RBE 549: Homework 0
Alohomora

Deepak Harshal Nagle
Robotics Engineering
Worcester Polytechnic Institute
Worcester, Massachusetts 01609
Email: dnagle@wpi.edu
Telephone: (774) 519-8335
Using one late day

***Abstract*—The goal of this project is to improve upon traditional boundary detection techniques in computer vision through the implementation of a new algorithm, the probability of boundary (pb) method. In addition, this project also explores the use of deep learning for image classification on the CIFAR-10 dataset, comparing the performance of various neural network architectures. Phase 1 focuses on the development and evaluation of the pb algorithm, while Phase 2 delves into the application of deep learning techniques to improve classification accuracy.**

INTRODUCTION

In the field of computer vision, boundary detection and image classification are crucial tasks that require a range of techniques to be effectively solved. This project aims to delve into these two areas by first focusing on the implementation of a new boundary detection algorithm, the probability of boundary (pb) method. This algorithm utilizes texture, brightness, and color information to improve upon traditional techniques such as Canny and Sobel. The second phase of this project explores the application of deep learning to improve image classification accuracy on the CIFAR-10 dataset, comparing the performance of various neural network architectures like ResNet, ResNet, DenseNet. The goal of this project is to gain a deeper understanding of these important computer vision problems and the techniques used to solve them. I wish you the best of success.

PHASE 1: SHAKE MY BOUNDARY

Edge detection is a fundamental problem in the field of computer vision, and the goal of this section is to explore a new method for boundary detection called the Probability of Boundary (Pb) algorithm. This algorithm takes into account not only intensity changes, but also texture, brightness, and color information to improve upon traditional techniques such as Canny Edge and Sobel Descriptors. In this section, we will describe the process of implementing a simplified version of the Pb algorithm, called Pb-Lite, and analyze the results obtained. The steps involved include generating filter banks, developing texture, brightness, and color maps, computing gradients, and combining the results with weighted Canny and Sobel baselines. The Pb-Lite algorithm will be tested on the Berkeley Segmentation Dataset 500.

I. FILTER BANKS

In order to detect textures in an image, a collection of filters known as filter banks are used. These filter banks are a set of various filters that are applied to the image at different scales and orientations to extract the low-level features of the image. In this project, we have used three filter banks: Oriented Derivative of Gaussian (DoG) filter bank, the Leung-Malik filter bank, and the Gabor Filters. These filter banks are applied to the input image and their output is used to measure and aggregate regional texture and brightness distributions, providing an enhanced understanding of the image's features. These filter banks are illustrated in figures 1, 2, and 3.

A. Oriented Derivative of Gaussian (DoG) Filters:

These are generated by convolving a Sobel filter with a Gaussian kernel at various scales and orientations. They represent the derivative of basic Gaussian kernels at various angles.

Fig. 1. DoG filters at scales 1.0 and 0.75, and 16 orientations each

B. Leung-Malik Filters:

The LM filter bank is a set of 48 filters of different scales and orientations that are designed to capture various features in an image. It includes eighteen first and eighteen second order derivatives (for our model, at scales = $\sqrt{2}$, 2, $2\sqrt{2}$ at six orientations each) of Gaussian filters, eight Laplacian of Gaussian filters (for our model, at scales = $\sqrt{2}$, 2, $2\sqrt{2}$, 4,$3\sqrt{2}$, 6, $6\sqrt{2}$, 12) and four ordinary Gaussian kernels (for our model, $\sqrt{2}$, 2, $2\sqrt{2}$, 4). These filters are arranged in a specific way to effectively extract features at different scales and orientations. The filter bank is shown in the figure 2.

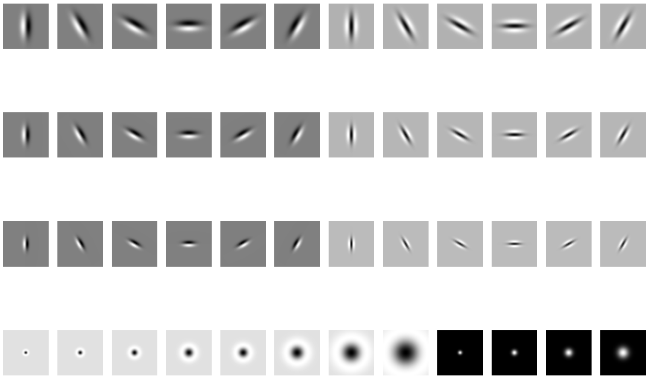


Fig. 2. LM filter bank

C. Gabor Filters:

Gabor filters are a powerful tool in image processing that are inspired by the way the human visual system processes information. They are created by combining a Gaussian kernel with a sinusoidal plane wave, which allows them to analyze specific frequency content in images in certain directions and localized regions. In this report, the Gabor filter bank used consists of multiple scales (scale = 5, 7, 9, 11, 13), at multiple sinusoidal wavelengths (lambda = 5, 8, 11, 14, 17), at 8 different orientations. These are illustrated in the figure 3, which shows the filter bank used in the project.

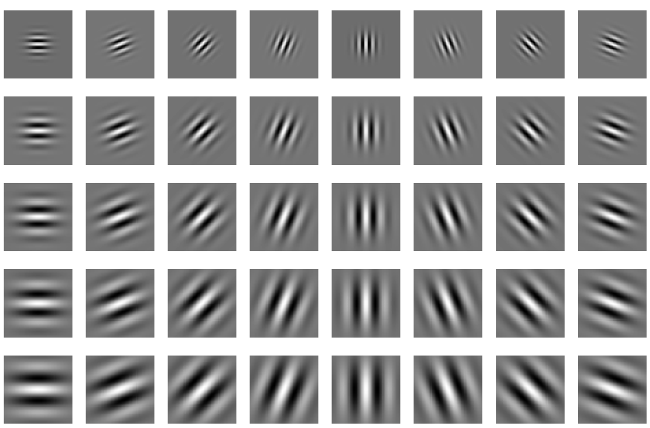


Fig. 3. Gabor filter bank

II. Texton, Brightness and Color Maps:

Texture map is generated by first convolving all the 120 filters with the input grayscale image to get a stack of 120 output kernels. Thus, each point on the original image now has 120 different values (cluster centers) assigned to it. Using KMeans clustering, we generate the Texture map where each pixel is now represented by a Texton ID (out of 0 to 63, both inclusive) assigned to it. Similar to this, we perform Color gradient based on the colored 3-layer (RGB) input to group together the similarities in colors of the image. For this color map, we perform the KMeans clustering to obtain 16 different cluster values. Finally, we perform KMeans clustering on the grayscale image to obtain the brightness map with 16 clusters.

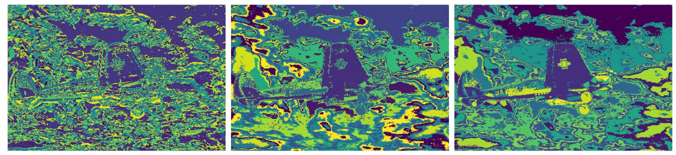


Fig. 4. Texton, Brightness and Color Maps for Image 1

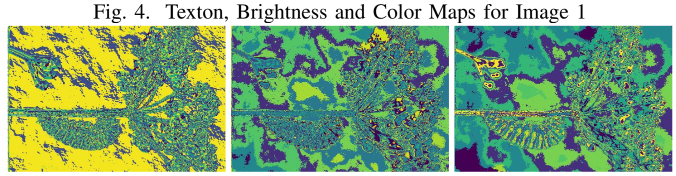


Fig. 5. Texton, Brightness and Color Maps for Image 2

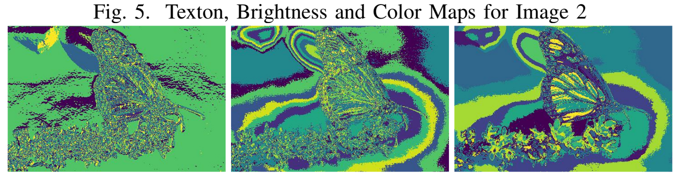


Fig. 6. Texton, Brightness and Color Maps for Image 3

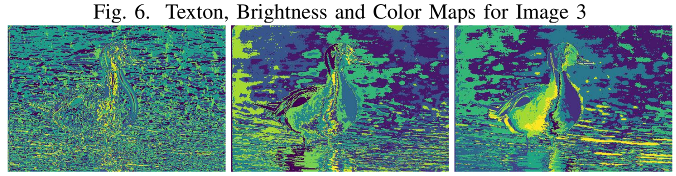


Fig. 7. Texton, Brightness and Color Maps for Image 4

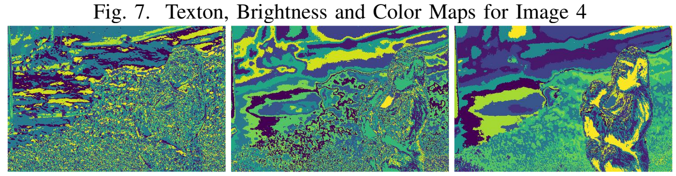


Fig. 8. Texton, Brightness and Color Maps for Image 5

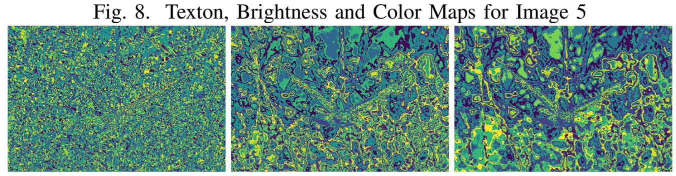


Fig. 9. Texton, Brightness and Color Maps for Image 6

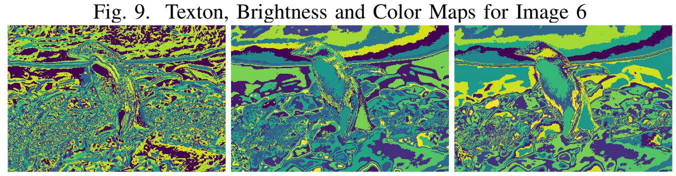


Fig. 10. Texton, Brightness and Color Maps for Image 7

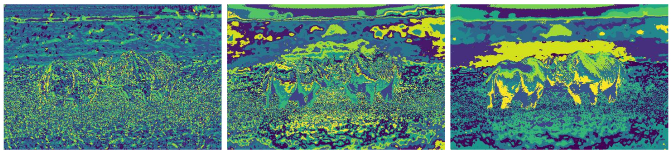


Fig. 11. Texton, Brightness and Color Maps for Image 8

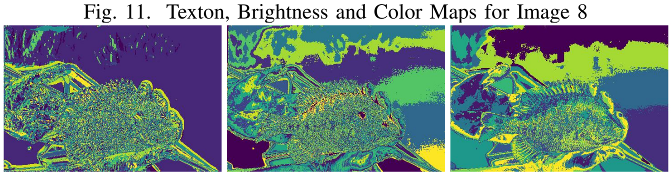


Fig. 12. Texton, Brightness and Color Maps for Image 9

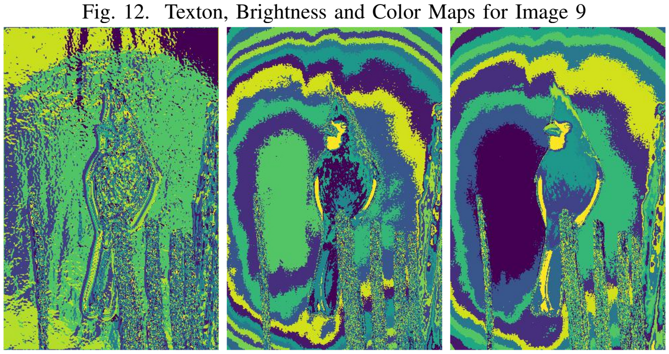


Fig. 13. Texton, Brightness and Color Maps for Image 10

III. Texture, Brightness and Color Gradients:

The Texture, Brightness and Color gradients (Tg, Bg, Cg) are computed to understand how much the distributions of Texture, Brightness and Color maps are changing at each pixel. These gradients are evaluated by convolving the half-disk masks of various orientations and scales with the maps generated earlier. The half-disk masks makes it possible to easily evaluate the gradient maps at different scales and angles, which enables us to capture the variations of texture, brightness and color at different orientations and scales. These help us calculate the chi-square distance between the filtered left and right parts around the image pixel. The chi-square distance is utilized for comparing histograms. It helps to measure how similar or different the filtered left and right parts of each image pixel are.

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

Fig. 14. Chi-square distance formula

The gradient maps provide a deeper level of analysis of the image by highlighting variations in texture, brightness, and color at each pixel. The Figures show Tg, Bg, and Cg gradients for test images.

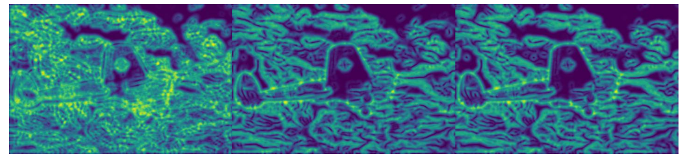


Fig. 15. Texture, Brightness and Color Gradients for Image 1

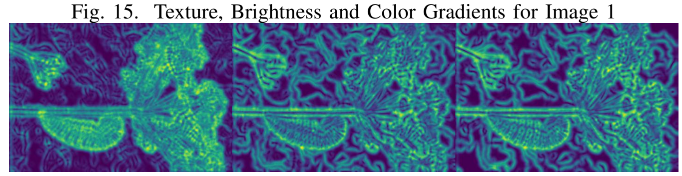


Fig. 16. Texture, Brightness and Color Gradients for Image 2

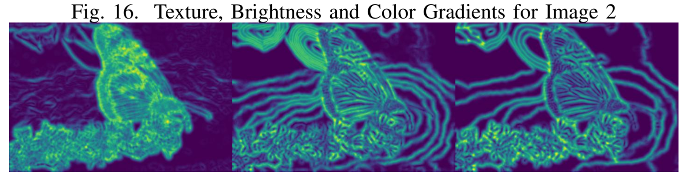


Fig. 17. Texture, Brightness and Color Gradients for Image 3

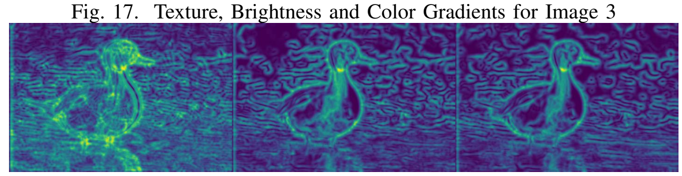


Fig. 18. Texture, Brightness and Color Gradients for Image 4

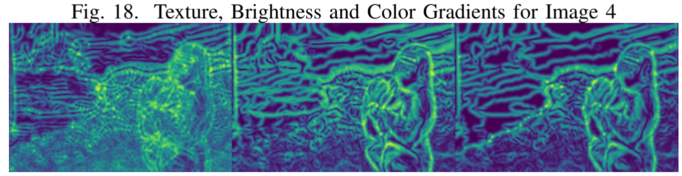


Fig. 19. Texture, Brightness and Color Gradients for Image 5

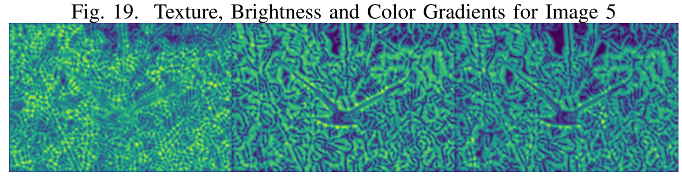


Fig. 20. Texture, Brightness and Color Gradients for Image 6

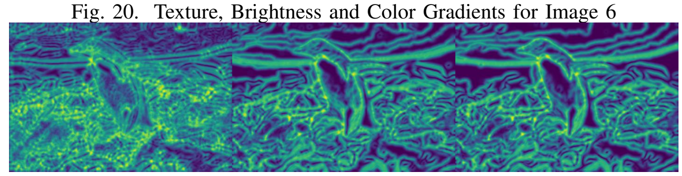


Fig. 21. Texture, Brightness and Color Gradients for Image 7

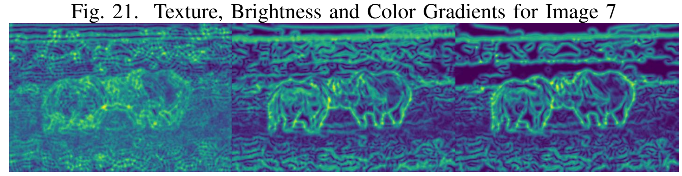


Fig. 22. Texture, Brightness and Color Gradients for Image 8

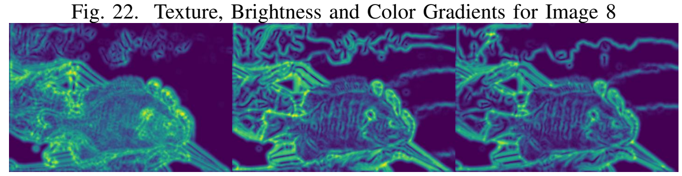


Fig. 23. Texture, Brightness and Color Gradients for Image 9

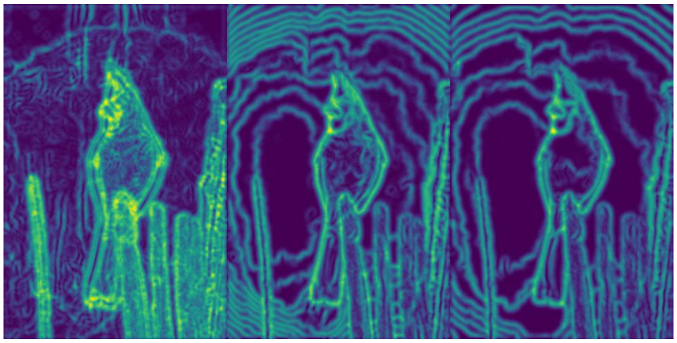


Fig. 24. Texture, Brightness and Color Gradients for Image 10

A. Pbline output:

The final pbline output is obtained by taking average of the Tg, Bg, and Cg gradients. Similarly, we perform weighted average between Sobel and Canny baselines and the resulting map is multiplied element-vise with the previous average map of Tg, Bg, and Cg gradients. This results in a map where the Texton, brightness, Color features as well as the features present in canny and sobel baselines.

$$PbEdges = \frac{(\mathcal{T}_g + \mathcal{B}_g + \mathcal{C}_g)}{3} \odot (w_1 * cannyPb + w_2 * sobelPb) \tag{1}$$

Fig. 25. pbline calculation

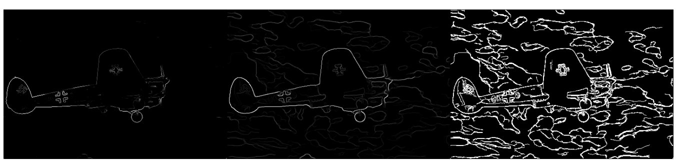


Fig. 26. Texture, Brightness and Color Gradients for Image 1

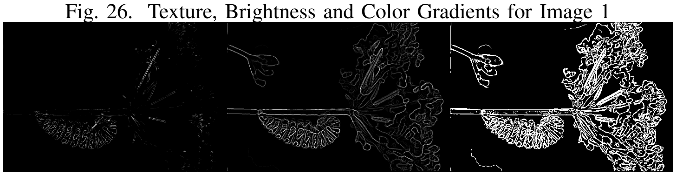


Fig. 27. Texture, Brightness and Color Gradients for Image 2

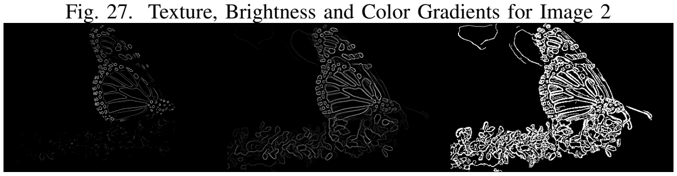


Fig. 28. Texture, Brightness and Color Gradients for Image 3

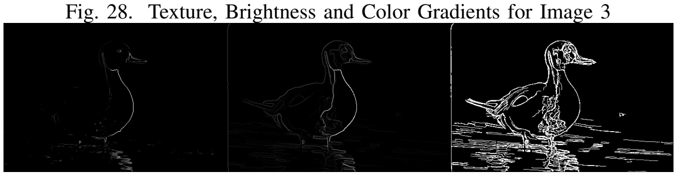


Fig. 29. Texture, Brightness and Color Gradients for Image 4

Fig. 30. Texture, Brightness and Color Gradients for Image 5

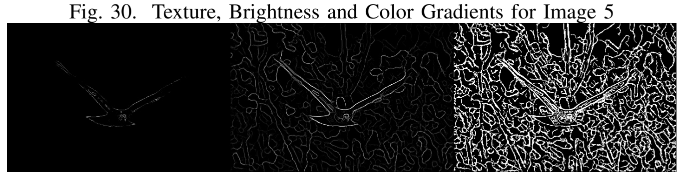


Fig. 31. Texture, Brightness and Color Gradients for Image 6

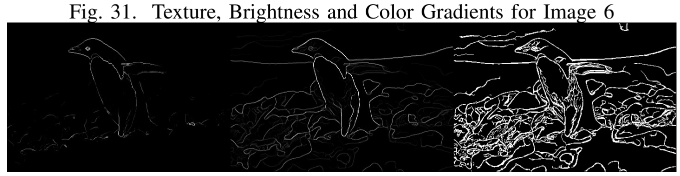


Fig. 32. Texture, Brightness and Color Gradients for Image 7

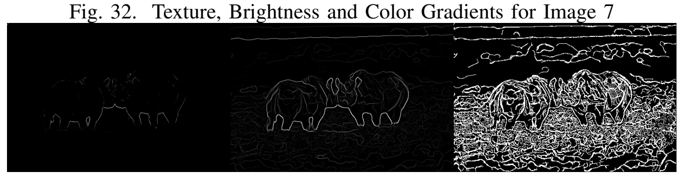


Fig. 33. Texture, Brightness and Color Gradients for Image 8

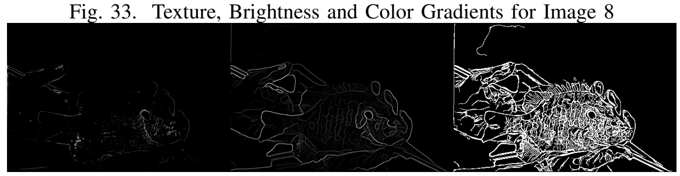


Fig. 34. Texture, Brightness and Color Gradients for Image 9

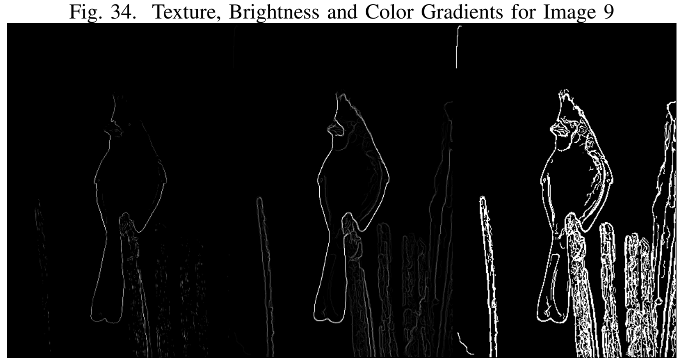


Fig. 35. Texture, Brightness and Color Gradients for Image 10

IV. Phase 2: Deep Dive on Deep Learning

Phase two involves implementing different deep-learning algorithms to classify the CIFAR-10 dataset and evaluating them based on the training and testing loss, accuracies, number of parameters, and confusion matrix. The CIFAR dataset contains 60000 images (50000 training images and 10000 testing images). These are 32x32 colored pixel images. Due to practical constraints on training speed, we will use one-tenth of the data for training purposes.

A. Convolutional Neural Network:

For quantitative analysis of different architectures, we first implement a simple 5-layered network with convolution layers and ReLU activations at each layer. Initially, the model did not converge at all. Finally, taking Adam as an optimizer (instead of SGD) and a large minibatch size (n = 100) ensured that the model converged as it is seen from the plots. Our estimate is that the model would converge even further with improved accuracy if it were implemented for a higher number of epochs.

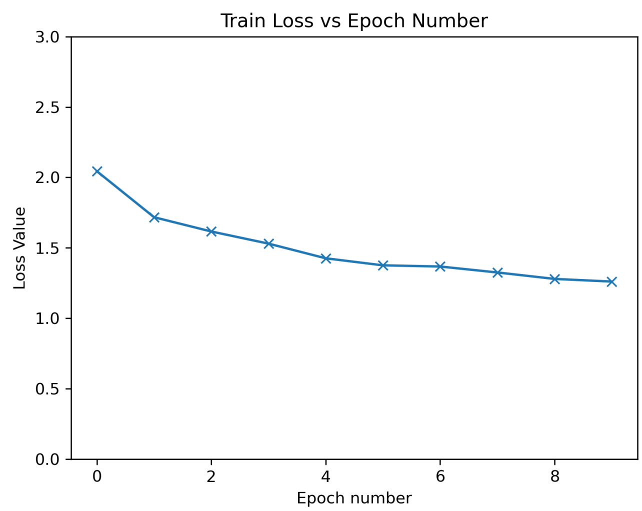


Fig. 36. Convolutional Neural Network: Loss on Train data

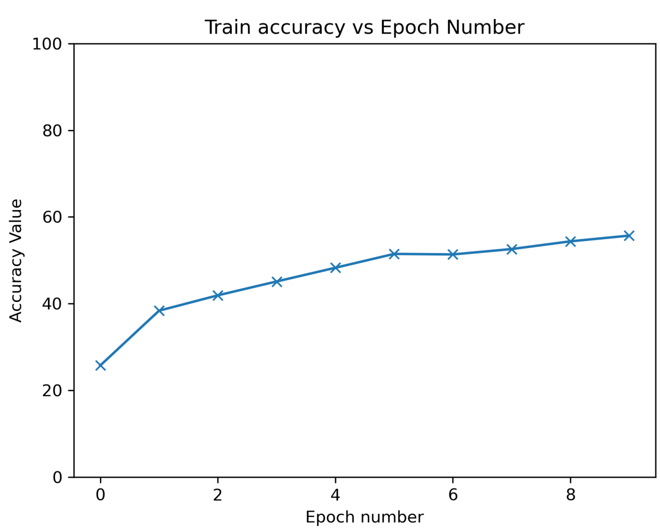


Fig. 37. Convolutional Neural Network: Accuracy on Train data

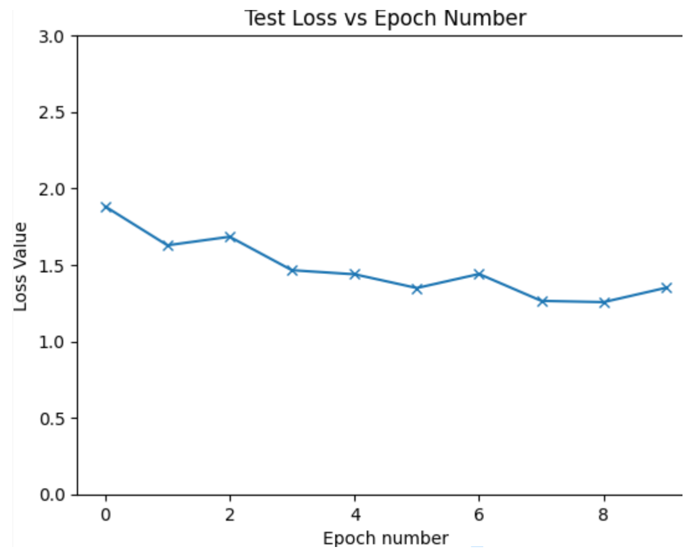


Fig. 38. Convolutional Neural Network: Loss on Test data

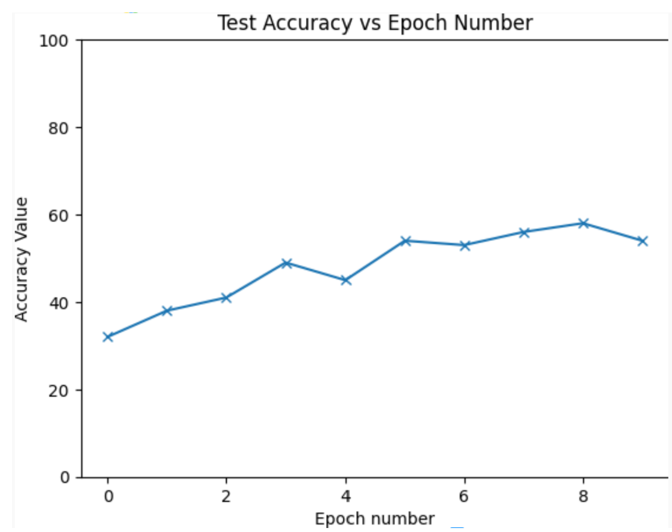
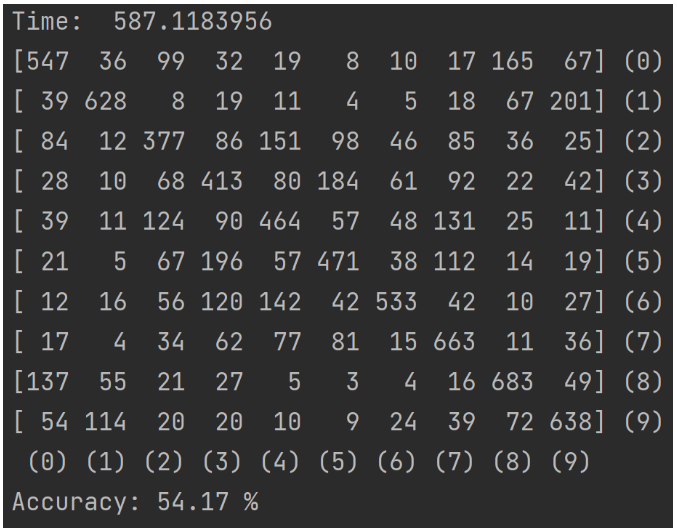


Fig. 39. Convolutional Neural Network: Accuracy on Test data

```
minibatch size: 100
devtrain (Factor by which the amount of train data is reduces): 10
Learning Rate = 0.001
optimizer = Adam
weight_decay = 0.0001
inference time = 587/50000 = 0.01174
number of types of parameters 12
total number of parameters 12513994
confusion map accuracy: 54.17%
```

Fig. 40. Convolutional Neural Network: Data and Parameters



```
Time:  587.1183956
[547  36  99  32  19   8  10  17 165  67] (0)
[ 39 628   8  19  11   4   5  18  67 201] (1)
[ 84  12 377  86 151  98  46  85  36  25] (2)
[ 28  10  68 413  80 184  61  92  22  42] (3)
[ 39  11 124  90 464  57  48 131  25  11] (4)
[ 21   5  67 196  57 471  38 112  14  19] (5)
[ 12  16  56 120 142  42 533  42  10  27] (6)
[ 17   4  34  62  77  81  15 663  11  36] (7)
[137  55  21  27   5   3   4  16 683  49] (8)
[ 54 114  20  20  10   9  24  39  72 638] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 54.17 %
```

Fig. 41. Convolutional Neural Network: Test Data Confusion Matrix

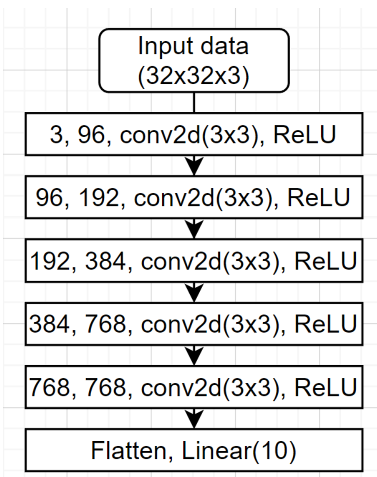


Fig. 42. Convolutional Neural Network: Architecture

B. *Improved Neural Network:*

The architecture of the previous neural network was improved by first implementing batch normalization at each of the layers. Also, ReLU was replaced by LeakyReLU of different slopes as well as with other kinds of activation functions like Softplus and Tanh. The minibatch size is reduced to 50. The learning rate was reduced by a factor of 10. As expected, the model performance was slightly better than the previous neural network architecture on the train set, even after having a smaller batch size. Our expectation is that this performance could improve even more if we use skipped connections which we will see in the next section.

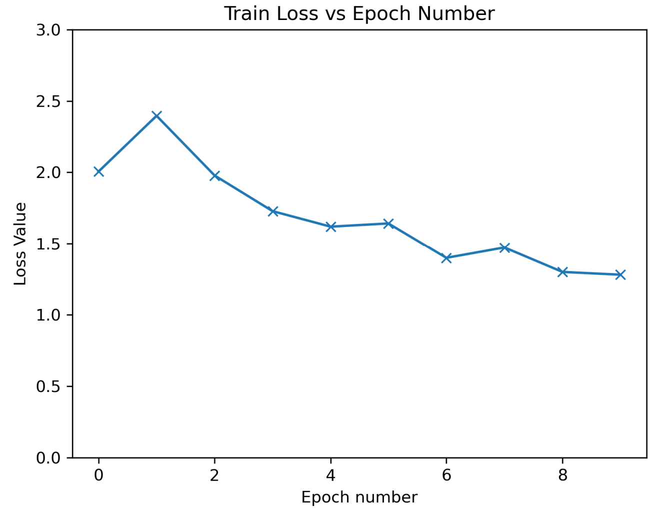
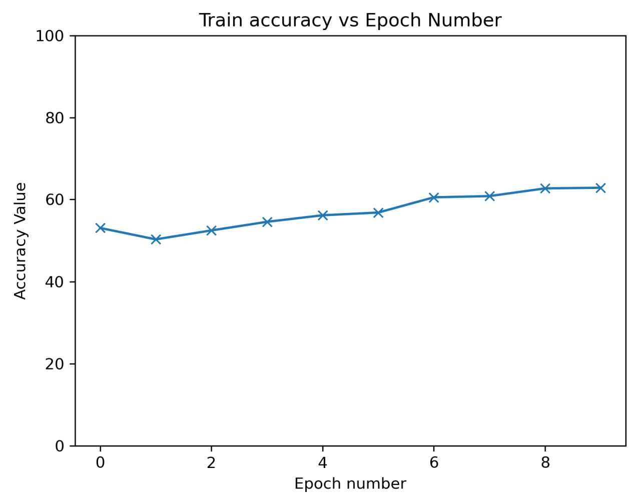


Fig. 43. Improved Neural Network: Loss on Train data



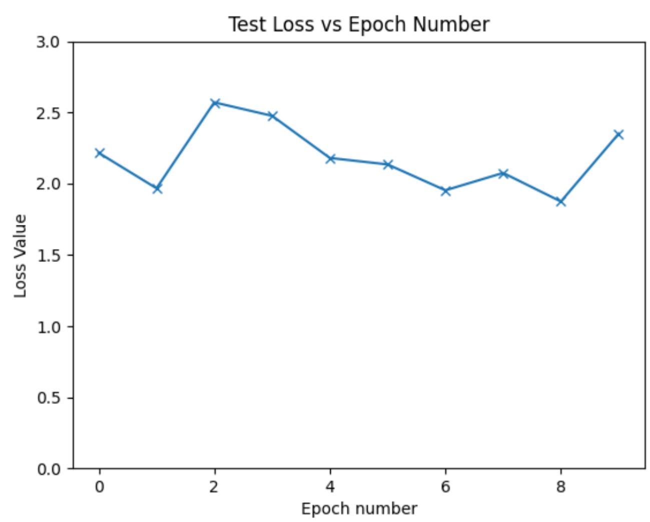


Fig. 45. Improved Neural Network: Loss on Test data

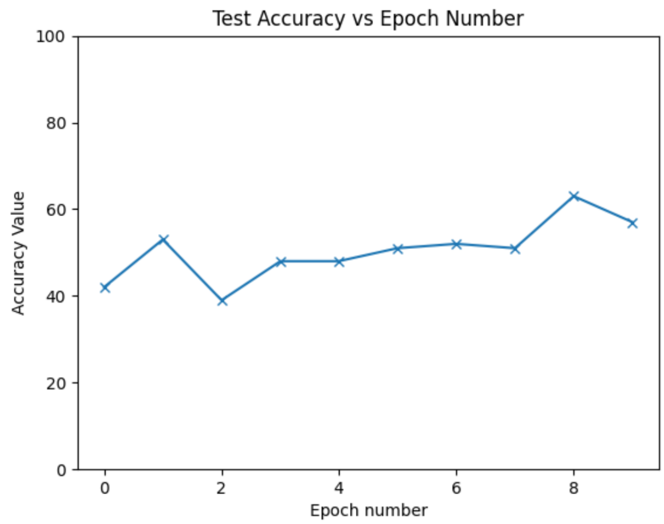


Fig. 46. Improved Neural Network: Accuracy on Test data

```
minibatch size: 100
devtrain (Factor by which the amount of train data is reduces): 10
Learning Rate = 0.0001
optimizer = Adam
number of epochs = 10
weight_decay = 0.0001
inference time = 593/50000 = 0.01186
number of types of parameters: 42
total parameters: 12518416
confusion map accuracy: 54.57%
```

Fig. 47. Improved Neural Network: Data and Parameters

```
Time:   593.1145497
[565  41  87  51  12  82  24  17  74  47] (0)
[ 42 773   7  18   4   9  18  12  25  92] (1)
[ 78  14 374 175  67 121  91  60   9  11] (2)
[ 29  17  45 565  49 124  88  60   7  16] (3)
[ 53  12  82 154 356  46 116 171   6   4] (4)
[ 19   5  54 423  26 315  52  93   7   6] (5)
[ 10  10  45 108  26  19 738  38   1   5] (6)
[ 30  11  23 151  42  48  23 654   2  16] (7)
[201  83  22  55  15  17  20   5 547  35] (8)
[ 59 197   6  39  10   7  16  49  47 570] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 54.57 %
```

Fig. 48. Improved Neural Network: Test Data Confusion Matrix

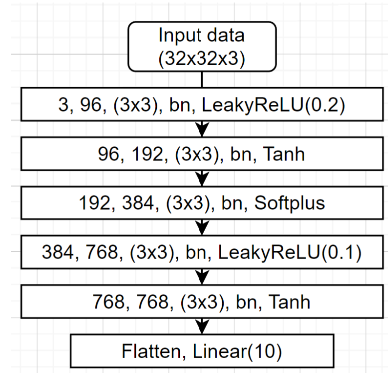


Fig. 49. Improved Neural Network: Architecture

C. ResNet:

The architecture consisted 11-layered equivalent of ResNet. It involved different blocks of convolution layers. As the architecture becomes deep, there is a high tendency for vanishing gradients. This results in the saturation of the model with very small gradients and unchanging weights. To avoid this, skipped connections are implemented which retain the relevant information and avoid vanishing gradients. In this model, we used two skipped connections, one between the outputs of the second and third blocks and the other between the fourth and fifth blocks. The plots indicate a clear decrease in the loss for both train and test sets. While the loss did not reach the minimum value, looking at the trend, we estimate that increasing the number of epochs will result in the loss decreasing even further, as deeper networks need more epochs to train. Also, due to the GPU constraints, the batch size used was just 20, still, it managed to canverge and had a performance similar to the previous network.

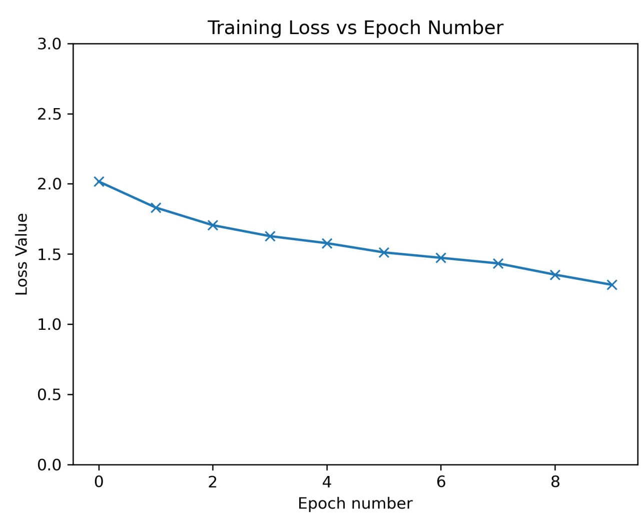


Fig. 50. ResNet: Loss on Train data

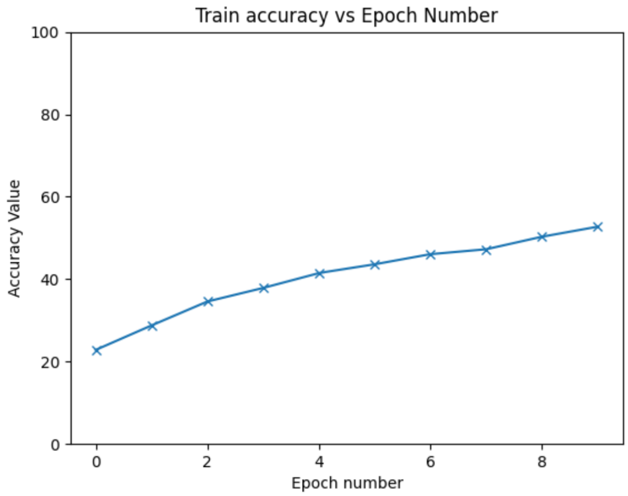


Fig. 51. ResNet: Accuracy on Train data

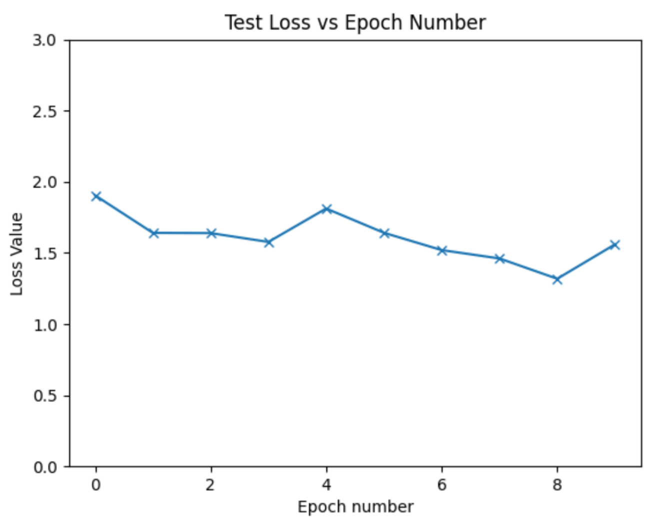


Fig. 52. ResNet: Loss on Test data

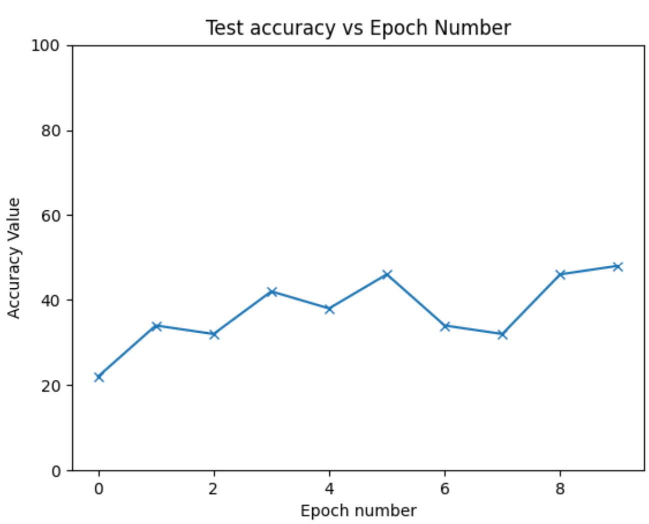


Fig. 53. ResNet: Accuracy on Test data

```
minibatch size: 20
devtrain (Factor by which the amount of train data is reduces): 10
Learning Rate = 0.001
optimizer = Adam
number of epochs = 10
weight_decay = 0.0001
number of types of parameters 79
total number of parameters 9987210
inference time: 1023/50000 = 0.02046
confusion map accuracy: 49.72%
```

Fig. 54. ResNet: Data and Parameters

```
Time:  1022.9219619
[363 106 130  12  10  11  32  41 265  30] (0)
[ 23 729  12   2  19   1  58  19  82  55] (1)
[ 65  12 417  56  98  40 185 103  18   6] (2)
[  9  23  84 208  67 160 260 164   8  17] (3)
[ 24  15 166  47 326  38 164 196  15   9] (4)
[  5  13  96 112  42 337  78 301   5  11] (5)
[  1   2  87  26  92  17 734  39   1   1] (6)
[ 13  23  37  20  54  48  41 734   3  27] (7)
[ 90 116  46  19   7   8  49  11 610  44] (8)
[ 11 240   6  14  25   3  45  35 107 514] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 49.72 %
```

Fig. 55. ResNet: Test Data Confusion Matrix

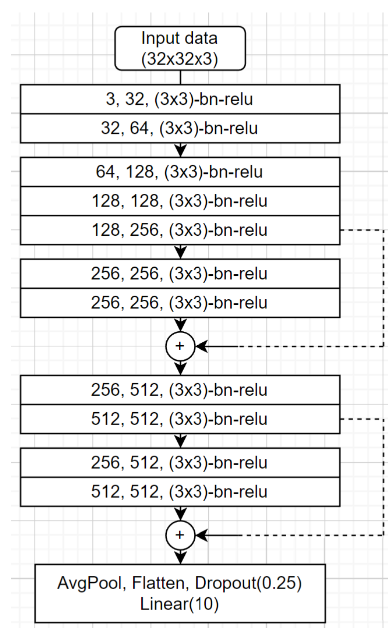


Fig. 56. ResNet: Architecture

D. ResNext:

ResNext is a modification to ResNet. It is observed from the ResNext Paper[***] that in deeper networks, the performance improves significantly if instead of making the network deeper, we make it wider. First, the layers are arranged into different blocks. Input to each block is separately (”k” times) passed through the same set of layers, and thus, ”k” different outputs are obtained. This ”k” is the cardinality of the network blocks All these outputs are added together before putting them into the next block. I implemented five blocks each containing six layers, i.e. 30-layered network, with k=8. This 30-layered structure with **cardinality ”8”** needs a higher batch size, but due to limits on GPU, I could only use a batch size of 25. Further, such big models should be trained for a much higher number of epochs. However, due to the slow training speed, I could only train it for 10 epochs, though it still managed to converge.

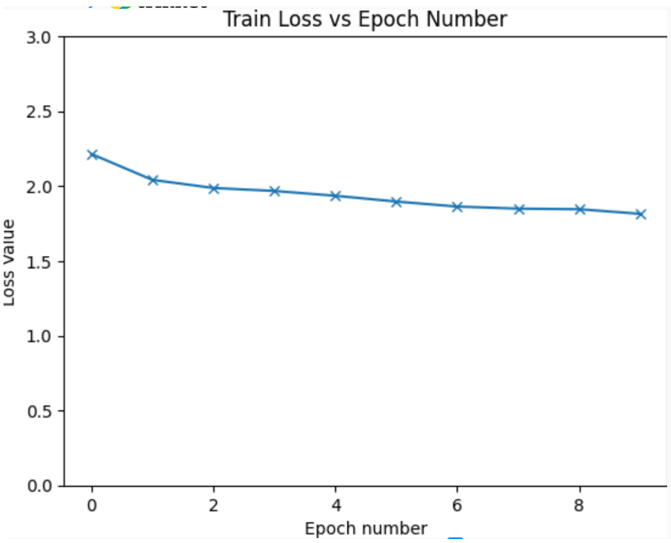


Fig. 57. ResNext: Loss on Train data

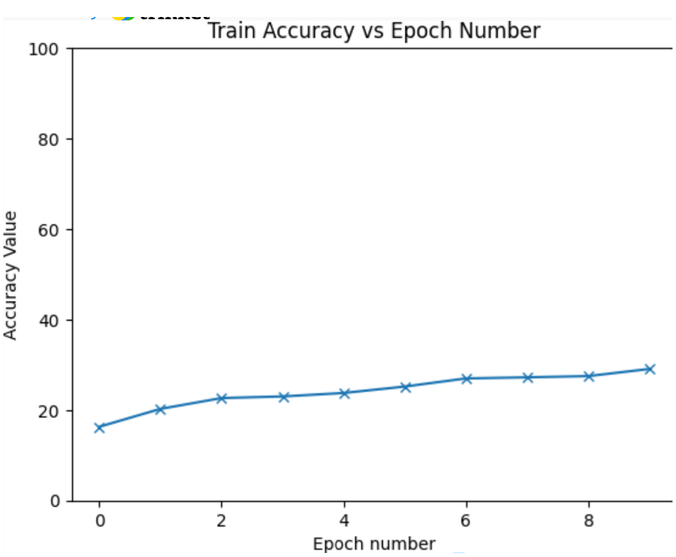


Fig. 58. ResNext: Accuracy on Train data

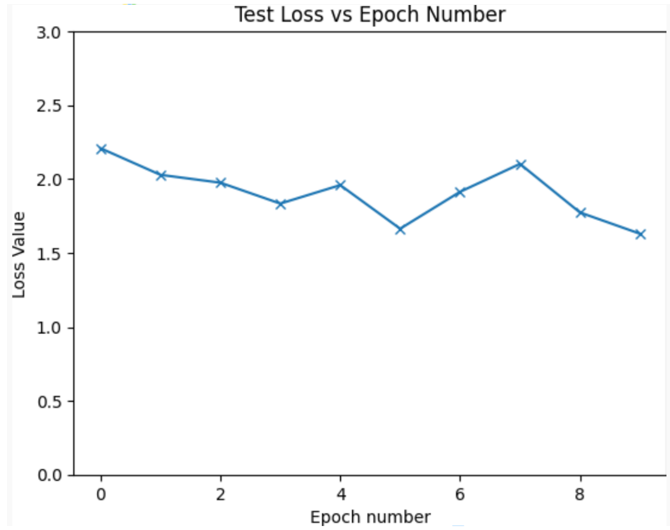


Fig. 59. ResNext: Loss on Test data

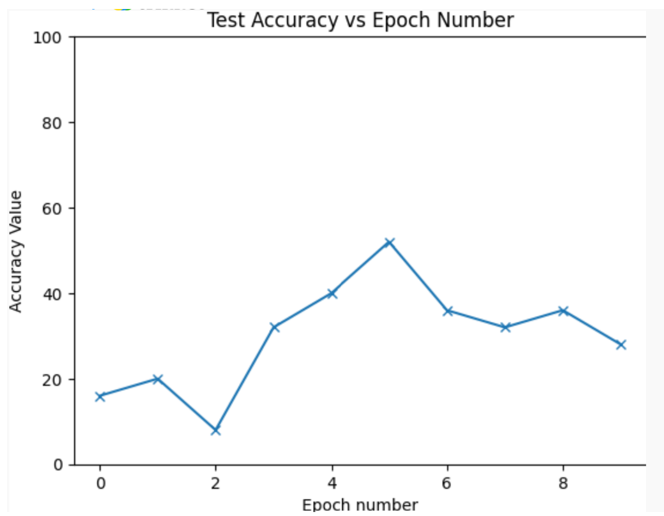


Fig. 60. ResNext: Accuracy on Test data

```
minibatch size: 25
devtrain (Factor by which the amount of train data is reduces): 10
number of epochs = 10
cardinality = 8
Learning Rate = 0.001
optimizer = Adam
weight_decay = 0.0001
inference time: 1375/50000 = 0.0275
number of types of parameters: 212
total number of parameters: 38214
confusion map accuracy: 26.93%
```

Fig. 61. ResNet: Data and Parameters

```
Time:  1375.0867163
[215  78   5   0  54   0   0  35 592  21] (0)
[ 22 530   0   0  37   0   0  39 345  27] (1)
[ 81  62  26   0 555  12   0 121 128  15] (2)
[ 33  68   4   0 544  39   0 224  72  16] (3)
[ 44  35   6   0 683  24   0 144  56   8] (4)
[ 27  72   8   0 524  49   0 241  72   7] (5)
[ 11  51   2   0 645  47   0 215  21   8] (6)
[ 49 118   0   0 326  30   0 365  68  44] (7)
[ 80 105   3   0  34   0   0  14 755   9] (8)
[ 66 397   0   0  31   1   0  76 359  70] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 26.93 %
```

Fig. 62. ResNext: Test Data Confusion Matrix

V. Conclusion and discussion

Model	No. of Parameters	Final Test Accuracy	Inference Run Time
base CNN	12513994	54.17%	0.01174
Improved CNN	12518416	54.57%	0.01186
ResNet	9987210	49.72%	0.02046
ResNext	38214	26.93%	0.0275

Fig. 63. Comparison Table

We have implemented various neural network models and analyized their performance. All these models are based on different deep learning techniques and have different depths. Although due to limitations the comparison is not perfect, still few things can be concluded from it. For one, the comparison between basic CNN and improved CNN tells us that, though initial accuracy is much higher for improved CNN, the basic CNN reaches the similar value after a few epochs. It is true from the comparison table that, larger, deeper models (ResNet and RangeNet) have longer inference time, even if they are implemented with much less number of parameters. Also, based on the test accuracies we can conclude that, in general, the bigger models perform poorly in the beginning of the training while they may outperform smaller models in the long run. These models (ResNet and RangeNet) expect more data for training. RangeNet in particular, would perform better if the cardinality ('k') and the number of epochs are increased. Finally, adding too many layers deep into the newtwork results in negligible gradient values. To avoid these, skipped-connections are useful.

References

[1]

1 Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun. Deep Residual Learning for Image Recognition (2015)
2 Pablo Ruiz. ResNet for CIFAR-10 (2018)
3 Lei Sun. ResNet on Tiny ImageNet (2017)