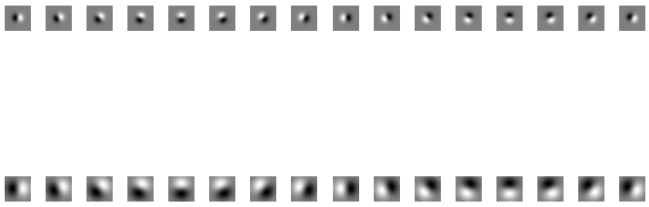


Fig. 1: Oriented Derivative of Gaussian (DoG) filters.

2) Leung-Malik Filters: LM Small (LMS), the filters occur at basic scales $\sigma = \{1, \sqrt{2}, 2, 2\sqrt{2}\}$. The first and second derivative filters occur at the first three scales with an elongation factor of 3, i.e., ($\sigma_x = \sigma$ and $\sigma_y = 3\sigma_x$).

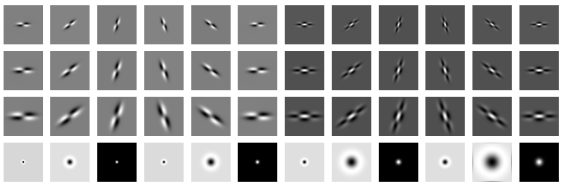


Fig. 2: Laplacian of Gaussian (LOG) filters

3) Gabor Filters: Gabor filter is a gaussian kernel function modulated by a sinusoidal plane wave. sigma3 = [3,5,7,9,12] orientations = 8 wavelength = 5 phase offset = 0 aspect ratio = 0.5

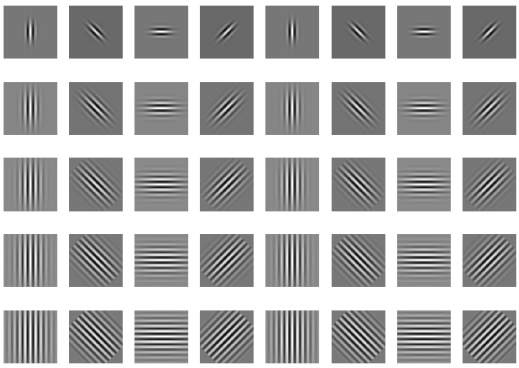


Fig. 3: Gabor Filters

B. Texton, Brightness, Color Map

An image is filtered through an N-filter array, turning each pixel into an N-dimensional vector based on filter responses. KMeans clustering then assigns each pixel a distinct texton ID, resulting in a Texton map. Brightness and Color maps are similarly derived from pixel grayscale values and RGB values, respectively.

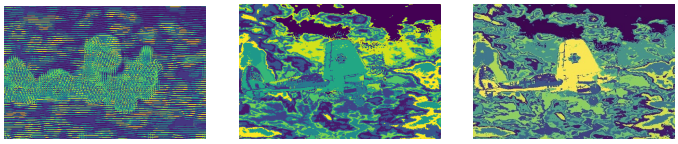


Fig. 4: Texton, brightness and color maps for image 1

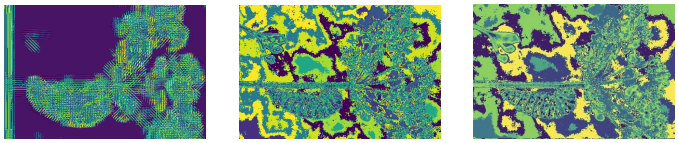


Fig. 5: Texton, brightness and color maps for image 2

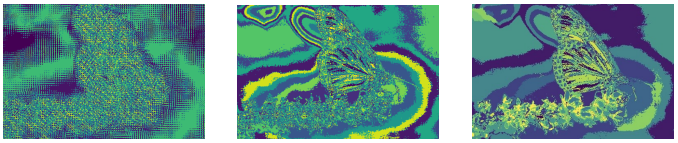


Fig. 6: Texton, brightness and color maps for image 3

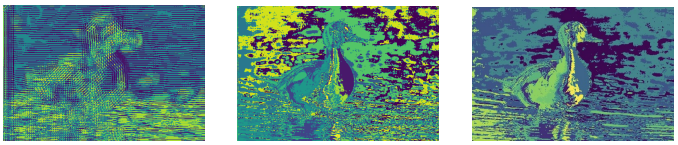


Fig. 7: Texton, brightness and color maps for image 4

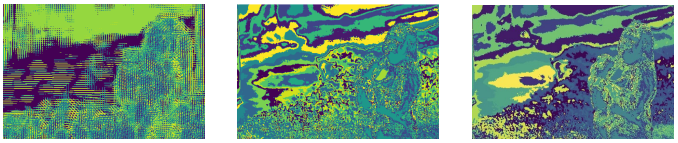


Fig. 8: Texton, brightness and color maps for image 5

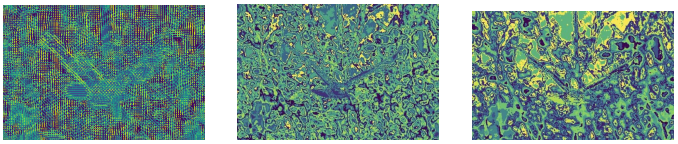


Fig. 9: Texton, brightness and color maps for image 6

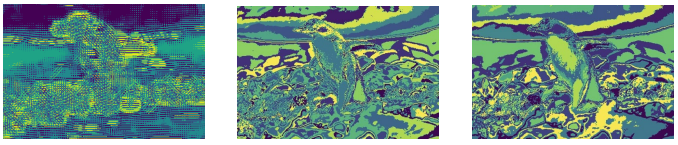


Fig. 10: Texton, brightness and color maps for image 7

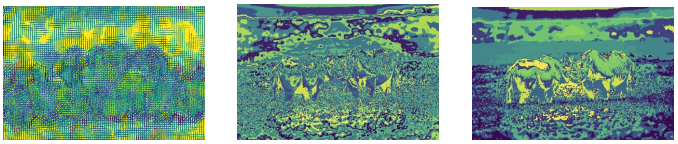


Fig. 11: Texton, brightness and color maps for image 8

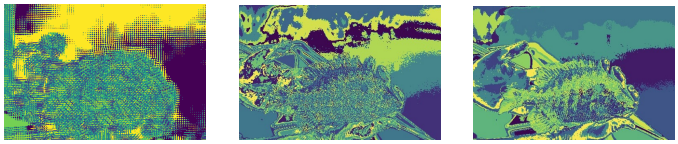


Fig. 12: Texton, brightness and color maps for image 9

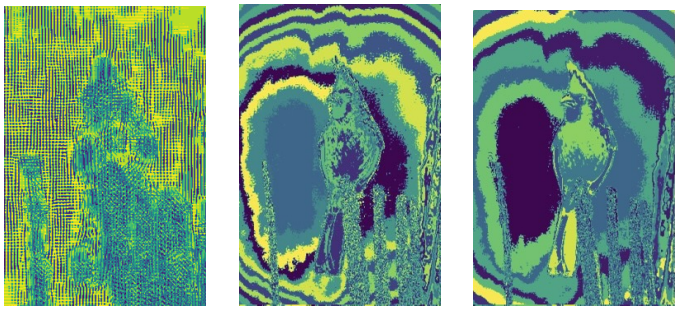


Fig. 13: Texton, brightness and color maps for image 10

C. Texture, brightness, color gradient map

To compute the texture, brightness, and color gradient map. We need to compute differences of values across different shapes and sizes. This can be achieved very efficiently by the use of Half-disc masks.

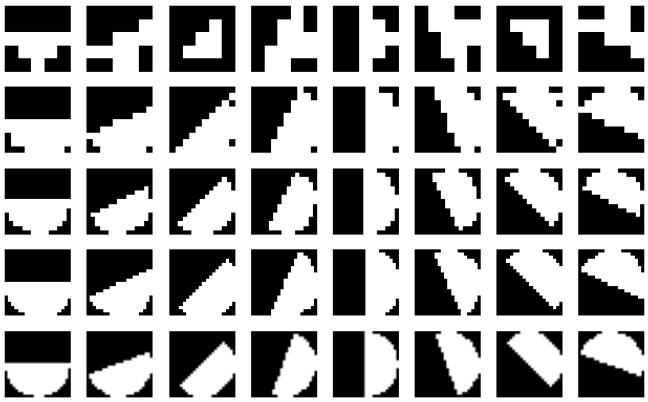


Fig. 14: Half-disc masks

Using these half-disc masks along with the Chi-square distance using a filtering operation. We compute by comparing the distributions in left/right half-disc pairs

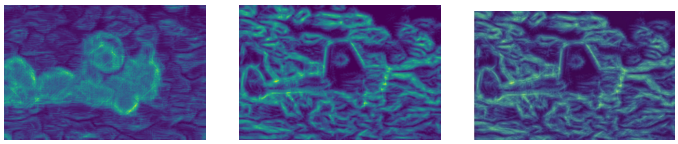


Fig. 15: Gradient maps for image 1

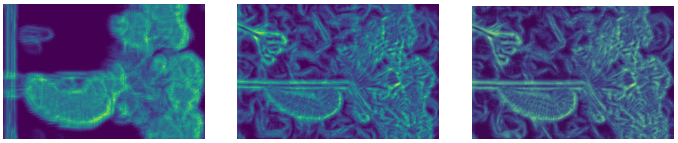


Fig. 16: Gradient maps for image 2

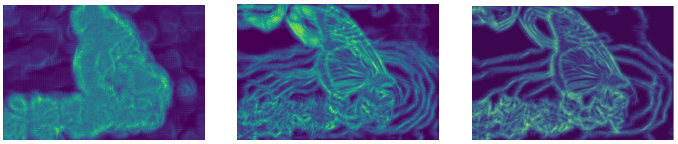


Fig. 17: Gradient maps for image 3

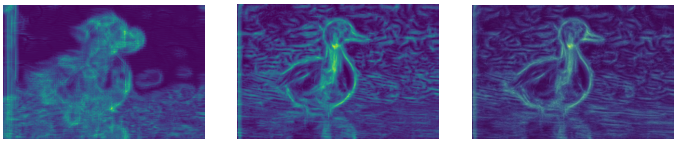


Fig. 18: Gradient maps for image 4

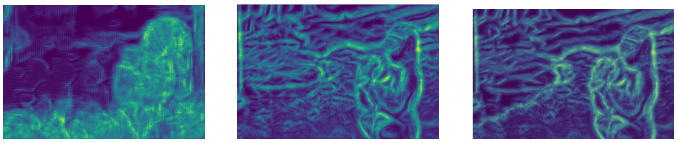


Fig. 19: Gradient maps for image 5

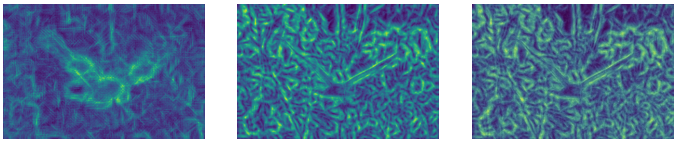


Fig. 20: Gradient maps for image 6

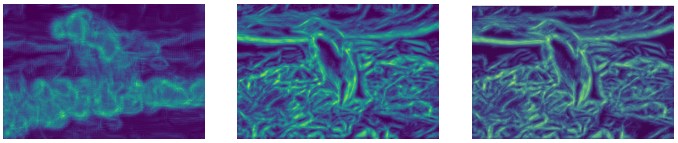


Fig. 21: Gradient maps for image 7

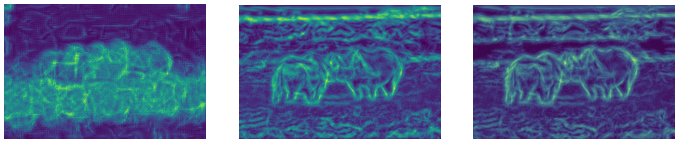


Fig. 22: Gradient maps for image 8

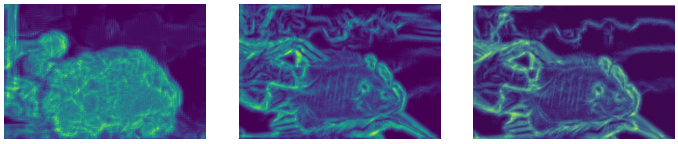


Fig. 23: Gradient maps for image 9

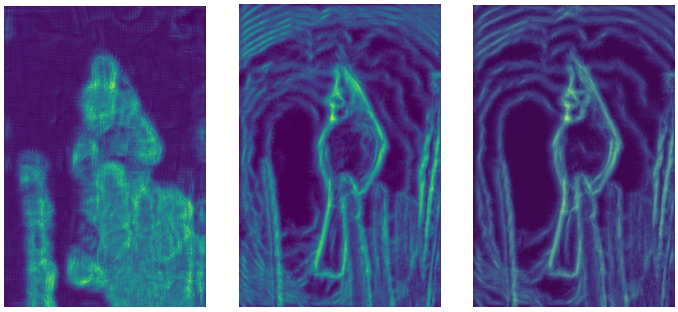


Fig. 24: Gradient maps for image 10

D. Pb-lite Output

The final step is to combine information from the features with a baseline method (based on Sobel or Canny edge detection or an average of both) using a simple equation

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

Here I choose w1 = 0.5 and w2 = 0.5. Compare Canny baseline, Sobel baseline and Pb-lite output.

Fig. 25: Canny, Sobel and Pblite for imgage 1

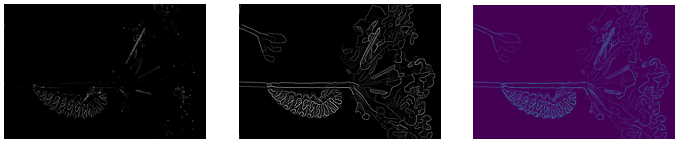


Fig. 26: Canny, Sobel and Pblite for imgage 2

Fig. 27: Canny, Sobel and Pblite for imgage 3

Fig. 28: Canny, Sobel and Pblite for imgage 4

Fig. 29: Canny, Sobel and Pblite for imgage 5

Fig. 30: Canny, Sobel and Pblite for imgage 6

Fig. 31: Canny, Sobel and Pblite for imgage 7

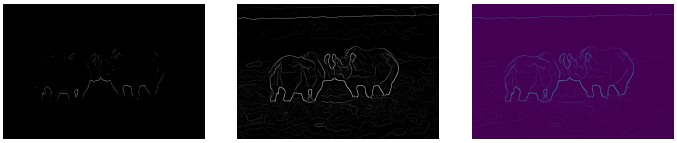


Fig. 32: Canny, Sobel and Pblite for imgage 8

Fig. 33: Canny, Sobel and Pblite for imgage 9

Fig. 34: Canny, Sobel and Pblite for imgage 10

E. Analysis

PB-Lite implementation offers a nuanced balance between sensitivity and specificity in edge detection across images with complex textures. Unlike the Canny baseline, which tends to overemphasize edges leading to numerous false positives, and the Sobel baseline, which might underplay important details by being overly conservative, PB-Lite appears to adeptly moderate edge detection. This moderation ensures that edges in highly textured areas are suppressed effectively, avoiding the pitfall of false positives without obscuring critical image features.

II. PHASE-2

"Deep Dive on Deep Learning" transitions into the application of deep learning architectures to the CIFAR-10 dataset, aiming to enhance classification accuracy. It involves training baseline neural networks, augmenting data, and experimenting with advanced architectures like ResNet, ResNeXt, and DenseNet. The report analyzes the performance and efficiency of these models, providing a comprehensive comparison of their capabilities.

A. Some detail

60000 CIFAR-10 data, 50000 training set and 10000 testing set. Using ADAM optimizer and Cross-Entropy loss function for all 5 training.

B. First neural network

The first task has no normalisation and standardization. With weight decay = 0.0001 and learning rate = 0.001. Size of the Mini Batch to use is 32. The number of parameters in this model are 12.

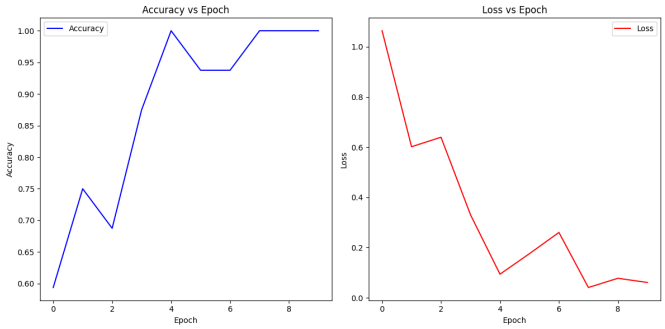


Fig. 35: Train Accuracy and Loss over Epochs for CNN basic

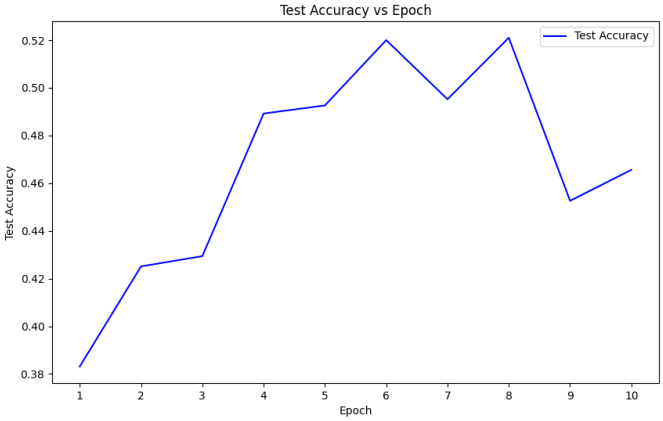


Fig. 36: Test Accuracy over Epochs for CNN basic

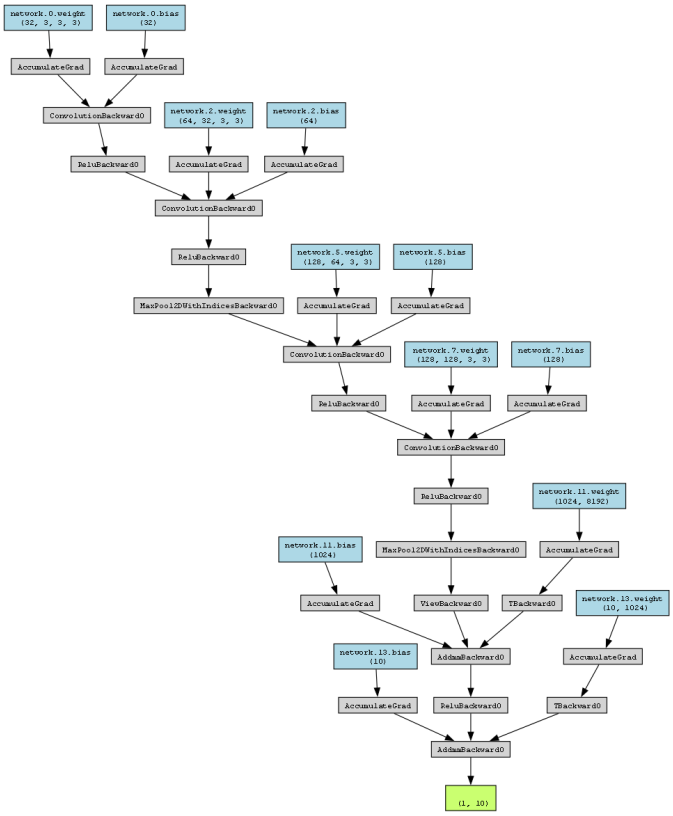


Fig. 37: CNN architecture

```
[517 189  68   7   4  10   5  11  50 139] (0)
[  2 923   2   0   0   1   0   0   3  69] (1)
[ 78 168 413  23   9  69  36  45   8 151] (2)
[ 47 218  99 175   5 149  24  32   9 242] (3)
[ 44 299 128  47  53  53  30  87  11 248] (4)
[ 30 135  80  62   2 470  19  41   9 152] (5)
[ 26 271  53  24   2  33 330  12  10 239] (6)
[ 19 167  43   9   4  39   6 498   3 212] (7)
[ 61 233  12   2   1   3   3   3 480 202] (8)
[  5 180   5   0   0   4   0   4   5 797] (9)
  (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 46.56 %
```

Fig. 38: Confusion Matrix on Testing data for CNN basic

```
[3114  945  159   21   11   24    9   27  129  561] (0)
[   4 4818    5    0    0    2    0    0    1  170] (1)
[ 248  704 2782   65    5  201   97  147   50  701] (2)
[ 167  920  407 1412   10  630   68  128   30 1228] (3)
[ 247 1489  510  135  564  214  151  373   39 1278] (4)
[  87  627  325  153    4 2837   59  183   20  705] (5)
[  86 1357  264   86    1  148 1839   37   23 1159] (6)
[  73  711  208   37    5  108   16 2986    9  847] (7)
[ 209 1116   26    7    1    9    2    7 2748  875] (8)
[  12  569    4    2    0    4    0    3    7 4399] (9)
  (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 54.998 %
```

Fig. 39: Confusion Matrix on Training data for CNN basic

C. *Improving Accuracy of neural network*

To improve my first neural network, I add normalisation with ((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)). Decrease learning rate to 0.005. Increase size of the Mini Batch to 64. Using tf.RandomCrop and tf.RandomHorizontalFlip. Number of parameters in this model are 12.

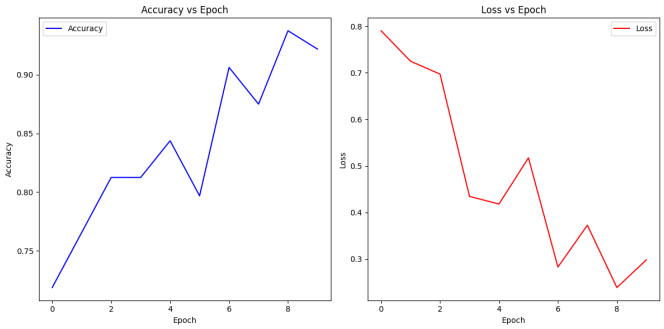


Fig. 40: Train Accuracy and Loss over Epochs for improved CNN

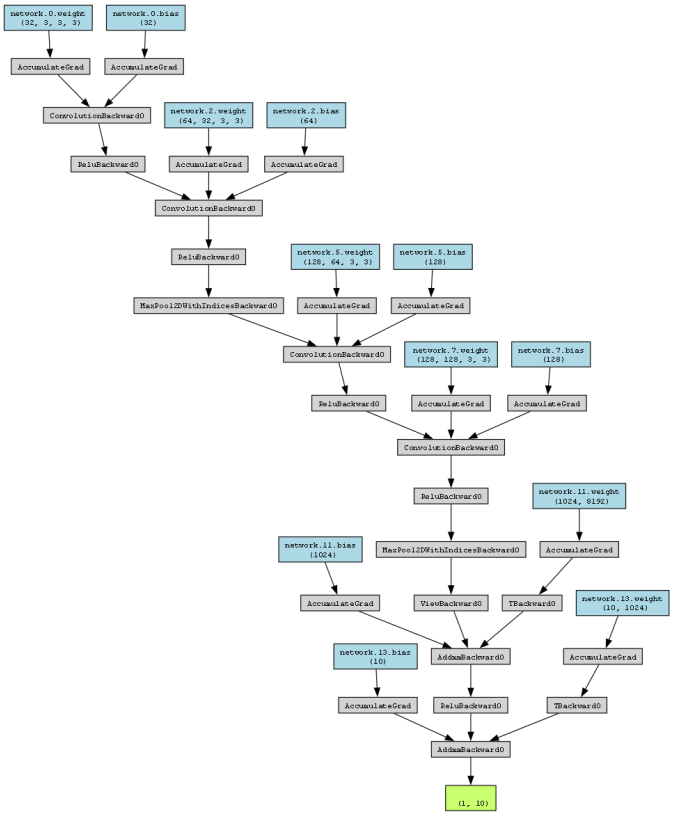
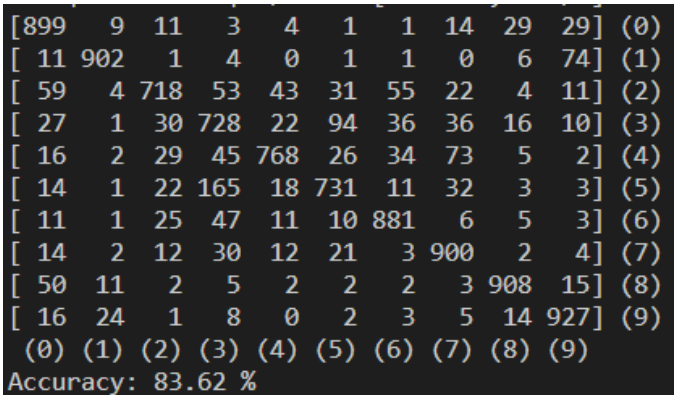


Fig. 42: Improved CNN architecture



```
[899   9  11   3   4   1   1  14  29  29] (0)
[ 11 902   1   4   0   1   1   0   6  74] (1)
[ 59   4 718  53  43  31  55  22   4  11] (2)
[ 27   1  30 728  22  94  36  36  16  10] (3)
[ 16   2  29  45 768  26  34  73   5   2] (4)
[ 14   1  22 165  18 731  11  32   3   3] (5)
[ 11   1  25  47  11  10 881   6   5   3] (6)
[ 14   2  12  30  12  21   3 900   2   4] (7)
[ 50  11   2   5   2   2   2   3 908  15] (8)
[ 16  24   1   8   0   2   3   5  14 927] (9)
  (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 83.62 %
```

Fig. 43: Confusion Matrix on Testing data for improved CNN

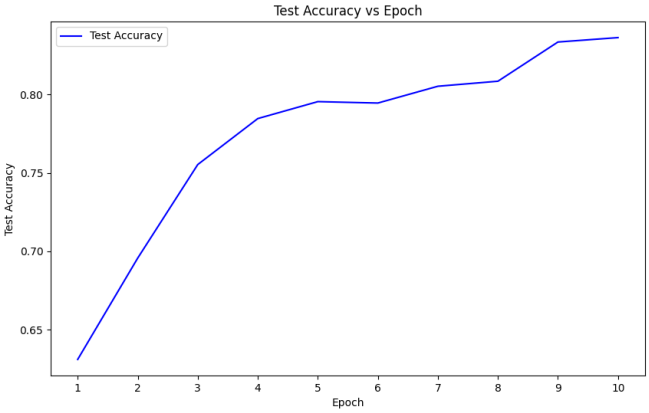
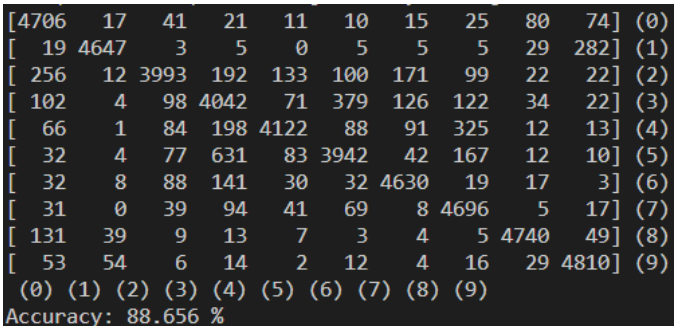


Fig. 41: Test Accuracy over Epochs for improved CNN



```
[4706   17   41   21   11   10   15   25   80   74] (0)
[  19 4647    3    5    0    5    5    5   29  282] (1)
[ 256   12 3993  192  133  100  171   99   22   22] (2)
[ 102    4   98 4042   71  379  126  122   34   22] (3)
[  66    1   84  198 4122   88   91  325   12   13] (4)
[  32    4   77  631   83 3942   42  167   12   10] (5)
[  32    8   88  141   30   32 4630   19   17    3] (6)
[  31    0   39   94   41   69    8 4696    5   17] (7)
[ 131   39    9   13    7    3    4    5 4740   49] (8)
[  53   54    6   14    2   12    4   16   29 4810] (9)
  (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 88.656 %
```

Fig. 44: Confusion Matrix on Training data for improved CNN

D. ResNet, ResNeXt, DenseNet

Normalisation with ((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)) Learning rate is 0.005. Size of the Batch is 64

1) ResNet: The number of parameters in this model are 122

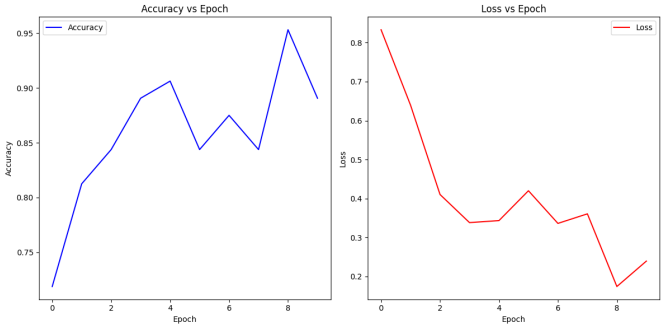


Fig. 45: Train Accuracy and Loss over Epochs for ResNet

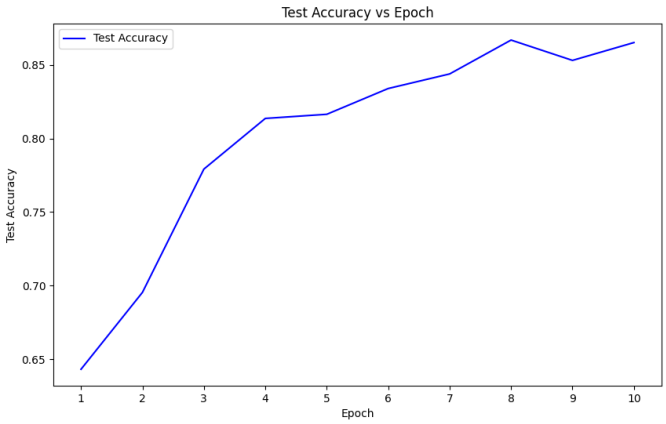


Fig. 46: Test Accuracy over Epochs for ResNet

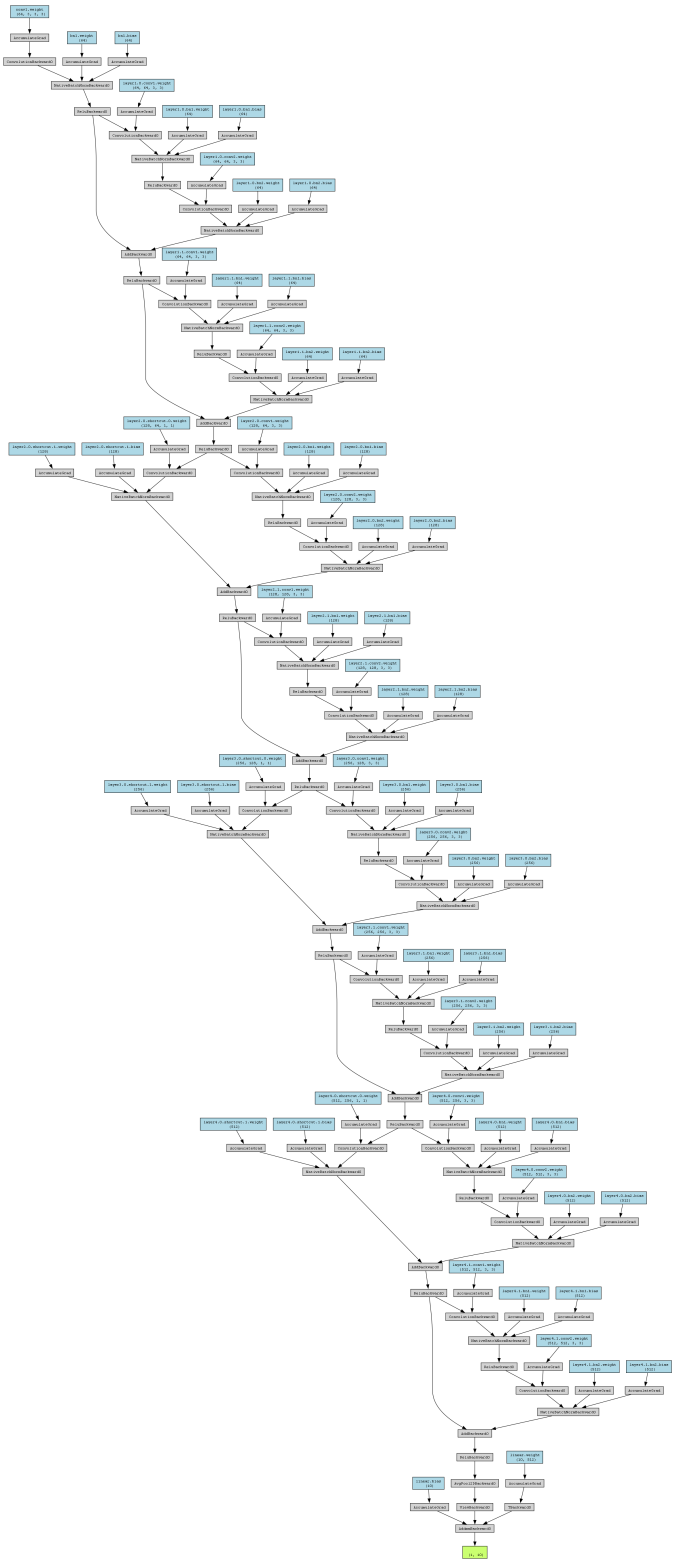


Fig. 47: ResNet architecture

```
[836  14  20  17   3   3  10   4  41  52] (0)
[  4 933   0   2   0   0   4   2   5  50] (1)
[ 38   0 763  46  26  22  77  14   7   7] (2)
[  6   1  24 827  14  46  43  23   5  11] (3)
[ 10   1  23  51 785  19  40  65   2   4] (4)
[  0   0  15 162  15 770   5  31   0   2] (5)
[  3   1   9  34   5   3 934   6   2   3] (6)
[  9   1   9  18   6  14   0 934   2   7] (7)
[ 27  23   4   3   0   1   9   0 908  25] (8)
[  1  20   1   6   0   0   3   1   7 961] (9)
  (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 86.51 %
```

Fig. 48: Confusion Matrix on Testing data for ResNet

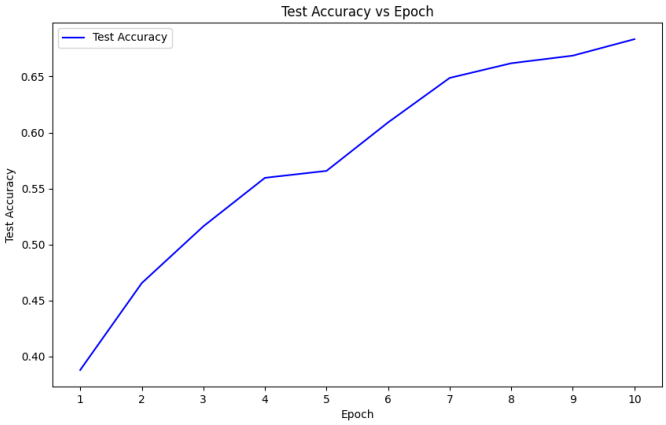


Fig. 51: Test Accuracy over Epochs for ResNeXt

```
[4373   53   82   88    7    5   42   12  116  222] (0)
[   4 4767    1   16    0    0   15    2   11  184] (1)
[ 144    5 4083  191   68   69  330   57   19   34] (2)
[  10    3   58 4416   46  188  174   56   13   36] (3)
[  32    3   56  265 4033   66  164  337   21   23] (4)
[   7    3   59  733   39 3935   38  177    2    7] (5)
[   9    7   18  103   18   34 4788    7   10    6] (6)
[  12    2   21  118   11   49   16 4743    2   26] (7)
[  73   48   13   15    0    2   17    6 4757   69] (8)
[  10   43    4   18    0    2   18   10   12 4883] (9)
  (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 89.556 %
```

Fig. 49: Confusion Matrix on Training data for ResNet

2) ResNeXt: The number of parameters in this model are 314

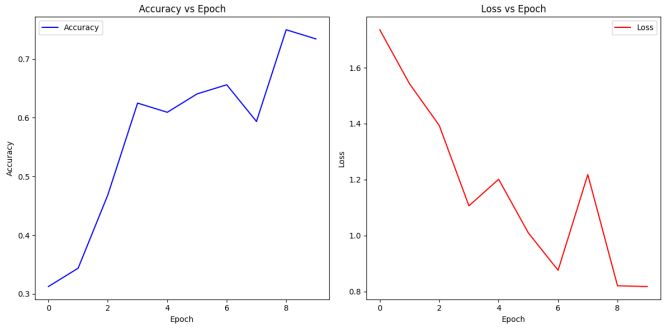


Fig. 50: Train Accuracy and Loss over Epochs for ResNeXt

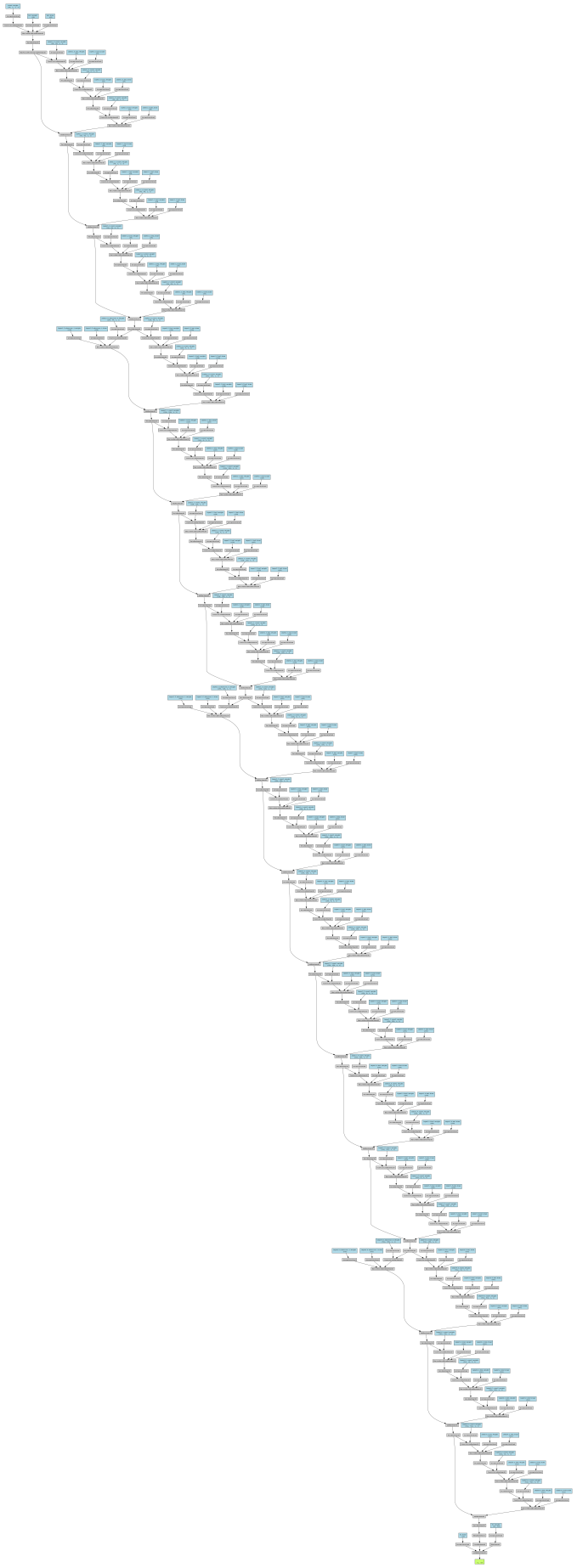


Fig. 52: ResNeXt architecture

```
[801  16  43  28  24  10  10  22  38   8] (0)
[158 706   8  16   4  12  13  11  24  48] (1)
[ 84   1 578  73  89  73  53  40   6   3] (2)
[ 19   7  76 553  35 179  66  49   9   7] (3)
[ 19   1  71  84 624  27  72  96   5   1] (4)
[  9   1  41 208  43 603  23  66   4   2] (5)
[  9   4  55  84  18  22 790   9   8   1] (6)
[  9   2  32  33  54  72  16 773   3   6] (7)
[130  26  15  13  10   1   7   5 783  10] (8)
[ 68 168   9  28   8  13  15  37  31 623] (9)
  (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 68.34 %
```

Fig. 53: Confusion Matrix on Testing data for ResNeXt

```
[4068   54  229  125  114   50   35  118  157   50] (0)
[ 764 3658   32   77   14   26   77   37  138  177] (1)
[ 349   14 3122  333  382  289  259  192   37   23] (2)
[  82   18  251 2994  161  870  297  258   48   21] (3)
[ 138    5  264  311 3306  149  291  510   19    7] (4)
[  26    4  210 1058  170 3011  148  339   21   13] (5)
[  49   23  287  476  134  124 3856   28   15    8] (6)
[  43    6  116  196  272  272   34 4026   12   23] (7)
[ 582  142   68   61   43   18   28   15 4006   37] (8)
[ 346  780   38  123   35   42   79  154  115 3288] (9)
  (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 70.67 %
```

Fig. 54: Confusion Matrix on Training data for ResNeXt

3) DenseNet: The number of parameters in this model are 727

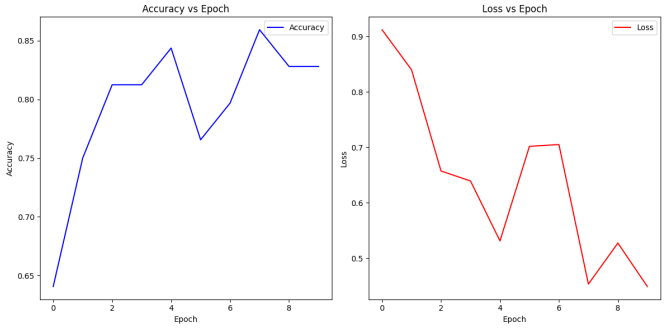


Fig. 55: Train Accuracy and Loss over Epochs for DenseNet

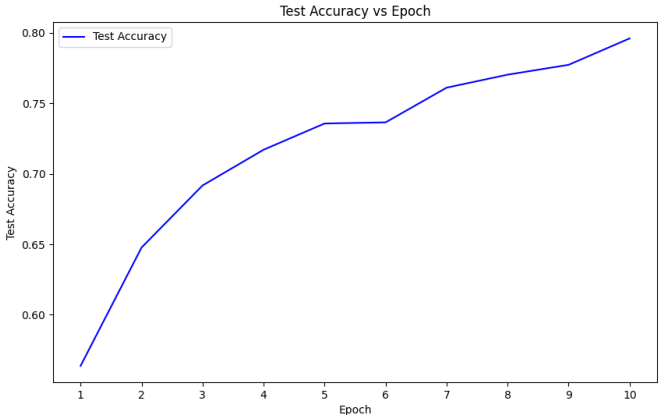


Fig. 56: Test Accuracy over Epochs for DenseNet

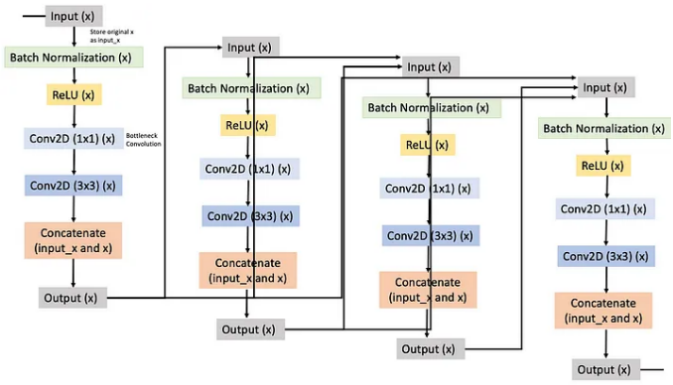


Fig. 57: DenseNet architecture, source: [1].

```
[841  11  20  17   9   3   6   7  43  43] (0)
[ 18 874   5   4   2   5  10   1  21  60] (1)
[ 49   0 688  56  71  46  48  15  15  12] (2)
[ 27   1  42 690  44 109  39  23  14  11] (3)
[  8   2  42  53 761  30  30  60  11   3] (4)
[  9   1  28 206  27 667  15  34  10   3] (5)
[  9   0  40  57  24  18 838   4   9   1] (6)
[ 12   0  12  35  21  49   6 843   5  17] (7)
[ 52  13   7  13   0   1   3   2 885  24] (8)
[ 19  61   3  16   0   3   2   5  19 872] (9)
  (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 79.59 %
```

Fig. 58: Confusion Matrix on Testing data for DenseNet

```
[4373   53   82   88    7    5   42   12  116  222] (0)
[   4 4767    1   16    0    0   15    2   11  184] (1)
[ 144    5 4083  191   68   69  330   57   19   34] (2)
[  10    3   58 4416   46  188  174   56   13   36] (3)
[  32    3   56  265 4033   66  164  337   21   23] (4)
[   7    3   59  733   39 3935   38  177    2    7] (5)
[   9    7   18  103   18   34 4788    7   10    6] (6)
[  12    2   21  118   11   49   16 4743    2   26] (7)
[  73   48   13   15    0    2   17    6 4757   69] (8)
[  10   43    4   18    0    2   18   10   12 4883] (9)
  (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 89.556 %
```

Fig. 59: Confusion Matrix on Training data for DenseNet

E. Analysis

Architecture Complexity: ResNet18, ResNeXt50, and DenseNet have more complex architectures compared to the basic and improved CNN models. This complexity allows for better feature extraction and learning capabilities, leading to higher accuracies.

Data Preprocessing: The introduction of normalization and data augmentation (random crop and flip) in the improved CNN model significantly boosts its accuracy by enhancing generalization.

Learning Rate and Mini-Batch Size: The adjustment of the learning rate and mini-batch size in the improved CNN model optimizes the learning process, allowing for more stable and effective training.

Number of Parameters: Although the improved CNN model has the same number of parameters as the basic CNN model, the more advanced models (ResNet18, ResNeXt50, DenseNet) have significantly more parameters, which correlates with their ability to learn more complex patterns. However, this also means they require more computational resources.

In summary, the ResNet18 outperforms the other models due to its balance of complexity and efficiency, followed by the improved CNN model, which shows significant improvement over the basic CNN through better preprocessing and optimized training settings. DenseNet and ResNeXt50 offer different trade-offs between parameter count, accuracy, and inference time, with DenseNet achieving higher accuracy due to its efficient feature reuse mechanism.

Model	No of Parameters	Train Accuracy	Test Accuracy	Run-time
CNN	12	54.998	46.56	0.0149
Improved CNN	12	88.656	83.62	0.0104
ResNet18	122	89.556	86.51	0.0103
ResNeXt50	314	70.670	68.34	0.0079
DenseNet	727	83.470	79.59	0.0159

Fig. 60: Analyze Table

REFERENCES

[1] C. Martins, "Densenet architecture explained with code examples," https://cdanielaam.medium.com/densenet-architecture-explained-with-code-examples-0d86d7936f83, 2023, accessed: 2024-02-14.