RBE 549 HW0: Alohamora

Cole Parks
Email: cparks@wpi.edu
Due: 1/12/2024
Using 2 late days

***Abstract*—This report details the results of the first homework assignment for Homework 0: Alohomora for RBE 549. In Phase I, a simple version of pb (probability of boundary) was implemented through the generation of filter sets from scratch to extract edges in an image. Phase II was a deep dive into deep learning, with the creation and analysis of multiple convolutional neural networks (CNNs) based on the architectures of ResNet, ResNeXt, and DenseNet.**

I. Shake My Boundary

In this section, we implemented a simplified version of the pb boundary detection algorithm (pb-lite). The output of our algorithm was a per-pixel probability of boundary, which we will compare against the Canny and Sobel baselines.

A. Filter Banks

The first step of the pb-lite boundary detection pipeline is to filter the image with a set of filter banks. We created three different sets of filter banks for this purpose. Once we convolved the image with these filters, we generated a texton map which depicts the texture in the image by clustering the filter responses. The three filters in the filter bank were oriented Difference of Gaussian (DoG) filters, Leung-Malik filters, and Gabor filters.

1) Oriented DoG filters: A simple but effective filter bank is a collection of oriented Derivative of Gaussian (DoG) filters. These filters were created by convolving a Sobel filter and a Gaussian kernel and then rotating the result. The resulting filter bank had two scales and sixteen orientations, shown in Figure 1 below.

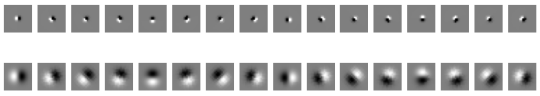

Fig. 1. Oriented DoG Filter Bank.

2) Leung-Malik Filters: The Leung-Malik (LM) filters are a set of 48 filters with multiple scales and orientations. It contains first and second order derivatives of Gaussians at six orientations and three scales, eight Laplacian of Gaussian (LOG) filters, and four Gaussians. We considered two versions of the LM filter bank. In LM Small (LMS), the filters occur at basic scales $\sigma = \{1, 2 - \sqrt{2}, 2, 2\sqrt{2}\}$. The first and second derivative filters occur at the first three scales with an elongation factor of 3, i.e., ($\sigma_x = \sigma$ and $\sigma_y = 3\sigma_x$). The Gaussians occur at the four basic scales while the 8 LOG filters occur at σ and 3σ. For LM Large (LML), the filters occur at the basic scales $\sigma = \{2 - \sqrt{2}, 2, 2\sqrt{2}, 4\}$. The LM filter bank is shown in Figure 2 below, with LMS being on the top and LML on the bottom.

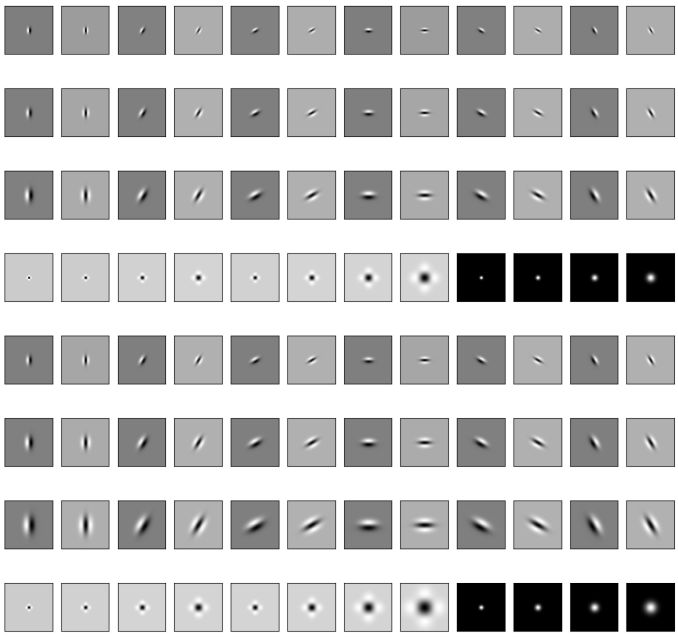

Fig. 2. Leung-Malik Filter Bank.

3) Gabor Filters: Gabor Filters are designed based on the filters in the human visual system. A gabor filter is a gaussian kernel function modulated by a sinusoidal plane wave. A sample of gabor filters is shown in Figure 3 below.

B. Texton, Brightness, and Color Maps T

Texton maps are used to depict the texture in the image by clustering the filter responses. Taking a combination of the three filter banks, we convolved the image with each filter in the filter bank. We then clustered the filter responses at all pixels in the image into K textons using kmeans clustering. Each pixel was then represented by a one dimensional, discrete cluster ID (or "Texton ID") instead of a vector of high-dimensional, real-valued filter responses. A similar approach was used to create brightness and color maps, using either the

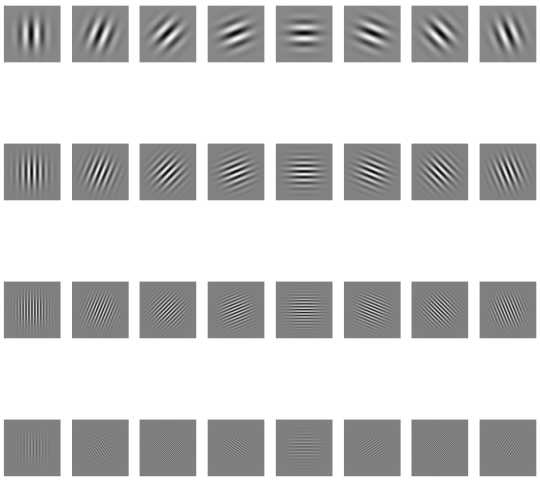

Fig. 3. Gabor Filter Bank.

grayscale or RGB color channels respectively. The maps are shown in Figures 4-13 below.

Fig. 4. T, B, and C for Image 1

Fig. 5. T, B, and C for Image 2

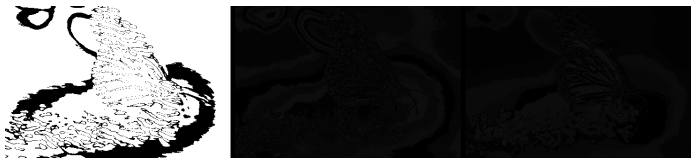

Fig. 6. T, B, and C for Image 3

C. Texture, Brightness, and Color Gradients T_g, B_g, C_g

To obtain T_g, B_g, C_g, we needed to compute differences of values across different shapes and sizes. This was achieved very efficiently by the use of Half-disc masks. The half-disc masks were simply (pairs of) binary images of half-discs. This was very important because it allowed us to compute the χ^2 (chi-square) distances using a using this equation:

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

These maps were convolved with the texton, brightness, and color maps to obtain their respective gradients. The half-disc masks are shown in Figure 14, with four sizes and eight

Fig. 7. T, B, and C for Image 4

Fig. 8. T, B, and C for Image 5

Fig. 9. T, B, and C for Image 6

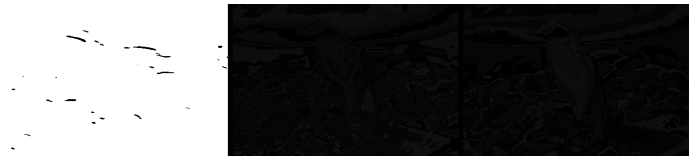

Fig. 10. T, B, and C for Image 7

Fig. 11. T, B, and C for Image 8

Fig. 12. T, B, and C for Image 9

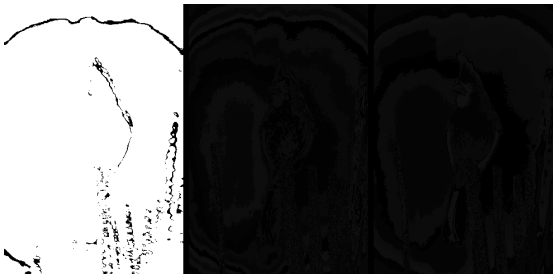

Fig. 13. T, B, and C for Image 10

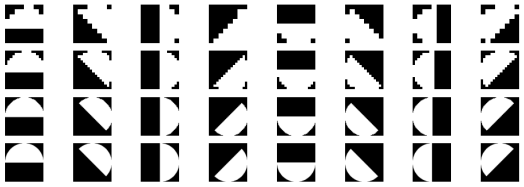

Fig. 14. Half-disc Masks

orientations. The resulting gradients are shown in Figures 15-24 below.

Fig. 15. T_g, B_g, and C_g for Image 1

Fig. 16. T_g, B_g, and C_g for Image 2

Fig. 17. T_g, B_g, and C_g for Image 3

Fig. 18. T_g, B_g, and C_g for Image 4

Fig. 19. T_g, B_g, and C_g for Image 5

Fig. 20. T_g, B_g, and C_g for Image 6

Fig. 21. T_g, B_g, and C_g for Image 7

Fig. 22. T_g, B_g, and C_g for Image 8

Fig. 23. T_g, B_g, and C_g for Image 9

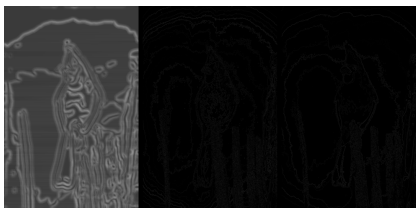

Fig. 24. T_g, B_g, and C_g for Image 10

D. Pb-lite Output

To improve the edge detection and sharpen the edges, we combined the information from the features with an average of the Sobel and Canny baselines using the following equation:

$$PbEdges = \frac{T_g + B_g + C_g}{3} \odot (0.5 \cdot cannyPb + 0.5 \cdot sobelPb) \tag{2}$$

We chose w1 and w1 to be 0.5, and the resulting pb-lite images are shown in Figures 25-34 below.

Fig. 25. Pb-Lite output, Canny output, and Sobel output for Image 1

Fig. 26. Pb-Lite output, Canny output, and Sobel output for Image 2

Fig. 27. Pb-Lite output, Canny output, and Sobel output for Image 3

Fig. 28. Pb-Lite output, Canny output, and Sobel output for Image 4

Fig. 29. Pb-Lite output, Canny output, and Sobel output for Image 5

Fig. 30. Pb-Lite output, Canny output, and Sobel output for Image 6

Fig. 31. Pb-Lite output, Canny output, and Sobel output for Image 7

Fig. 32. Pb-Lite output, Canny output, and Sobel output for Image 8

Fig. 33. Pb-Lite output, Canny output, and Sobel output for Image 9

Fig. 34. Pb-Lite output, Canny output, and Sobel output for Image 10

E. Evaluation

The pb-lite filter outperforms the Canny and Sobel baselines in edge detection for several reasons. Firstly, the pb-lite filter combines the information from the features using an average of the Sobel and Canny baselines. This combination allows for a more comprehensive representation of the edges in the image. By taking into account both the gradient magnitude and the gradient direction, the pb-lite filter is able to capture a wider range of edge features. Secondly, this pb-lite filter applies a weighting factor to the combined features, with equal weights given to the Canny and Sobel outputs. This balanced weighting ensures that both the high-frequency details captured by the Canny filter and the gradient information captured by the Sobel filter are properly incorporated into the final edge map. Lastly, the pb-lite filter produces sharper and more well-defined edges compared to the Canny and Sobel baselines. This is evident in the resulting pb-lite images, where the edges are more pronounced and accurately delineated. The pb-lite filter is able to enhance the edges by effectively combining the strengths of the Canny and Sobel filters. Overall, the pb-lite filter provides a superior edge detection result by leveraging the complementary strengths of the Canny and Sobel baselines.

II. Deep Dive on Deep Learning

In this section, multiple neural networks architecures were explored and compared on various criteria.

A. Base/Improved Model

The base model was a simple convolutional neural network with 4 convolutional layers, 2 max pooling layers, and 2 fully connected layers. The addition of the max pooling layers, normalization, and random rotation and cropping after the initial network improved the accuracy of the network on the test set. The model structure is shown in Figure 35. The model was trained for 10 epochs with a batch size of 64 and a learning rate of 0.001, momentum of 0.8, and with cross-entropy loss for the loss function. The model has 917750 parameters. The training accuracy, training loss, and testing accuracies over epochs are shown in Figures 36, 37, and 38. In addition, the confusion matrices on training and testing data are in Figures 39 and 40.

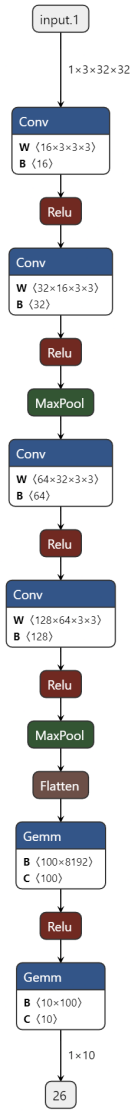

Fig. 35. Base Model Architecture

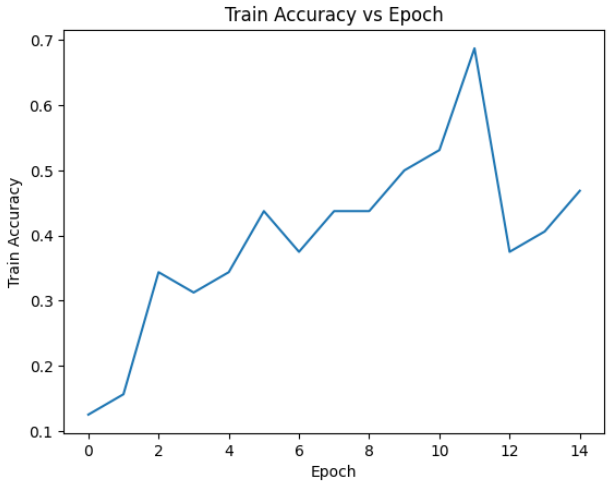

Fig. 36. Base Model Training Accuracy

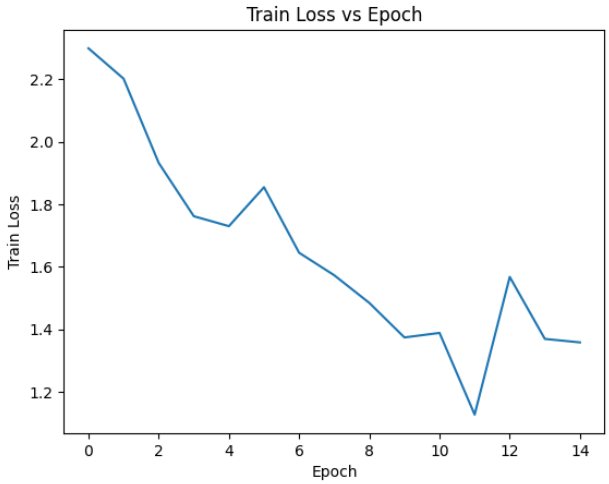

Fig. 37. Base Model Training Loss

B. ResNet

The ResNet architecture was implemented with four residual blocks, each with six convolutional layers. The model structure is shown in Figure 41. Skip connections were added between the convolutional layers to avoid vanishing gradients and allow for deeper networks. The model was trained for 15 epochs with a batch size of 64, with cross-entropy loss for the loss function. The model has 1110650 parameters. The training accuracy, training loss, and testing accuracies over epochs are shown in Figures 42, 43, and 44. The ResNet architecture performed marginally better than the base model, with a test accuracy of 68.6% In addition, the confusion matrices on training and testing data are in Figures 45 and 46.

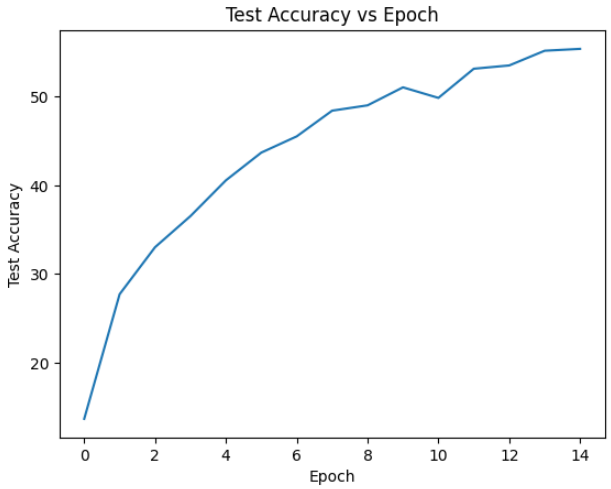

Fig. 38. Base Model Training Accuracy

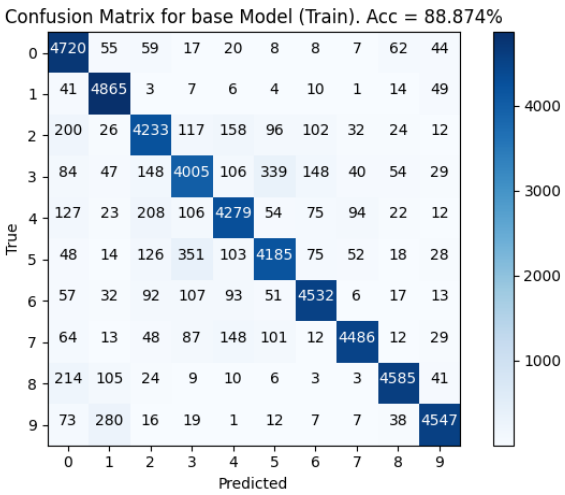

Fig. 39. Base Model Training Confusion Matrix

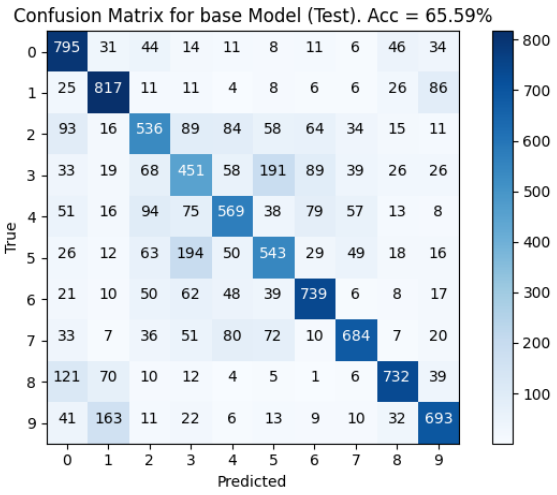

Fig. 40. Base Model Testing Confusion Matrix

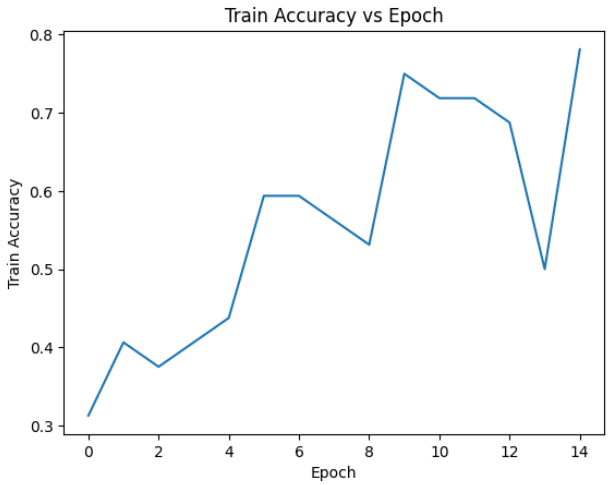

Fig. 42. ResNet Training Accuracy

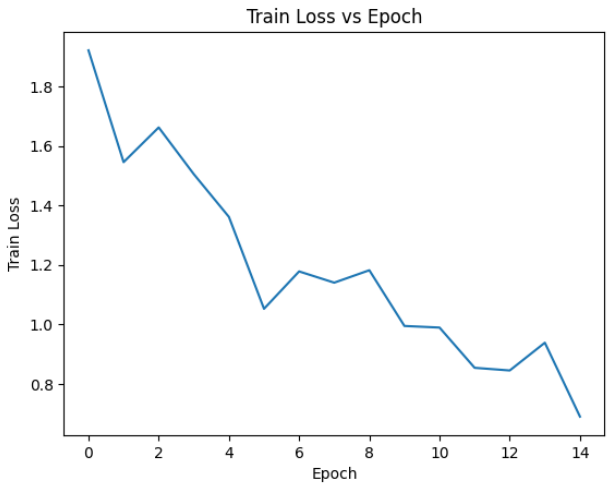

Fig. 43. ResNet Training Loss

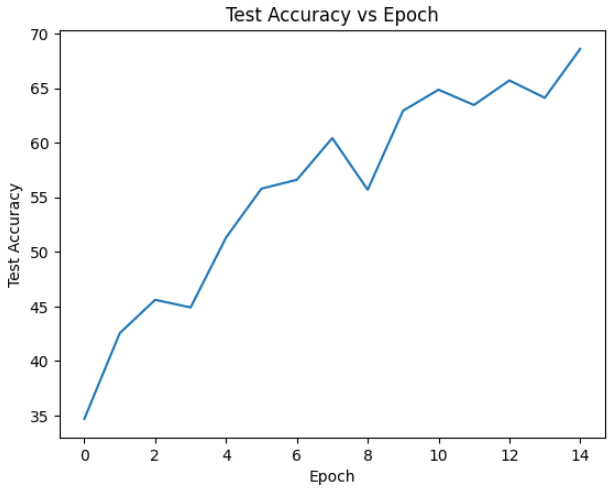

Fig. 44. ResNet Training Accuracy

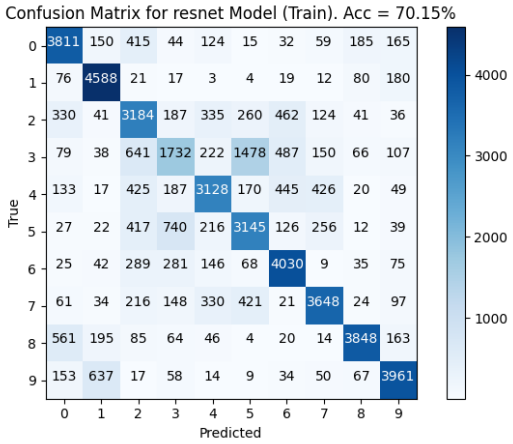

Fig. 45. ResNet Training Confusion Matrix

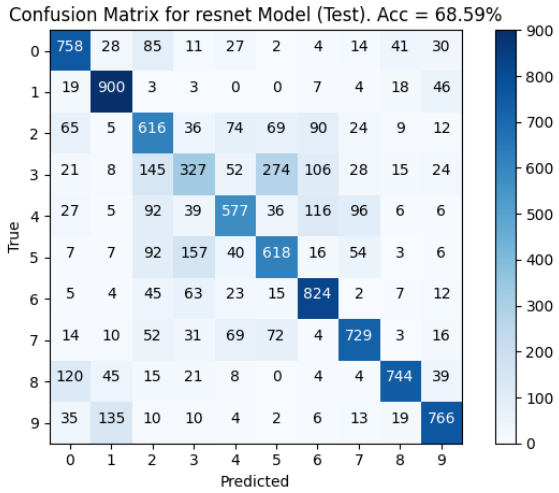

Fig. 46. ResNet Testing Confusion Matrix

C. ResNeXt

The version of ResNeXt implemented had 11 residual blocks, each with four convolutional layers. The model structure is shown in Figure 47. Skip connections were added between the convolutional layers to avoid vanishing gradients in this deeper network structure. The model was trained for 15 epochs (due to time constraints) with a batch size of 32, with cross-entropy loss for the loss function. Unfortunately, the testing accuracy of this model was only 30.36%, but could have been higher if time allowed for more epochs. The model has 120410 parameters. The training accuracy, training loss, and testing accuracies over epochs are shown in Figures 48, 49, and 50. In addition, the confusion matrices on training and testing data are in Figures 51 and 52. When training ResNeXt, there were often dimension mismatches between the output of the residual block and the input to the next residual block. This was solved by adding a convolutional layer with a kernel size of 1 and a stride of 2 to the skip connection. This allowed the output of the residual block to be the same size as the input to the next residual block. Also, a deeper network than ResNet was implemented to try to improve the accuracy of the network, but this did not end up yielding any better results.

D. DenseNet

The densenet architecture was implemented where outputs of each convolutional layer are passed to future layers. Unfortunately, the testing accuracy of this model was only 10.3%, showing that the model did not learn, just picking one of two classes for all images. The model had 46378 parameters, and was trained with a batch size of 16 due to GPU memory constraints. The model was trained for 15 epochs, and used cross-entropy loss for the loss function. The training accuracy, training loss, and testing accuracies over epochs are shown in Figures 53, 54, and 55. In addition, the confusion matrices on training and testing data are in Figures 56 and 57.

E. Comparison

Between the basic network, ResNet, ResNeXt, and DenseNet, ResNet performed the best, with a testing accuracy of 68.6%. With more time and tuning, the DenseNet could perform better, since theoretically the increase in connections between layers should allow for better learning.

Model	Parameters	Train Accuracy	Test Accuracy
Base	917750	88.87%	65.59%
ResNet	1110650	70.15%	68.59%
ResNeXt	120410	29.43%	30.36%
DenseNet	46378	100%	10.3

TABLE I
COMPARISON OF MODELS

REFERENCES

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun *Deep Residual Learning for Image Recognition* arXiv:1512.03385v1 [cs.CV] 10 Dec 2015

[2] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, Kaiming He *Aggregated Residual Transformations for Deep Neural Networks* arXiv:1611.05431v2 [cs.CV] 22 Dec 2016

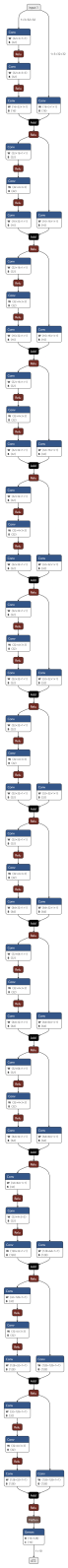

Fig. 47. ResNeXt Architecture

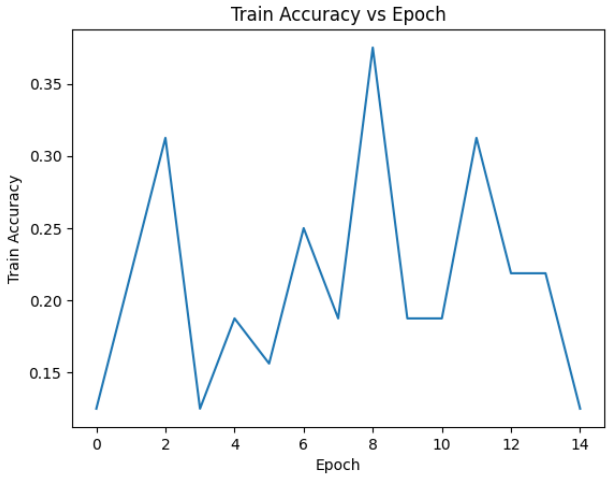

Fig. 48. ResNeXt Training Accuracy

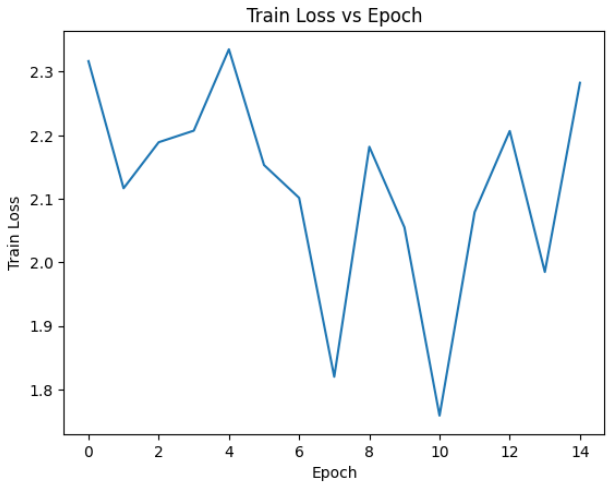

Fig. 49. ResNeXt Training Loss

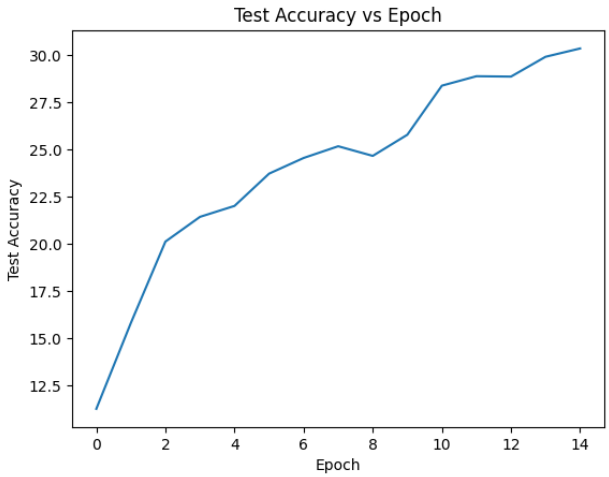

Fig. 50. ResNeXt Training Accuracy

[3] Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger *Densely Connected Convolutional Networks* arXiv:1608.06993v5 [cs.CV] 26 Jan 2018

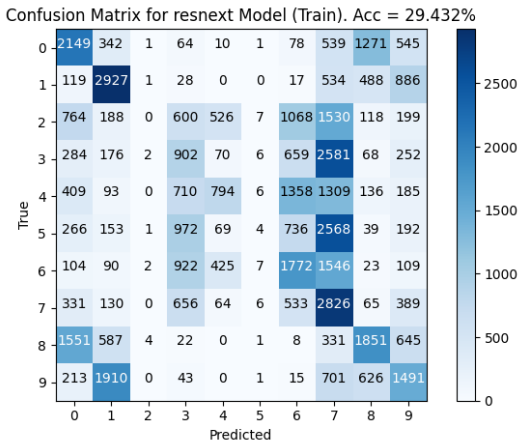

Fig. 51. ResNeXt Training Confusion Matrix

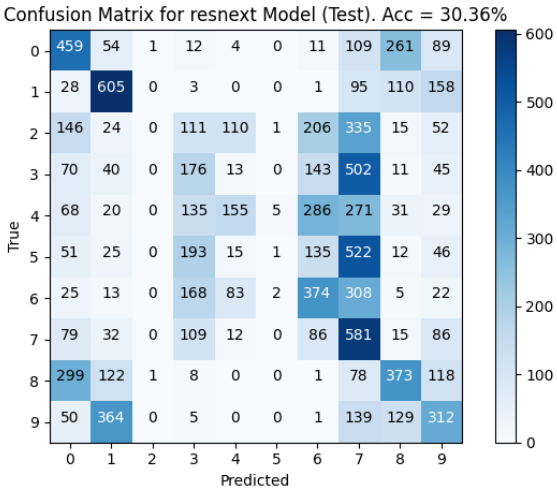

Fig. 52. ResNeXt Testing Confusion Matrix

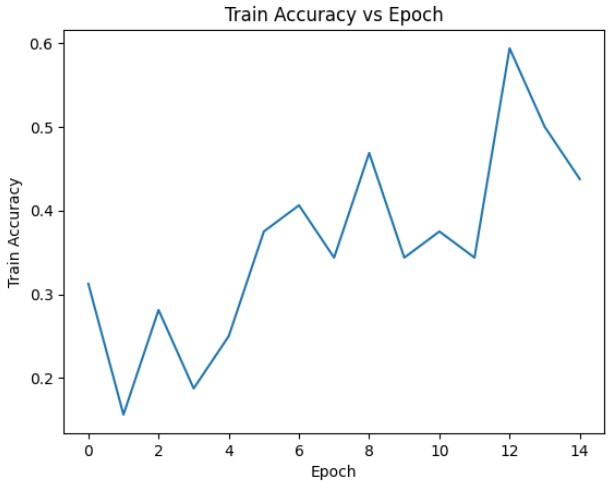

Fig. 53. DenseNet Training Accuracy

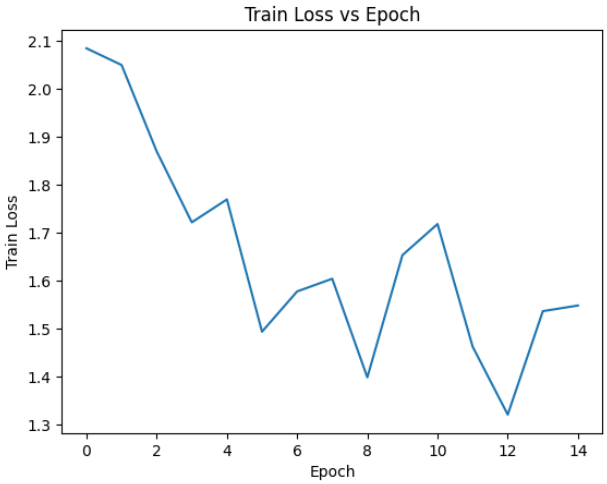

Fig. 54. DenseNet Training Loss

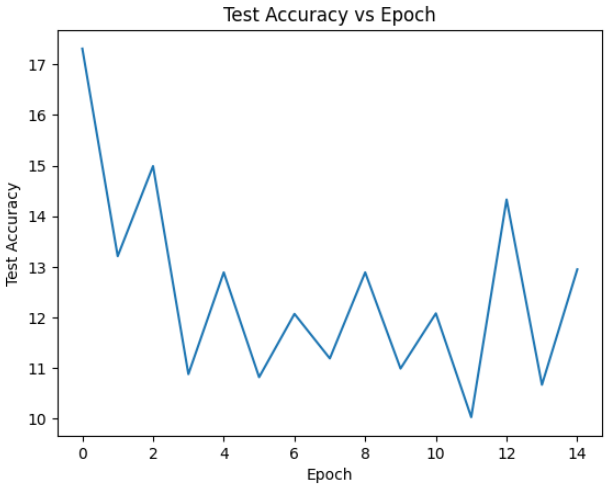

Fig. 55. DenseNet Training Accuracy

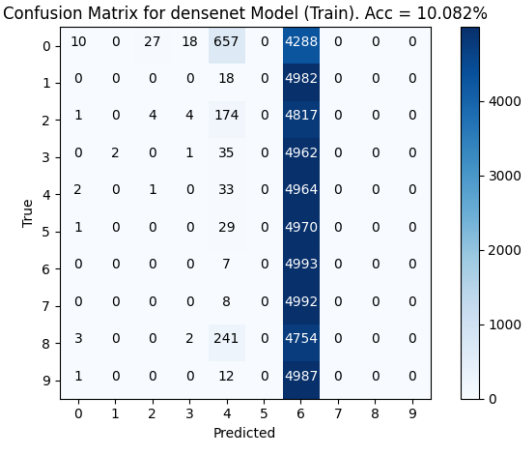

Fig. 56. DenseNet Training Confusion Matrix

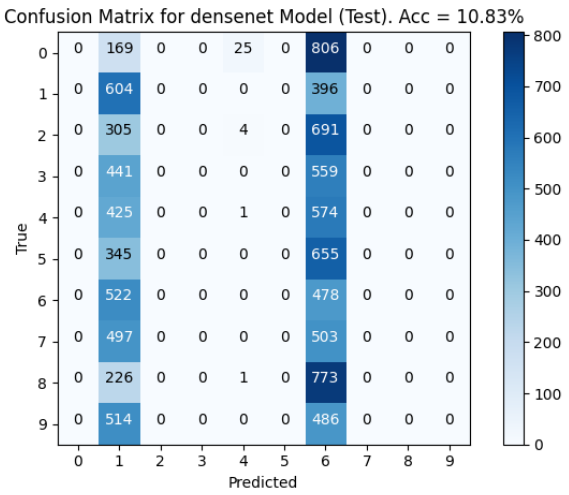

Fig. 57. DenseNet Testing Confusion Matrix