Computer Vision (RBE549) Homework 0

Simran Chauhan
MS Robotics Engineering
Worcester Polytechnic Institute
schauhan@wpi.edu

***Abstract*—This project investigates computer vision techniques across two distinct phases: traditional boundary detection methods and neural network optimization. Phase1 examines the creation and implementation of diverse filter banks for image processing. Phase2 focuses on enhancing neural network performance through advanced architectural modifications and optimization strategies.**

I. Phase 1: Shake My Boundary

The objective of this section is to implement a simplified version of the PB-lite boundary detection algorithm, where "Pb" stands for Probability of Boundary. The algorithm leverages texture, brightness, and color information at multiple scales to improve upon the Sobel and Canny baseline methods. The algorithm can be divided into four steps:

1) Generation of Filter bank
2) Generation of Texton, Brightness, and Color maps
3) Generation of Texton, Brightness, and Color gradient maps
4) Boundary detection using the maps, Sobel, and Canny baselines

A. Filter Bank Generation

The first step in the boundary detection algorithm is to create filter banks that capture texture information from input images. This implementation uses three types of filters: Oriented Derivative of Gaussian (DoG) filters, Leung-Malik filters, and Gabor filters.

1) Oriented Derivative of Gaussian (DoG) Filters: This filter bank consists of oriented Derivative of Gaussian (DoG) filters. These filters are created by convolving a Sobel filter with a Gaussian kernel and then rotating the result. The filter bank, generated with 4 scales and 16 orientations, is shown in Figure 1.

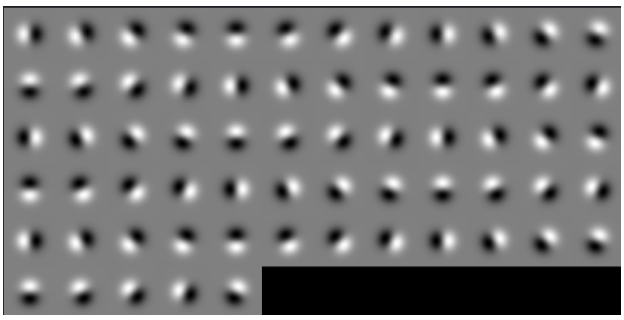


Fig. 1: Oriented Derivative of Gaussian (DoG) filters.

2) Leung-Malik (LM) Filters: The Leung-Malik filters are a set of multi-scale, multi-orientation filters, designed to capture texture and edge features with a total of 48 filters. This filter bank includes the first and second-order derivatives of Gaussians at 6 orientations and 3 scales, totaling 36 filters. Additionally, it contains 8 Laplacian of Gaussian (LOG) filters and 4 Gaussian filters.

The Leung-Malik Small filter bank is generated using the scales $\sigma = \{1, \sqrt{2}, 2, 2\sqrt{2}\}$, whereas the Leung-Malik Large filter bank is generated using the scales $\sigma = \{\sqrt{2}, 2, 2\sqrt{2}, 4\}$. The generated Leung-Malik Small filter bank is shown in Figure 2 and the generated Leung-Malik Large filter bank is shown in Figure 3.

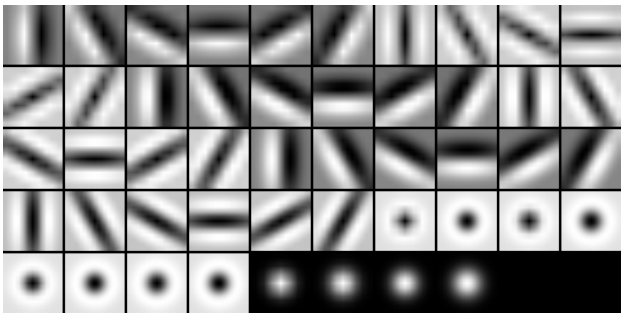


Fig. 2: Leung-Malik Small (LMS) filters.

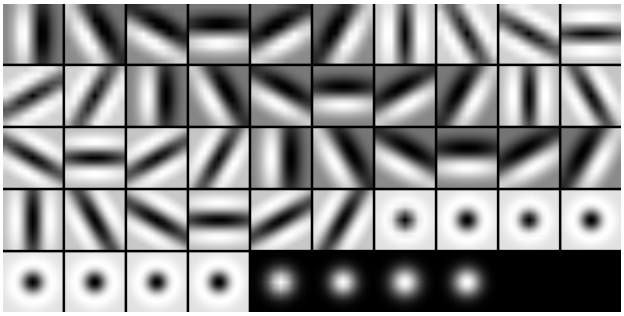


Fig. 3: Leung-Malik Large (LML) filters.

3) Gabor Filters.: Gabor filters are designed to mimic the filters found in the human visual system. A Gabor filter is formed by modulating a Gaussian kernel function with a sinusoidal plane wave, enabling it to capture both frequency and orientation information.The filter bank generated with 5 scales and 16 orientations is shown in Figure 4.

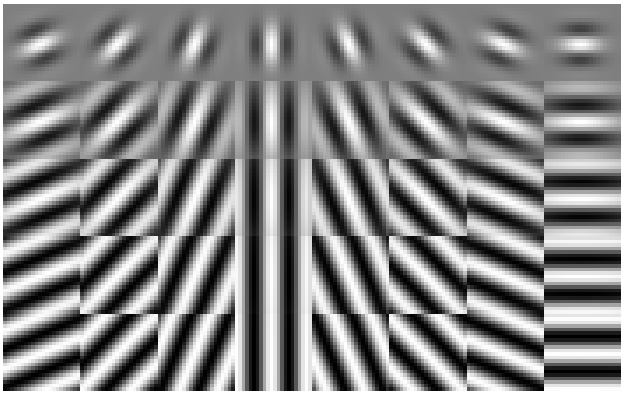


Fig. 4: Gabor Filters.

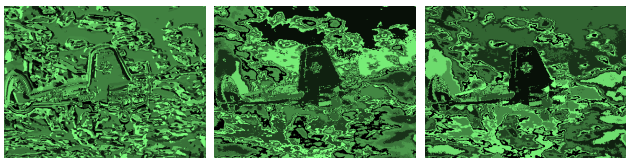


Fig. 5: T, B, and C for image 1.

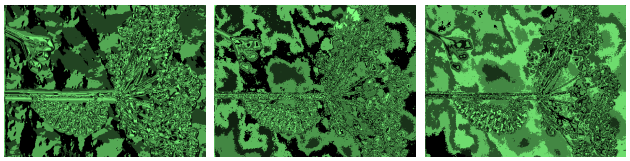


Fig. 6: T, B, and C for image 2.

B. Texton, Brightness, Color Map Computation

Textons are created by clustering filter responses to capture texture information. Brightness and color maps are generated by normalizing image intensities and transforming RGB values into feature spaces to extract significant boundaries.

For each input image, every filter in the bank is applied, producing an N-dimensional vector of filter responses centered at each pixel, where N is the number of filters. These response vectors are then clustered using K-means to assign each pixel a discrete texton ID, creating the Texton map (T). The output is a single-channel image with pixel values ranging from 1 to K, where $K = 64$ in this implementation.

The Brightness map (B) is generated by performing K-means clustering on the grayscale version of the input image. Similarly, the Color map (C) is created by applying K-means clustering to the three-channel color image. For both maps, $K = 16$. The resulting maps for all input images are shown in Figures 5 through 14.

C. Texton, Brightness and Color Gradient Map

The Texton gradient (T_g), Brightness gradient (B_g), and Color gradient (C_g) capture the changes in texture, brightness, and color distributions at each pixel. To calculate the gradients of the images, we use the half-disc masks shown in Figure 15. These masks are pairs of binary half-disc images. The goal of the masks is to compute the χ^2 distances by applying a simple filtering operation. This method makes it easier to calculate the χ^2 distances, which would normally involve aggregating

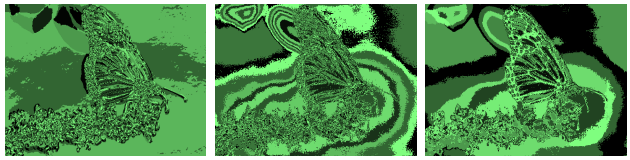


Fig. 7: T, B, and C for image 3.

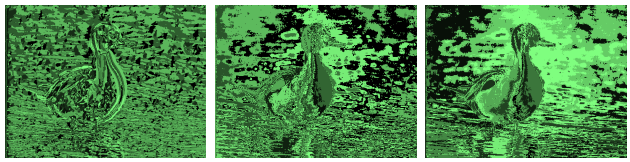


Fig. 8: T, B, and C for image 4.

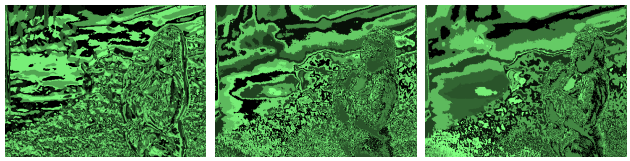


Fig. 9: T, B, and C for image 5.

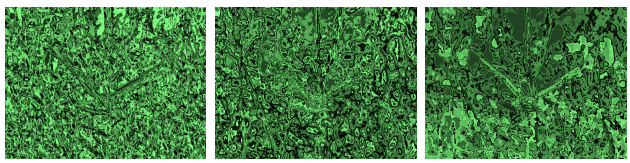


Fig. 10: T, B, and C for image 6.

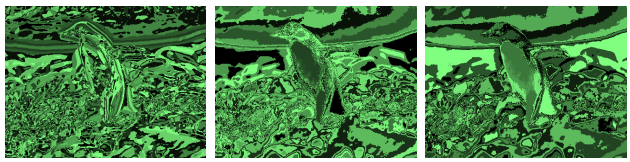


Fig. 11: T, B, and C for image 7.

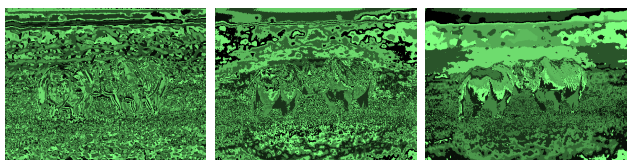


Fig. 12: T, B, and C for image 8.

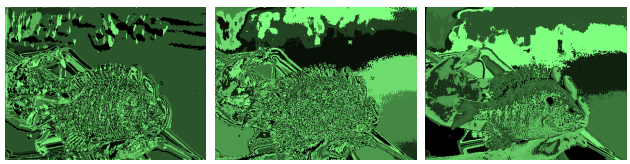


Fig. 13: T, B, and C for image 9.

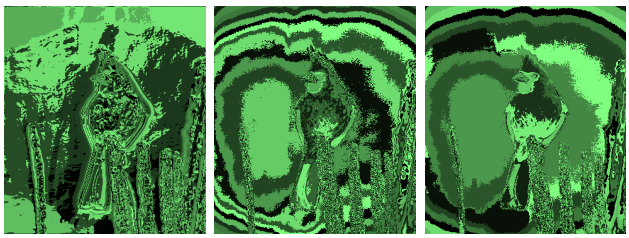


Fig. 14: T, B, and C for image 10.

histogram counts over the entire pixel neighborhoods. The half-disc masks generated with 4 scales and 8 orientations is shown in Figure 15.

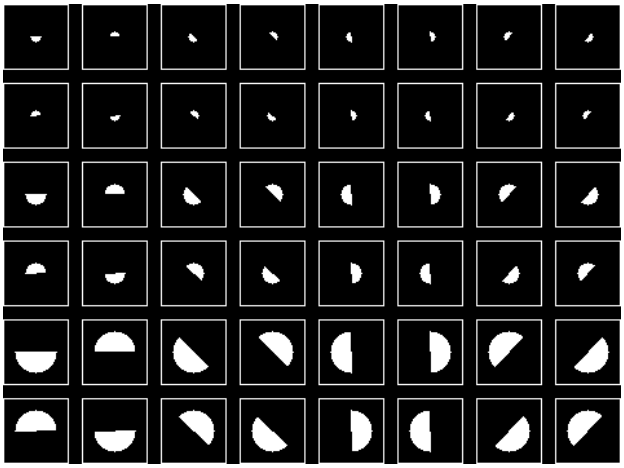


Fig. 15: Half Disks.

These gradients are computed by comparing the distributions between left and right half-disc pairs, which are generated by applying filters at the same scale in opposing directions. This method effectively captures the variations in texture, brightness, and color features, contributing to more precise boundary detection.The resulting gradients for all input images are shown in Figures 16 through 25.

D. Boundary Detection

The final step in this algorithm involves combining the Texton, Brightness, and Color gradients with the Sobel and Canny baselines to produce the final output using a below equation:

$$\text{PbEdges} = \frac{(Tg + Bg + Cg)}{3} \odot (w_1 \cdot \text{cannyPb} + w_2 \cdot \text{sobelPb})$$

In this equation, $\odot$ represents the Hadamard product operator. I choose the weights $w_1 = 0.5$ and $w_2 = 0.5$ in this algorithm. The generated pb-lite boundaries, along with their corresponding Sobel and Canny baselines, for all 10 provided images are shown in Figures 26 through 35.

E. Analysis

On comparing the pb-lite results with the Sobel and Canny baselines, it is evident that pb-lite effectively reduces many false positives present in the Canny baseline, while also capturing additional information that the Sobel baseline might overlook. Furthermore, the flexibility provided by pb-

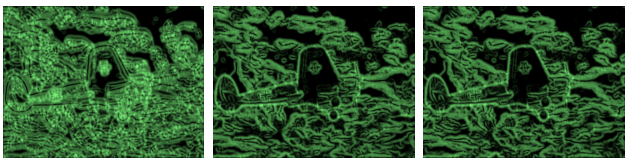


Fig. 16: (T_g), (B_g), (C_g) for image 1.

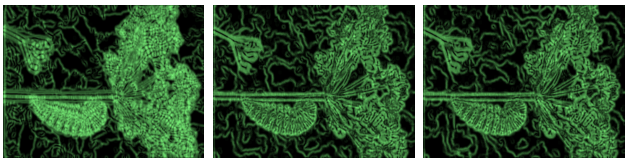


Fig. 17: (T_g), (B_g), (C_g) for image 2.

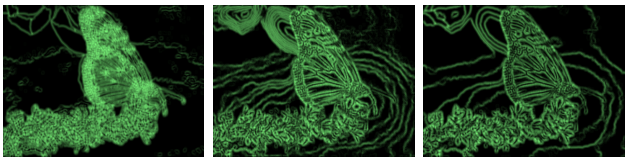


Fig. 18: (T_g), (B_g), (C_g) for image 3.

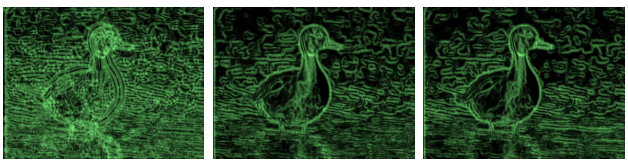


Fig. 19: (T_g), (B_g), (C_g) for image 4.

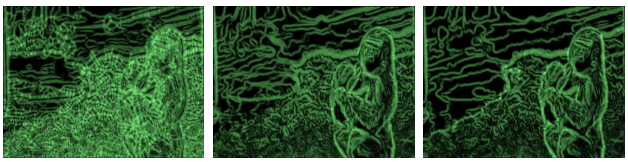


Fig. 20: (T_g), (B_g), (C_g) for image 5.

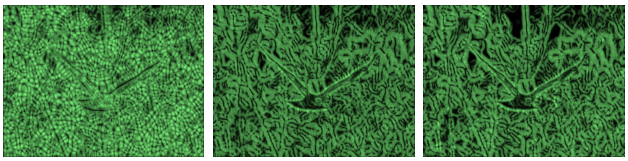


Fig. 21: (T_g), (B_g), (C_g) for image 6.

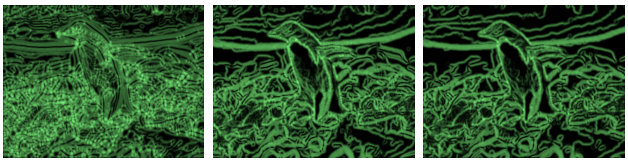


Fig. 22: (T_g), (B_g), (C_g) for image 7.

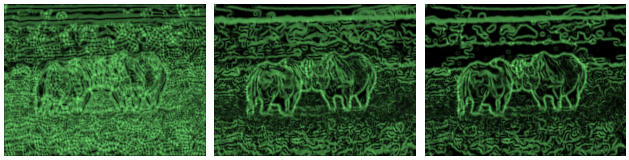


Fig. 23: (T_g), (B_g), (C_g) for image 8.

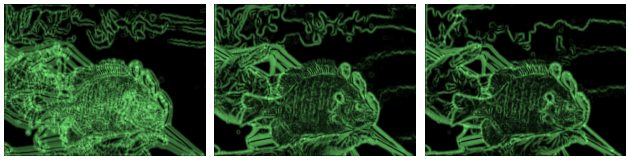


Fig. 24: (T_g), (B_g), (C_g) for image 9.

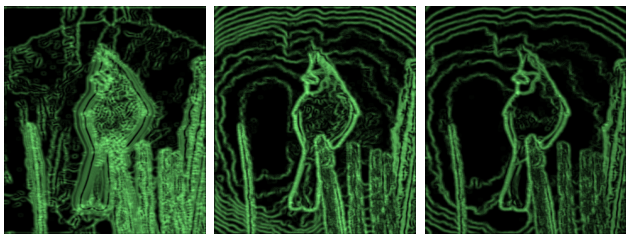


Fig. 25: (T_g), (B_g), (C_g) for image 10.

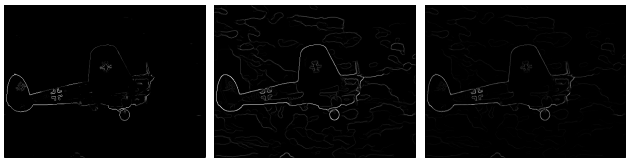


Fig. 26: Sobel, Canny, and pb-lite output of image 1.

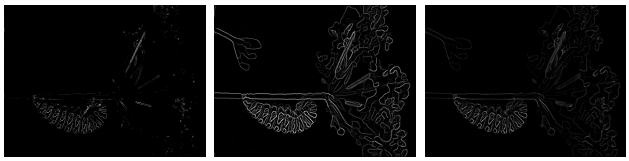


Fig. 27: Sobel, Canny, and pb-lite output for image 2.

Fig. 28: Sobel, Canny, and pb-lite output for image 3.

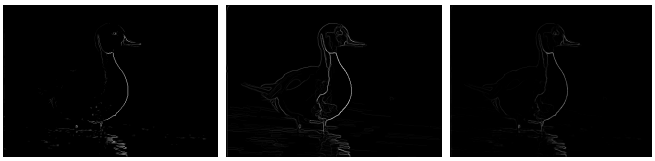


Fig. 29: Sobel, Canny, and pb-lite output for image 4.

Fig. 30: Sobel, Canny, and pb-lite output for image 5.

Fig. 31: Sobel, Canny, and pb-lite output for image 6.

Fig. 32: Sobel, Canny, and pb-lite output for image 7.

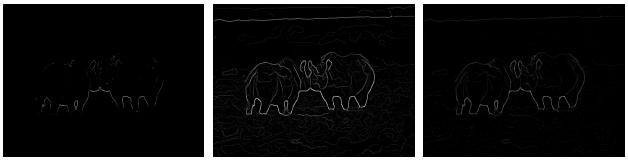


Fig. 33: Sobel, Canny, and pb-lite output for image 8.

Fig. 34: Sobel, Canny, and pb-lite output for image 9.

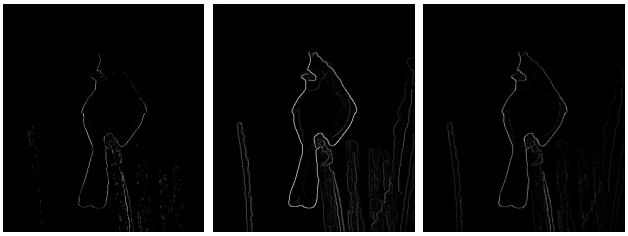


Fig. 35: Sobel, Canny, and pb-lite output for image 10.

lite—allowing the user to select filter banks with various scales and orientations—enables the boundary detection output to be fine-tuned according to specific user requirements. Therefore, it can be concluded that the pb-lite boundary detection method outperforms both Sobel and Canny algorithms.

II. Phase 2: Deep Dive on Deep Learning

In the second phase of this project, I aim to implement multiple neural network architectures to perform image clas-

sification on a randomized version of the CIFAR-10 dataset. This dataset comprises 50,000 training images and 10,000 testing images, each of size 32×32, spanning 10 distinct classes. I trained a total of five models for this study, which are as follows:

1) Basic Neural Network
2) Improved Neural Network
3) ResNet
4) ResNeXt
5) DenseNet

A. *My basic Neural Network*

In this section, I implemented a simple classification network, BasicNet, which includes two convolutional layers, each followed by a ReLU activation function. These activations are then passed through max-pooling layers to reduce the spatial dimensions. Finally, the output from the second max-pooling layer is flattened and passed through a series of fully connected layers. The architecture of this basic network is presented in Figure 36.

The model architecture contained $542,72$ parameters. Training was conducted using the AdamW optimizer with a learning rate of 0.01 and weight decay of 1×10^{-4}, using Cross-Entropy loss as the objective function. The data was processed in batches of 256 samples. Training ran for 50 epochs, taking approximately one hour to complete. The experimental results are shown in Figures 37 through 38.

The basic model achieved 67.10% accuracy on the training set but only 52.45% on the test set, indicating significant overfitting. The confusion matrices for both sets visualize this performance gap across different classes.

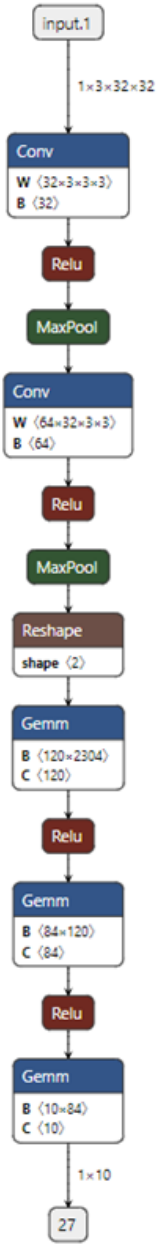


Fig. 36: Architecture of the Basic Neural Network

B. *My Improved Neural Network*

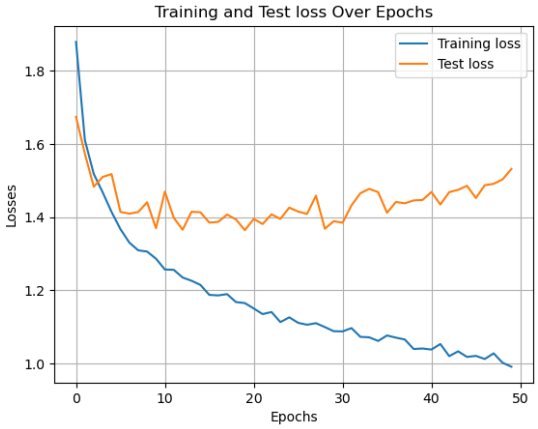
The analysis of the baseline model revealed two significant limitations: a test accuracy of 52.45% and a substantial training-testing performance gap, suggesting poor generalization. To address these constraints, this section presents architectural modifications and optimization strategies that enhance the model's classification capabilities.

1) Adding Batch Normalization: Batch normalization layers were added between the convolutional layers to stabilize the training process by normalizing the activations, reducing internal covariate shift, and improving the overall performance of the network.

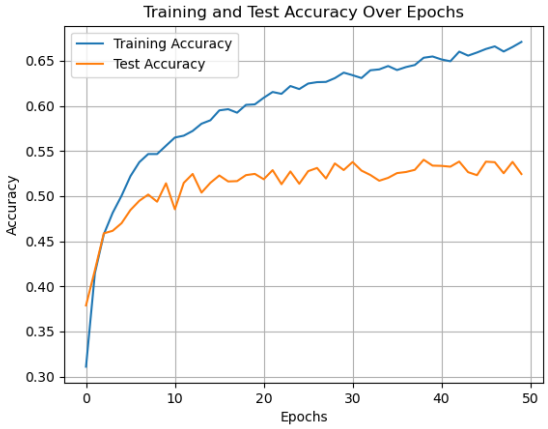
2) Data Augmentation and Standardization: The implemented data preprocessing pipeline consists of two main components: augmentation and standardization. For augmentation, each input image is first padded with 4 pixels on all sides, then subjected to random horizontal flipping with 50% probability, followed by random cropping to restore the original dimensions of 32×32 pixels. These transformations effectively increase the diversity of the training data while preserving the semantic content of the images. To standardize the input distribution, each image is normalized using the CIFAR-10 dataset's mean and standard deviation values. This normalization step ensures consistent feature scaling and improves training stability and model convergence.

These preprocessing techniques were uniformly applied to all advanced architectures (ResNet, ResNeXt, and DenseNet) to maintain experimental consistency.

3) Learning Rate Decay: To further enhance model performance, I implemented learning rate decay using the StepLR scheduler. Starting with an initial rate of 0.01 and is reduced by a factor of 0.1 every 15 epochs. This gradual decay schedule enables larger optimization steps during initial epochs while

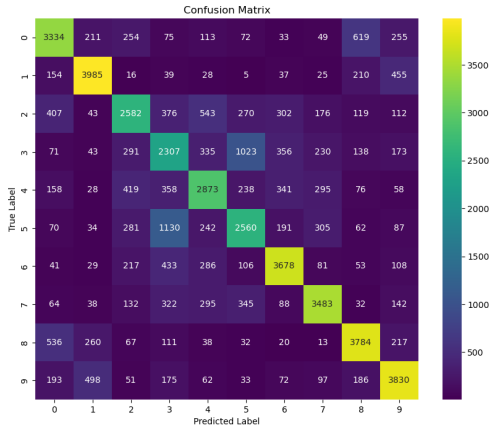


(a) Training and Testing Loss

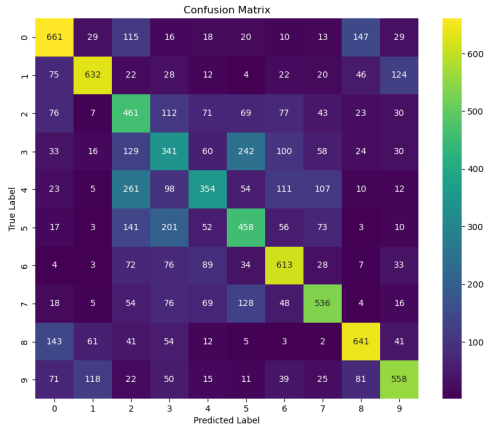


(b) Training and Testing Accuracy

Fig. 37: Performance metrics of the Basic Neural Network: (a) Training and testing loss, (b) Training and testing accuracy



(a) Training Confusion Matrix



(b) Testing Confusion Matrix

Fig. 38: Confusion matrices of the Basic Neural Network: (a) Training data, (b) Testing data

allowing finer adjustments as training progresses, preventing overshooting of the optimal solution and improving overall model convergence.

The architecture of this network is shown in Figure 39. The AdamW optimizer was used with a batch size of 256, and the model was found to have 522,238 parameters. The training process lasted for approximately 3 hours, achieving an overall accuracy of 78.58% on the test data and 78.84% on the training data. After applying all the optimization strategies described above, the model demonstrated notable improvements and outperformed the baseline model.The experimental results are shown in Figures 40 through 41.

C. ResNet -14

In this section, I present the implementation of a Residual Network (ResNet). The ResNet-14 is a compact network architecture with 14 layers, specifically designed for efficient image classification tasks. The architecture consists of three stages, each progressively increasing the number of channels. Residual connections are employed throughout the network to enhance gradient flow and mitigate the vanishing gradient problem.

The proposed architecture follows the structure described below. The first layer consists of 3×3 convolutions. The network then uses a stack of 12 layers with 3×3 convolutions applied to feature maps of sizes $\{32, 16, 8\}$,, respectively, with four layers for each feature map size. The numbers of filters for these layers are $\{16, 32, 64\}$, respectively. The subsampling is achieved through convolutions with a stride of 2, reducing the spatial dimensions. Finally, the network ends with global average pooling, followed by a 10-way fully connected layer. There are 14 total stacked weighted layers.

When shortcut connections are employed, they are linked to the pairs of 3×3 convolutions. Each residual block consists of two 3×3 convolutions, followed by batch normalization and ReLU activation. The network incorporates skip connections

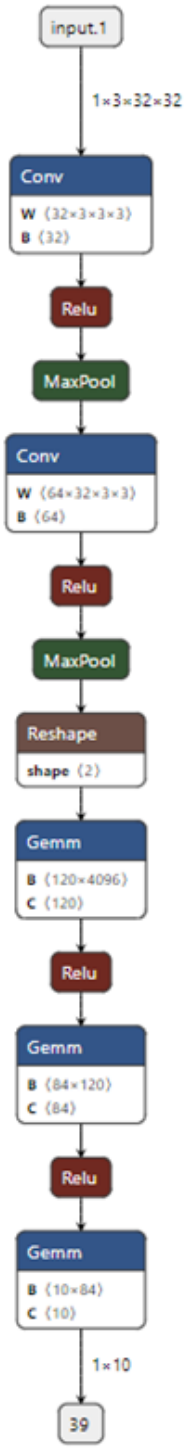
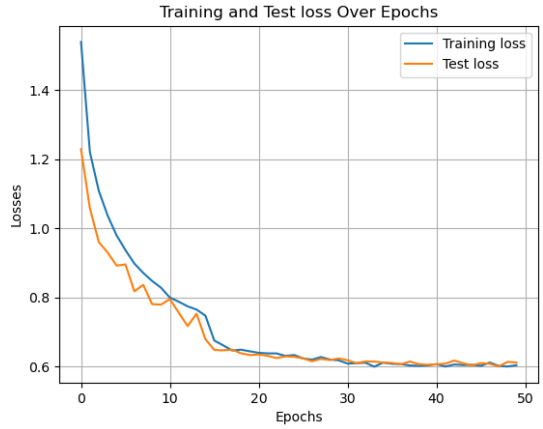
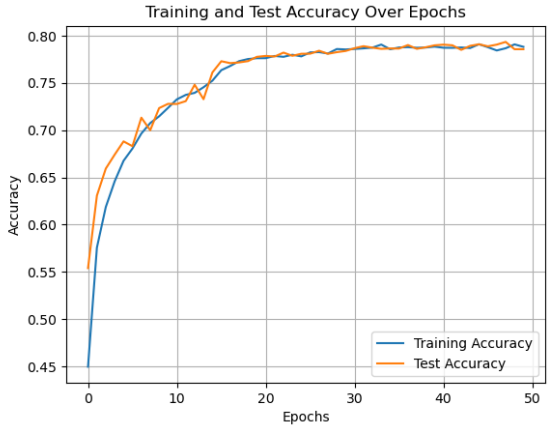


Fig. 39: Architecture of my Improved Neural Network



(a) Training and Testing Loss



(b) Training and Testing Accuracy

Fig. 40: : (a) Training and Testing Loss of Basic Network (b)Training and Testing Accuracy

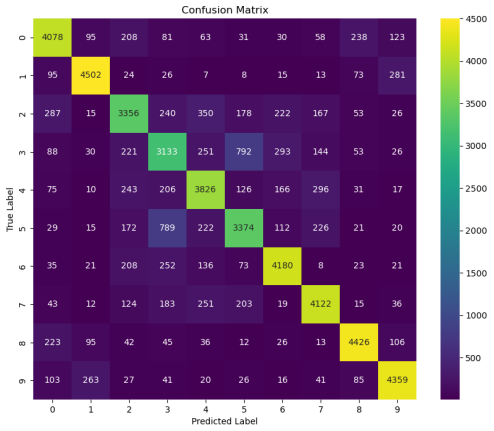
throughout, with projection shortcuts at the stage transitions. The architecture of the network is shown in Figure 42.

The Stochastic Gradient Descent (SGD) optimizer was used with a batch size of 256, and the model was found to have 52,897 parameters.Total number of epochs is 50. The learning rate is set to 0.01, and momentum is set to 0.9. Learning rate decay is implemented using step decay every 15 epochs, with a decay factor $\gamma = 0.1$. Weight decay is applied at 1×10^{-4} for regularization. For the ResNet - 14, the aforementioned data augmentation techniques were applied.The time taken to train this network is about 3 hour.

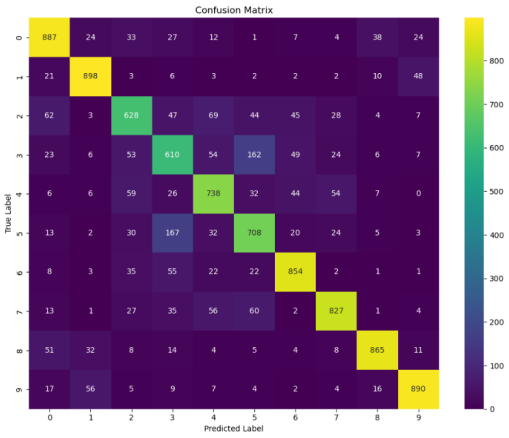
The model achieves an overall accuracy of 84.18% on the testing data and 87.74% on the training data. The experimental results are shown in Figures 43 through 44.

D. ResNeXt-14

In this section, I present the implementation of a ResNeXt network, which extends the ResNet architecture by introducing grouped convolutions with a cardinality factor of 8. This



(a) Training Confusion Matrix



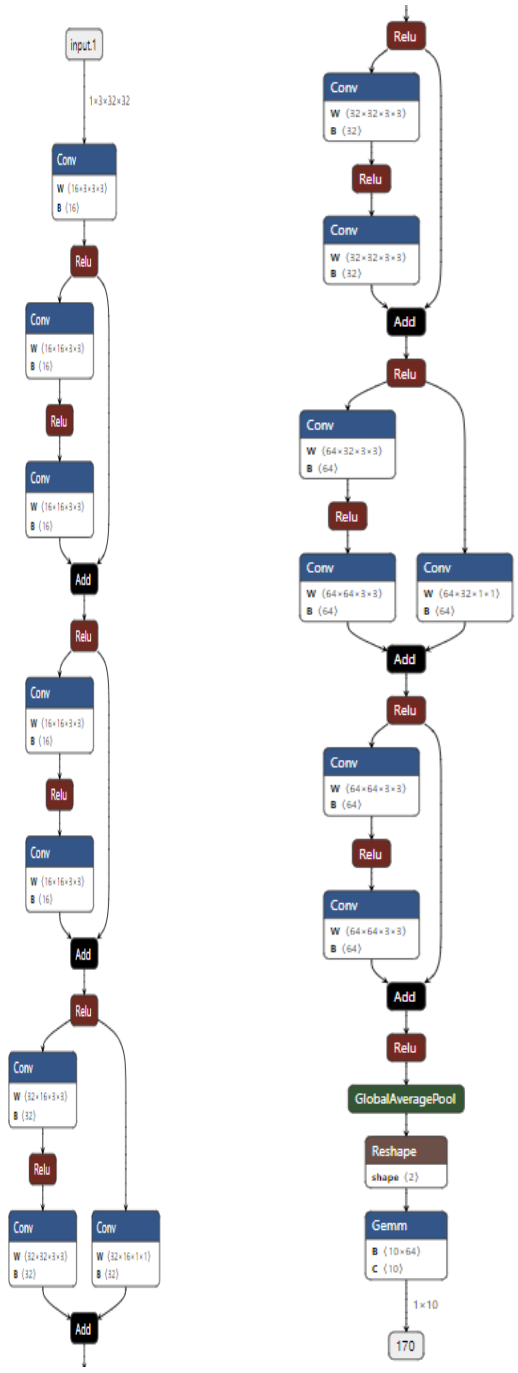
(b) Testing Confusion Matrix

Fig. 41: Train and Test Confusion Matrix of Improved Basic model.

network consists of multiple stages, progressively increasing the number of channels ($16 \rightarrow 32 \rightarrow 64$), and uses both regular residual connections and grouped convolutions to enhance feature learning and gradient flow.

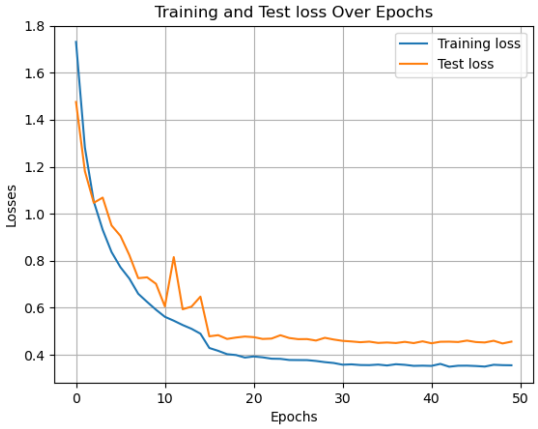
The proposed architecture begins with a 3×3 convolution layer as shown in Figure 45. The network includes two residual blocks, each employing 1×1 grouped convolutions for dimensionality reduction and 3×3 grouped convolutions for feature extraction, maintaining 16 channels throughout. The subsampling is achieved through convolutions with a stride of 2, increasing the number of features from 16 to 32 channels. This is followed by two additional residual blocks that apply grouped convolutions and projection shortcuts to align feature dimensions.

A subsequent downsampling operation further doubles the number of features to 64 channels, with two more residual blocks. The network concludes with global average pooling to reduce spatial dimensions, followed by a fully connected layer.

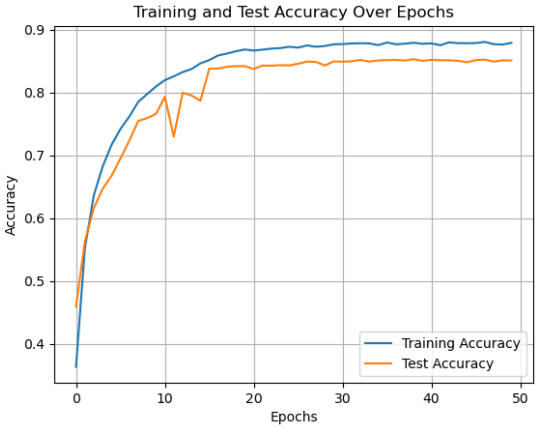


(a) Part 1 (b) Part 2

Fig. 42: Architecture of ResNet-14 Neural Network

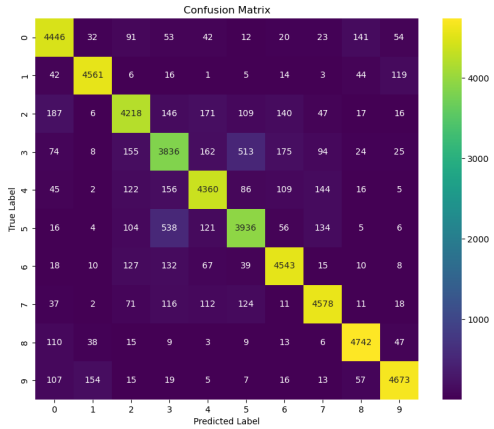


(a) Training and Testing Loss

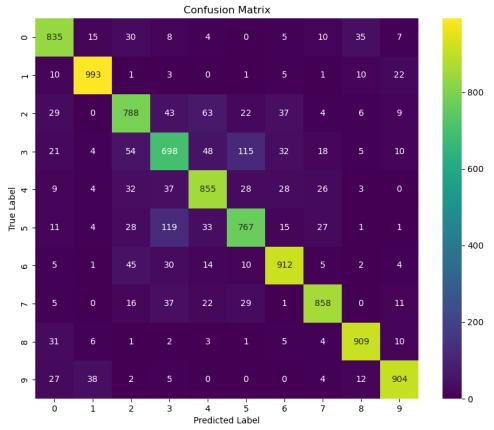


(b) Training and Testing Accuracy

Fig. 43: Training and Testing Performance of ResNet-14 Network: (a) Loss curves (b) Accuracy curves



(a) Training Confusion Matrix



(b) Testing Confusion Matrix

Fig. 44: Training and Testing Confusion Matrices of ResNet-14 Network

This architecture effectively combines grouped convolutions and residual connections to achieve superior feature extraction and robust gradient propagation.

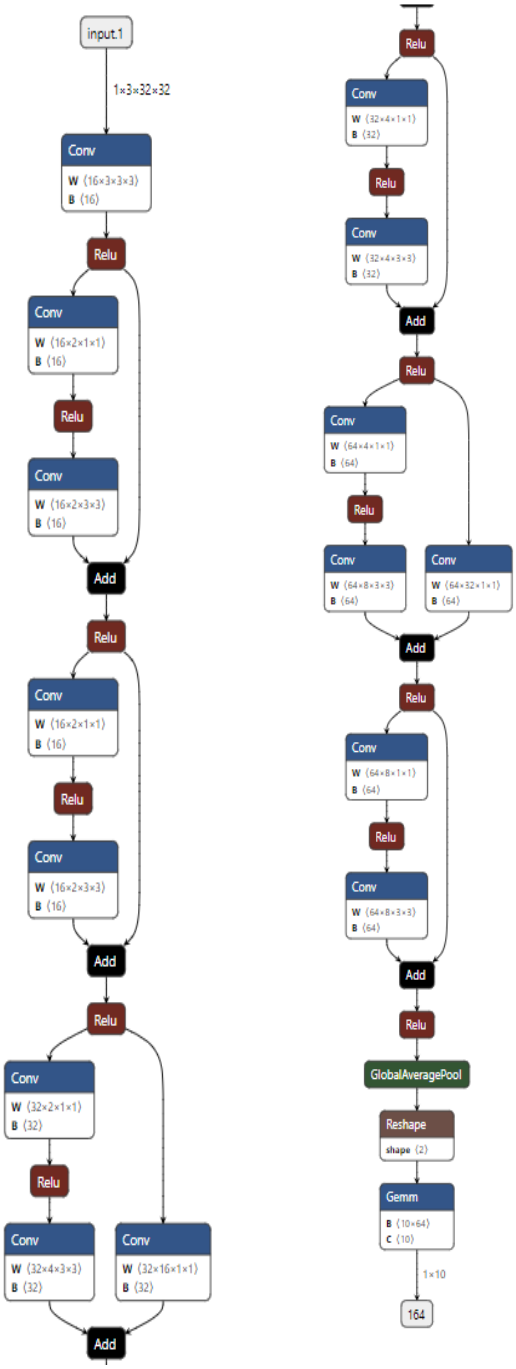
The Stochastic Gradient Descent (SGD) optimizer was used with a batch size of 256, and the model was found to have 56,879 parameters. The total number of epochs is 50. The learning rate is set to 0.01, and momentum is set to 0.9. Learning rate decay is implemented using step decay every 15 epochs, with a decay factor $\gamma = 0.1$. Weight decay is applied at 1×10^{-4} for regularization. For the ResNeXt-14, the aforementioned data augmentation techniques were applied. The time taken to train this network is about 3.2 hours.

The model achieves an overall accuracy of 66.67% on the testing data and 67.17% on the training data. The experimental results are shown in Figures 46 through 47.

E. DenseNet

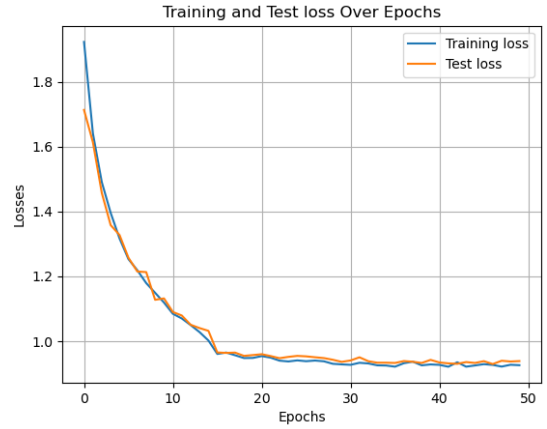
In this section, I present the implementation of a DenseNet-inspired network, designed for efficient image classification on the CIFAR-10 dataset. This network leverages densely connected convolutional layers, where each layer is directly connected to all subsequent layers within its block. The model progressively increases the number of features ($16 \rightarrow 32 \rightarrow 64$), promoting feature reuse and reducing the number of parameters compared to traditional convolutional architectures.

The proposed architecture begins with a 3×3 convolution layer with 16 output channels as shown in Figure 48. The network includes a dense block, composed of four layers, where each layer adds 12 new features (growth rate: 12) through 3×3 convolutions. After the dense block, a transition layer reduces the number of features by half using a 1×1 convolution and 2×2 average pooling. Another dense block follows, with four more layers adding features, and a second transition layer further reduces the number of features by half. The final dense block adds additional features before the network concludes with global average pooling to reduce

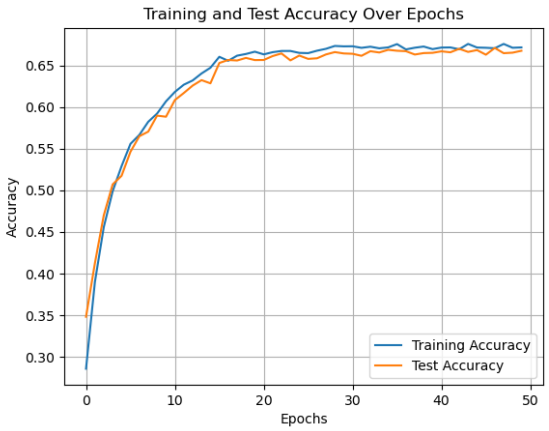


(a) Part 1 (b) Part 2

Fig. 45: Architecture of ResNeXt-14 Neural Network



(a) Training and Testing Loss



(b) Training and Testing Accuracy

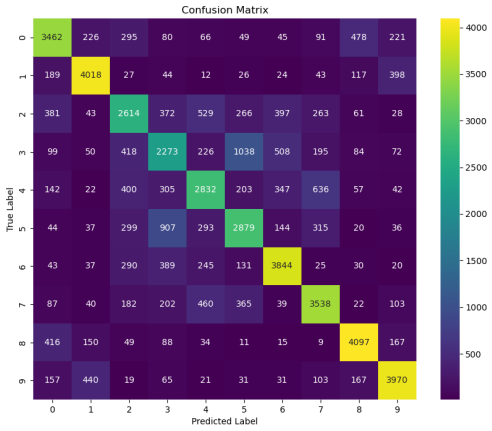
Fig. 46: Training and Testing Performance of ResNeXt-14 Network: (a) Loss curves (b) Accuracy curves

spatial dimensions, and a fully connected layer.

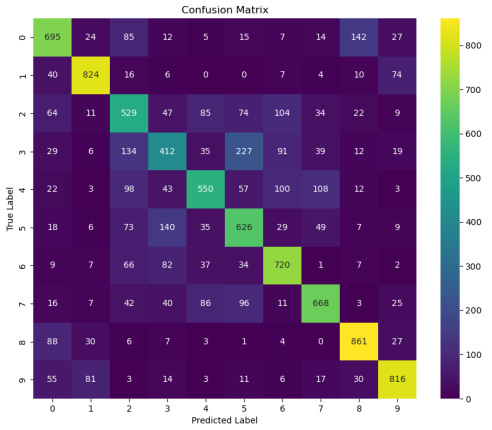
Throughout the architecture, batch normalization and ReLU activation are applied to stabilize training and introduce non-linearity, allowing the network to learn complex features efficiently. This architecture effectively leverages dense connections and transition layers to achieve efficient feature propagation, reuse, and robust gradient flow, making it well-suited for image classification tasks.

The Stochastic Gradient Descent (SGD) optimizer was used with a batch size of 256, and the model was found to have 69,778 parameters. The learning rate is set to 0.01, and momentum is set to 0.9. Learning rate decay is implemented using step decay every 15 epochs, with a decay factor $\gamma = 0.1$. Weight decay is applied at 1×10^{-4} for regularization. For the DenseNet, the aforementioned data augmentation techniques were applied. The time taken to train this network is about 4.5 hours.

The model achieves an overall accuracy of 83.07% on the testing data and 85.36% on the training data. The experimental



(a) Training Confusion Matrix



(b) Testing Confusion Matrix

Fig. 47: Training and Testing Confusion Matrices of ResNeXt-14 Network

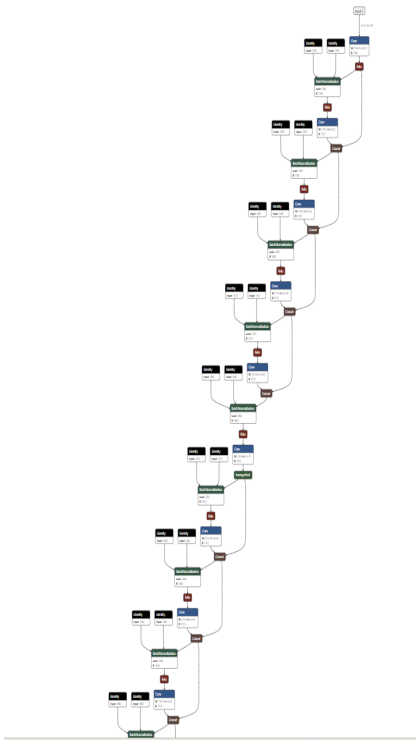
TABLE I: Performance Comparison of Different Models

Model	Train accuracy	Test accuracy	Inference Time(avg)
BasicNet	67.10%	52.45%	1ms
BatchNormNet	78.84%	78.58%	1.25ms
ResNet	87.74%	84.18%	3ms
ResNeXt	67.16%	66.76%	5.2ms
DenseNet	85.36%	83.07%	9.8ms

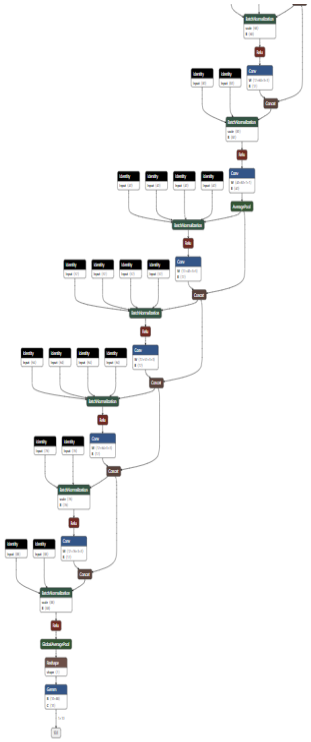
results are shown in Figures 49 through 50.

III. Performance Comparison

The table I summarizes the performance of all implemented models. The inference time of each model is included in this comparative analysis. Note that the time refers to average inference time per image.

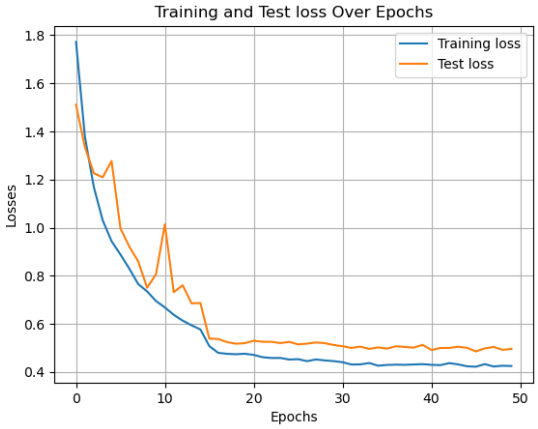


(a) Part 1

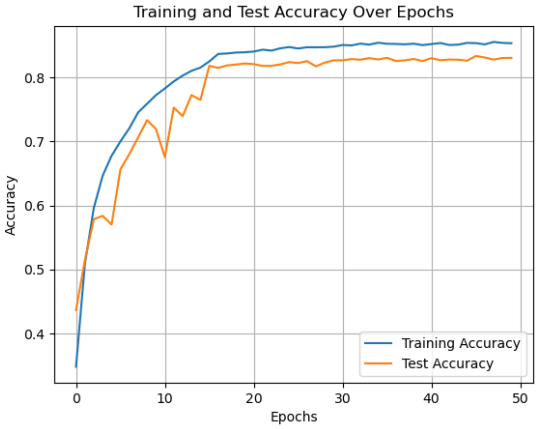


(b) Part 2

Fig. 48: Architecture of DenseNet Neural Network



(a) Training and Testing Loss

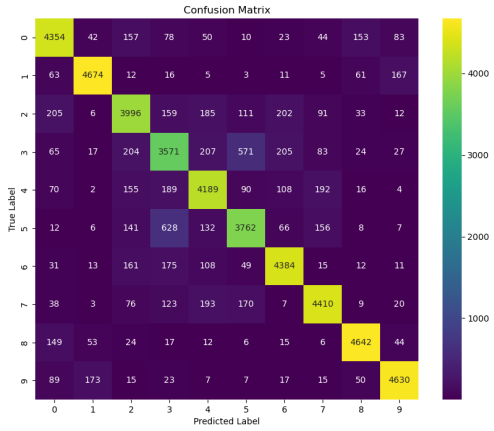


(b) Training and Testing Accuracy

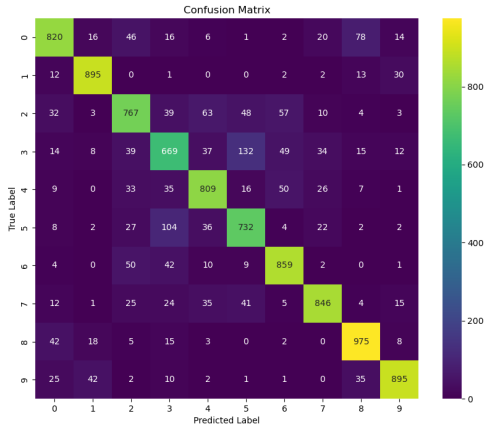
Fig. 49: Training and Testing Performance of DenseNet Network: (a) Loss curves (b) Accuracy curves

IV. Conclusion

From the comparative analysis, it is clear that ResNet and DenseNet outperform BasicNet and ResNeXt in terms of testing accuracy. However, the performance of ResNeXt could be improved by further tuning the hyperparameters. Future work will focus on optimizing these models to achieve better accuracy and efficiency for specific tasks.



(a) Training Confusion Matrix



(b) Testing Confusion Matrix

Fig. 50: Training and Testing Confusion Matrices of DenseNet Network