Homework 0: Alohomora

Kushagra Srivastava
Email: ksrivastava1@wpi.edu
Using 1 Late Day

***Abstract*—This report comprises of 2 sections: edge detection and neural network implementations. Section 1 of the report describes the implementation of a simplified probability of boundary (pb) algorithm that leverages filter banks to enhance edge detection outputs of classical algorithms (canny and sobel). Section 2 of this report describes the implementation of 4 neural network architectures: CNN, ResNet, ResNeXT, and DenseNet for classifying images in the CIFAR-10 dataset. A thorough analysis of the implementation and training details for these networks is provided.**

I. Edge Detection: Shake My Boundary

This section describes the implementation of a simplified version of the pb algorithm [1] by leveraging texture, brightness, and color information encoded in an image. Figure 1 describes the pb algorithm. This algorithm can be summarized in 4 steps.

1) Generation of 4 Filter banks: (i) Orientated Difference of Gaussian (DoG) filters, Leung-Malik (LM) filters, and Gabor filters. (Section I-A)
2) Computing texture, brightness, and color maps. (Section I-B)
3) Using the aforementioned maps to compute the texture, brightness, and color gradients of each pixel in an image. (Section I-C)
4) Combining the gradient information with sobel and canny edge detection outputs to obtain the final result. (Section I-D)

A. Filter Bank Generation

1) Orientation DoG filters: A DoG filter is generated by convolving a Gaussian kernel with the Sobel Filter (see Equation 1). The generated DoG filter was rotated across 16 directions, each direction obtained through uniform partitioning of the angular range between 0 and 360 degrees. The

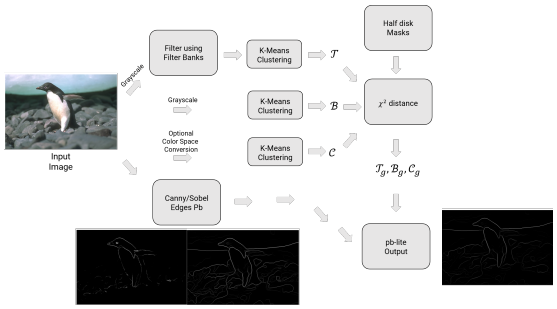


Fig. 1: Simplified implementation of Probability of boundary (pb)

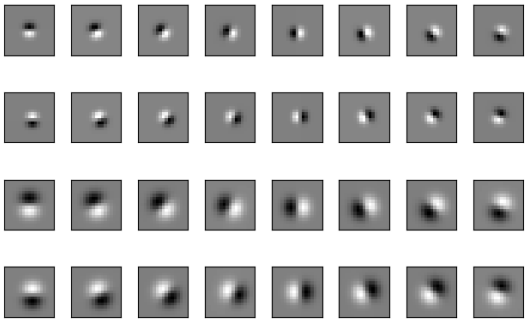


Fig. 2: Kernel size was 17 while scales for the Gaussian kernel were 1 and 2

Gaussian kernel was generated for two scales. Thus, the total count of the DoG filters was 32. Refer to Figure 2 for more details.

$$S = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{1}$$

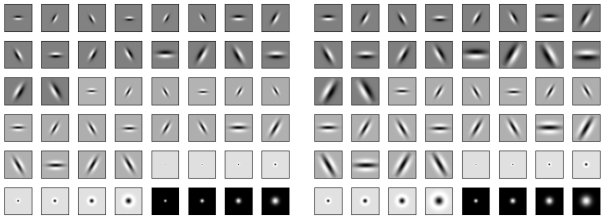
2) LM filters: LM filters consist of first and second-order derivatives of Gaussian (at 6 orientations and 3 scales), 8 Laplacian of Gaussian (LOG) filters, and 4 Gaussian filters (total 48 filters). Derivatives of Gaussian filters are obtained by convolving a Gaussian kernel with the sobel filter (see Equation 1) k times, where k represents the order of the Gaussian derivative. These filters are generated at an elongation factor of 3 ($\sigma_x = \sigma$ and $\sigma_y = 3\sigma$. LOG filters are obtained by convolving a Gaussian filter with the Laplacian filter given in Equation 2. Depending on the scales used, the LM filters are further categorized into LM small (LML) and LM large (LML). For further implementation details, refer to figure 3

$$L = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{2}$$

3) Gabor Filters: A Gabor filter is a Gaussian kernel function modulated by a sinusoidal plane wave[1]. The Gabor filter bank is shown in Figure 4

B. Texture, Brightness, and Color Maps

[1]For more details please refer to https://en.wikipedia.org/wiki/Gabor_filter



LMS filters LML filters

Fig. 3: Scales used for LMS = $\begin{bmatrix}1 & \sqrt{2} & 2 & 2\sqrt{2}\end{bmatrix}$ and LML = $\begin{bmatrix}\sqrt{2} & 2 & 2\sqrt{2} & 4\end{bmatrix}$. The derivatives of the Gaussian filters were generated for the first three scales (with $\sigma_x = \sigma$ and $\sigma_y = 3\sigma$) while LOG filters were generated for all scales (σ) and 3 times the scales (3σ). The Gaussian kernels were generated at all the scales.

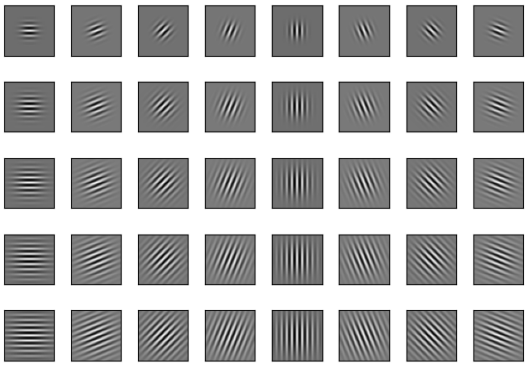


Fig. 4: Scales used for the Gabor filter bank were 5, 7, 9, 11, and 13. λ was set to 5, ψ was set to 0, and γ was 1. The obtained filter was rotated between 0 and π at an interval of $\pi/8$. A total of 40 filters were generated.

1) Texture Maps: After generating the filter banks, each of these filters is applied to the images. This will form a vector of filter responses for each pixel in the image. These responses capture the texture properties of a small neighborhood around the pixel. The next step is to cluster pixels as per the filter responses. For this operation, KMeans clustering is used which replaces the N-dimensional (N being the total number of filters) vector to a single value belonging to the set $\{1, 2, 3, \ldots, K\}$. For texture maps, K was chosen to be 64.

2) Brightness and Color Maps: A similar idea is followed to cluster the pixels as per their brightness and RGB color values. For brightness maps, the grayscale equivalent of the image while for color maps, the RGB image is used directly. For both maps, K was set to 16. The texture, brightness, and color maps are illustrated in figure 5.

C. Texture, Brightness, and Color Gradients

To understand the change of texture, color, and brightness in a neighborhood around a pixel, the corresponding gradients are calculated. To find these gradients for each of the maps

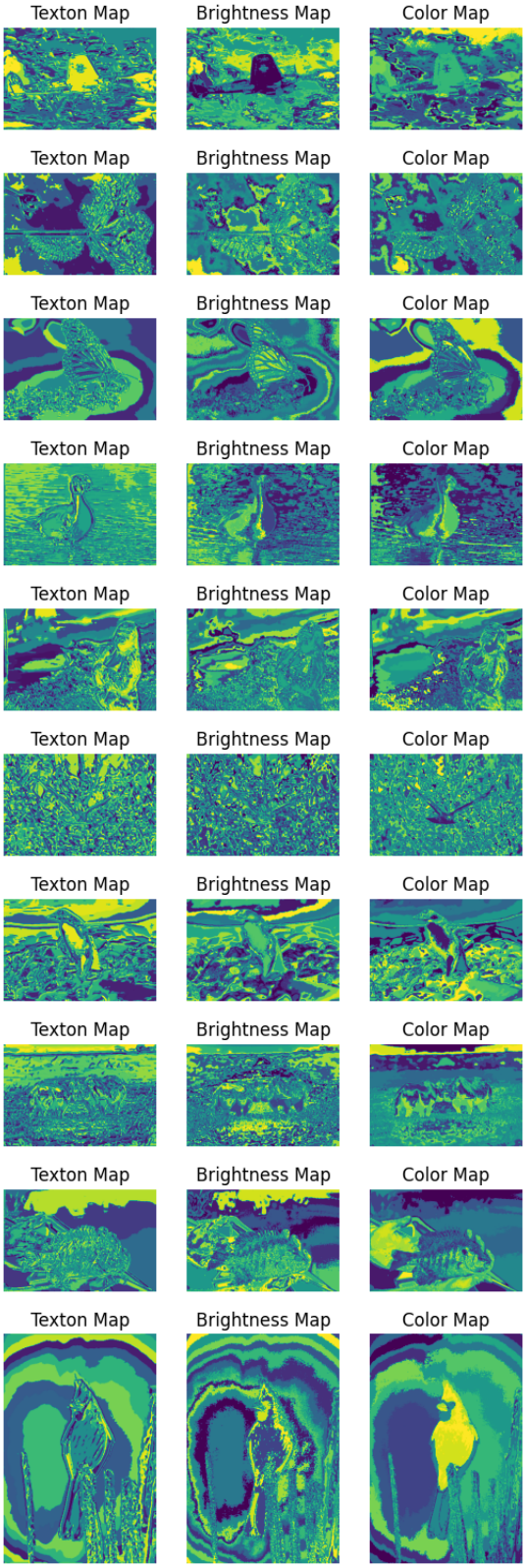


Fig. 5: Texture, brightness, and color maps

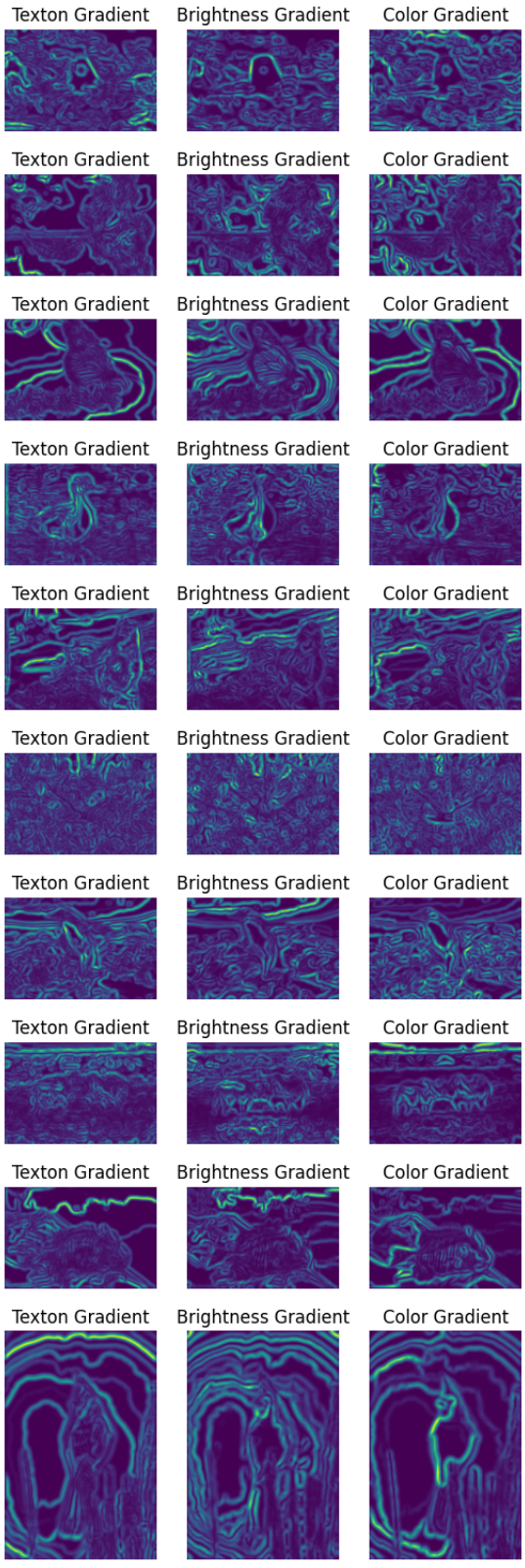


Fig. 6: Texture, brightness, and color gradients

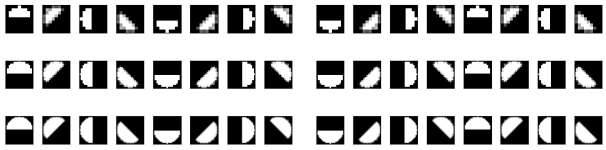


Fig. 7: Left and right half-disc masks generated for 3 radii values and 8 orientations.

discussed above, binary half-disc masks in pairs (left and right) are used. See figure 7 for an illustration. These masks calculate the $\mathcal{X}^2$ (chi-square) distances through filtering. Computing $\mathcal{X}^2$ is much faster than looping over each pixel neighborhood and aggregating counts for histograms. If the distributions obtained by convolving the images with the left and right disc masks are similar, then the gradient is small while for dissimilar distributions, the gradients will be large. The different scales and angles using which the masks were created, encodes how quickly the distributions change at the corresponding scales and angles. Mathematically, $\mathcal{X}^2$ can be calculated using equation 3 where g and h are histograms with the same binning scheme and K is the number of bins. Figure 6 shows an illustration of the gradients obtained.

$$\mathcal{X}^2(g,h) = \frac{1}{2}\sum_{i=1}^{K}\frac{(g_i - h_i)^2}{g_i + h_i} \tag{3}$$

D. Pb Output Generation

For calculating the final output using the texture ($\mathcal{T}_g$), brightness($\mathcal{B}_g$), color($\mathcal{C}_g$) gradients and the corresponding canny (C) and sobel (C) outputs, equation 4 is used. The weights were determined empirically. Results are illustrated in 8.

$$O = \frac{\mathcal{T}_g + \mathcal{B}_g + \mathcal{C}_G}{3} \times (0.1S + 0.9C) \tag{4}$$

E. Analysis

The simplified pb algorithm was able to reject the false positives detected by the canny edge detection algorithm while it also rejected some useful information from the sobel algorithm. Since the choice, orientation, size, scales, and number of filter banks are hyperparameters, better results can be achieved by tuning them. This flexibility of the pb algorithm gives it an edge over the canny and sobel methods.

II. Neural network Implementation

this section reports the implementation details of various architectures that were eventually trained to classify CIFAR-10 dataset images. In particular, the following architectures were implemented. The final layer of each neural network consists of 10 neurons, and it is followed by a softmax activation function. The softmax function assigns probabilities to each of the 10 classes, representing the likelihood of the input belonging to each class.

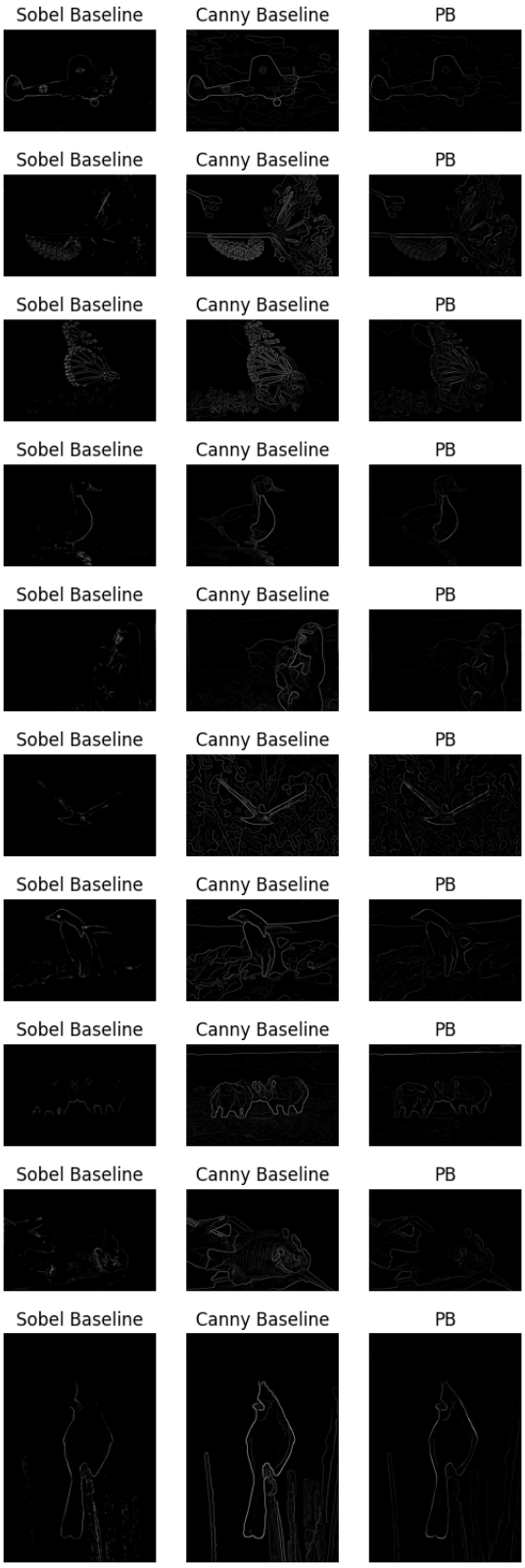


Fig. 8: Sobel, Canny, and PB outputs

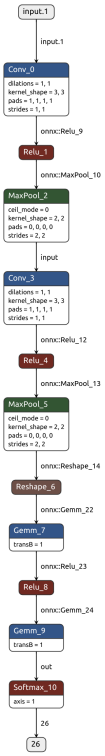


Fig. 9: Network Architecture for Basic CNN

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

A. Basic Convolutional Neural Network

The basic network comprises a Convolutional-ReLU-Convolutional-ReLU layer followed by 2 fully-connected layers. Adam optimizer, learning rate of 0.0001, and a batch size of 64 were used and the network was trained for 10 epochs. Please refer to Fig 9 for architecture, Fig 10 for testing confusion matrix, Fig 11 training confusion matrix, Fig 12 for training loss, Fig 13 for training accuracy, Fig 14 for testing loss, and Fig 15 for testing accuracy.

B. Convolutional Neural Network

This network comprises two sequential layers followed by 2 fully-connected layers. The sequential layer comprises a convolutional layer, followed by a batch normalization layer, and a ReLU activation function layer. Adam optimizer, learning rate of 0.0001, and a batch size of 64 were used and the network was trained for 10 epochs. Please refer to Fig 16 for architecture, Fig 17 for testing confusion matrix, Fig 18 training confusion matrix, Fig 19 for training loss, Fig 20 for accuracy, Fig 21 for testing loss, and Fig 22 for testing accuracy.

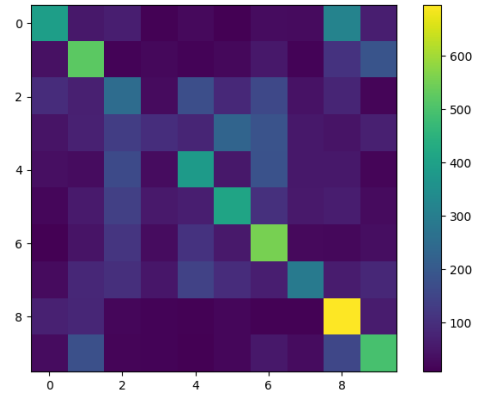


Fig. 10: Confusion Matrix for Basic CNN

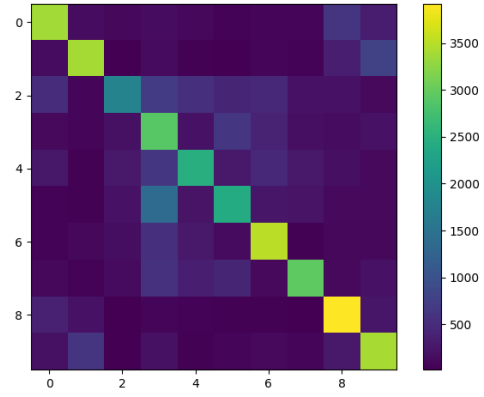


Fig. 11: Training Confusion Matrix for Basic CNN

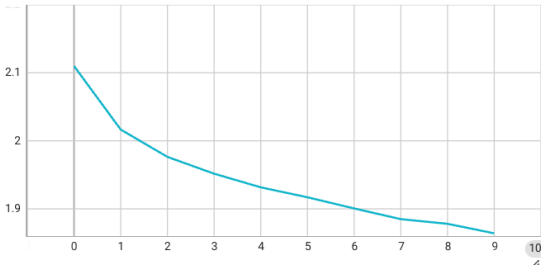


Fig. 12: Training Loss vs Epochs for Basic Net

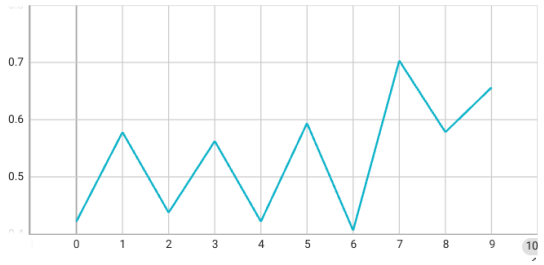


Fig. 13: Training Accuracy vs Epochs for BasicNet

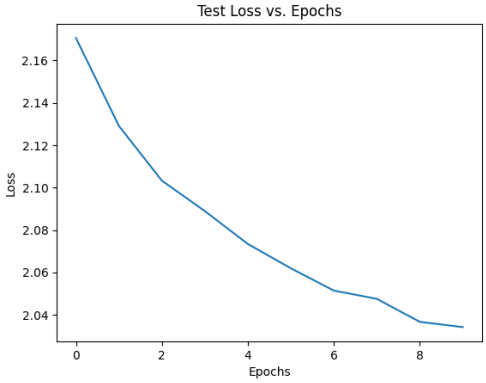


Fig. 14: Testing Loss vs Epochs for Basic Net

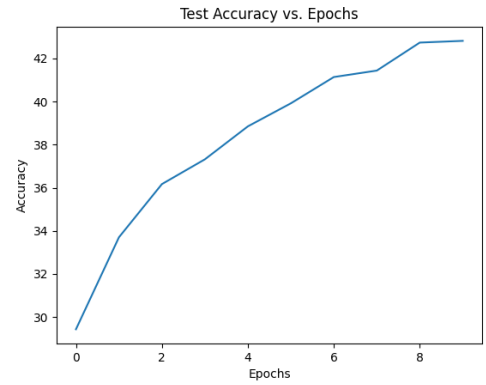


Fig. 15: Testing Accuracy vs Epochs for BasicNet

C. ResNet

For the scope of this assignment, a ResNet [2] comprising two ResNet blocks followed by two fully-connected layers was implemented. Each block is a standard implementation of a residual block. The first block has a projection layer that transforms the 3-channel input into a high-dimensional feature vector. Adam optimizer, learning rate of 0.0001, and a batch size of 64 were used and the network was trained for 10 epochs. Please refer to Fig 23 for architecture, Fig 24 for testing confusion matrix, Fig 25 training confusion matrix, Fig 26 for training loss and Fig 27 for training accuracy, Fig 35 for testing loss, and Fig 36 for testing accuracy.

D. ResNeXt

A ResNeXt [3] comprising 4 standard ResNeXt blocks was implemented followed by 1 fully-connected layer. The cardinality, which defines the number of ”parallel paths” the input data takes was set to 3. 2D Averaging pooling was used to reduce the spatial dimensions before it was fed to the fully-connected layer. Adam optimizer, learning rate of 0.0001, and a batch size of 64 were used and the network was trained for 10 epochs. Please refer to Fig 30 for architecture, Fig 31 for testing confusion matrix, Fig 32 training confusion matrix, Fig

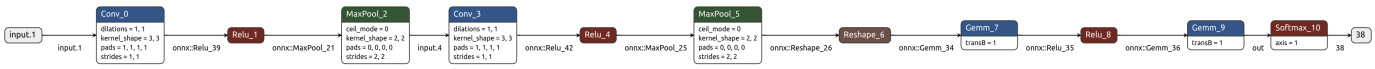


Fig. 16: Network Architecture for CNN

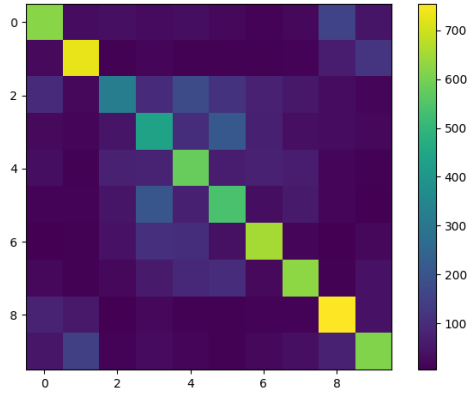


Fig. 17: Confusion Matrix for CNN

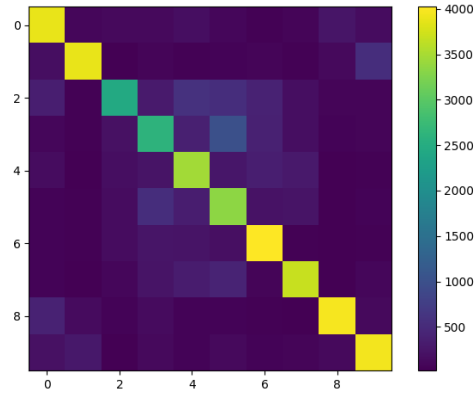


Fig. 18: Training Confusion Matrix for CNN

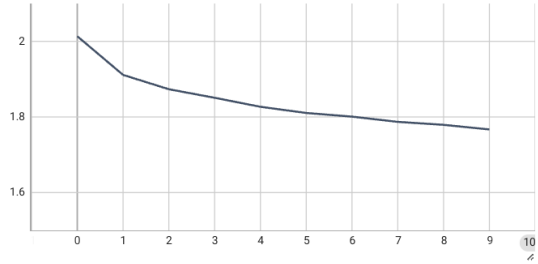


Fig. 19: Training Loss vs Epochs for CNN

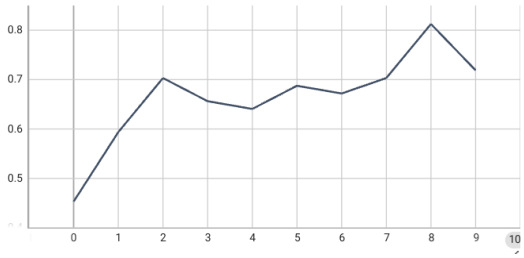


Fig. 20: Training Accuracy vs Epochs for CNN

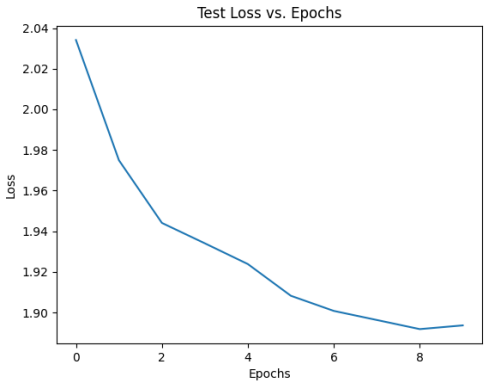


Fig. 21: Testing Loss vs Epochs for CNN

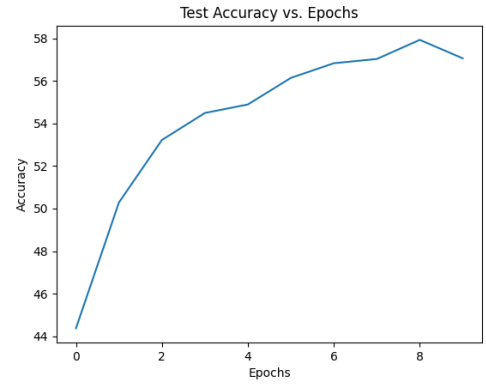


Fig. 22: Testing Accuracy vs Epochs for CNN

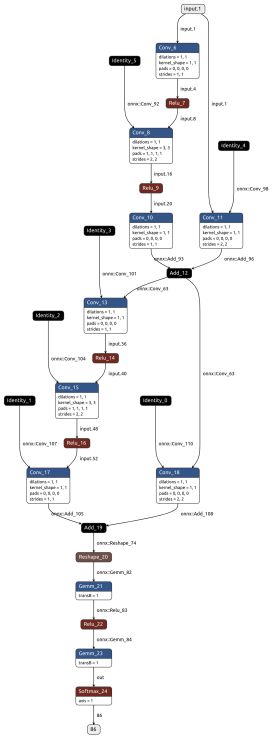


Fig. 23: Network Architecture for ResNet

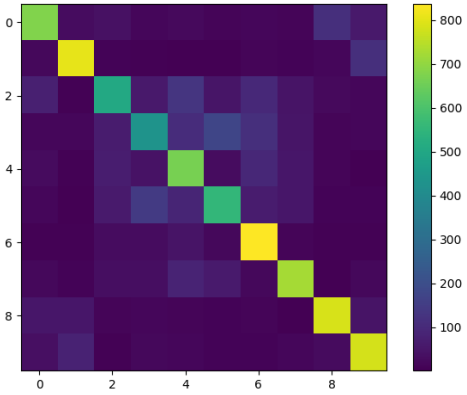


Fig. 24: Confusion Matrix for ResNet

33 for training loss, Fig 34 for training accuracy, Fig 28 for testing loss, and Fig 28 for testing accuracy.

E. DenseNet

A DenseNet [4] comprising of a dense block, transition layer, and 1 fully-connected layer was implemented. The input was projected to 64 channels through a convolutional layer. The dense block had 6 standard densenet layers while the transition layer reduced the number of channels of the latent vector. 2D averaging pooling was used to reduce the spatial dimensions of the latent vector before it was fed to the fully-connected layer. Adam optimizer, learning rate of 0.0001, and a batch size of 64 were used and the network was trained for

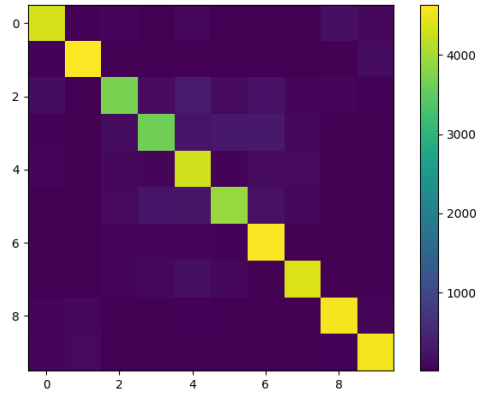


Fig. 25: Training Confusion Matrix for ResNet

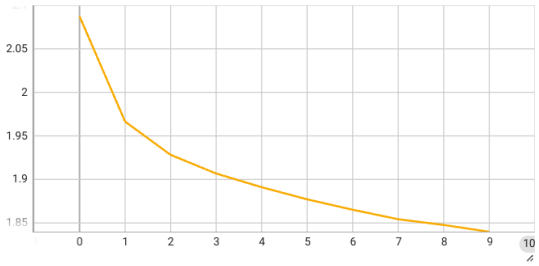


Fig. 26: Loss for ResNet

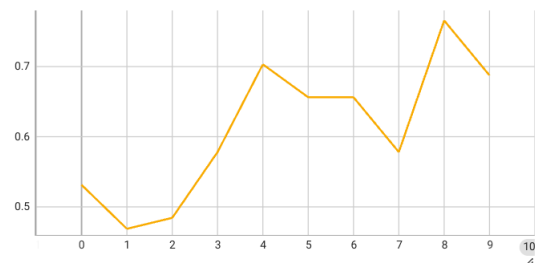


Fig. 27: Accuracy for ResNet

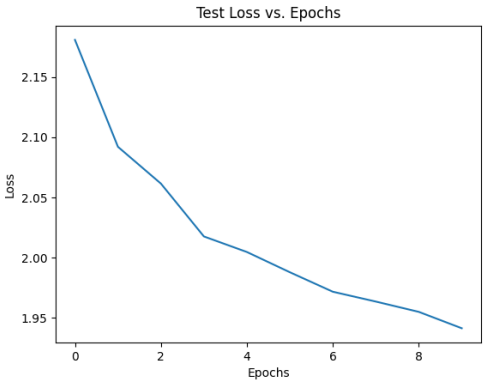


Fig. 28: Testing Loss vs Epochs for ResNet

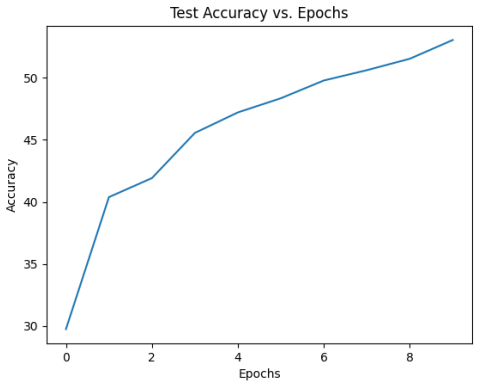


Fig. 29: Testing Accuracy vs Epochs for ResNet

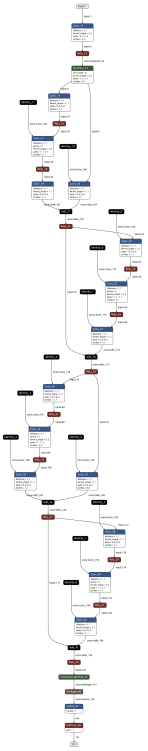


Fig. 30: Architecture for ResNeXt

10 epochs. Please refer to Fig 37 for architecture, Fig 38 for testing confusion matrix, Fig 39 training confusion matrix, Fig 40 for training loss, Fig 41 for training accuracy, Fig 42 for testing loss, and Fig 43 for testing accuracy.

F. Comparison

5 neural network architectures were implemented from scratch out of which DenseNet performed the best. A detailed analysis is documented in Table I. ResNeXt and ResNet need hyperparameter tuning and more layers to boost their performance. CNN improved the performance of Basic CNN through the introduction of batchnorm layers.

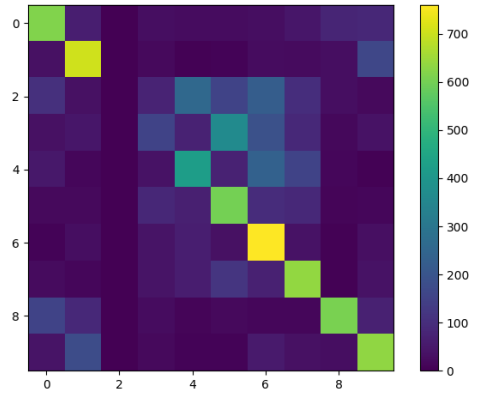


Fig. 31: Confusion Matrix for ResNeXt

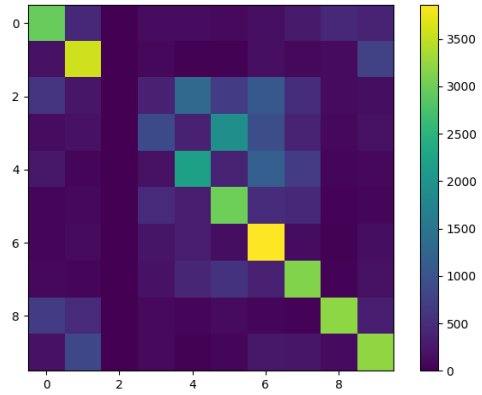


Fig. 32: Training Confusion Matrix for ResNeXt

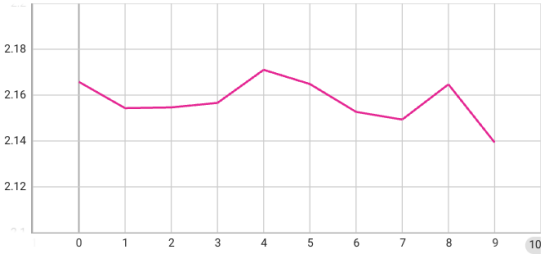


Fig. 33: Training Loss vs Epochs for ResNeXt

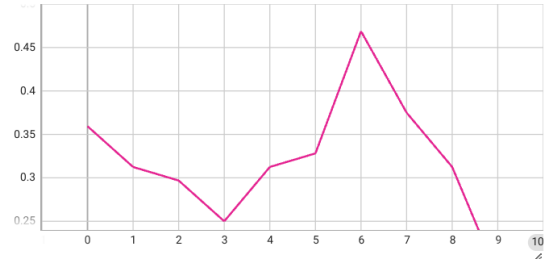


Fig. 34: Training Accuracy vs Epochs for ResNeXT

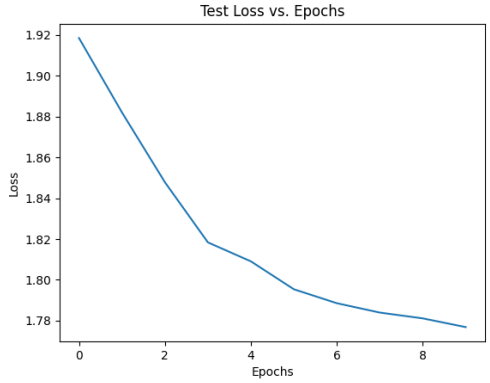


Fig. 35: Testing Loss vs Epochs for ResNeXt

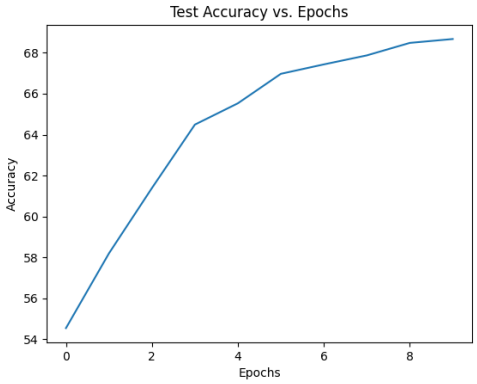


Fig. 36: Testing Accuracy vs Epochs for ResNeXT

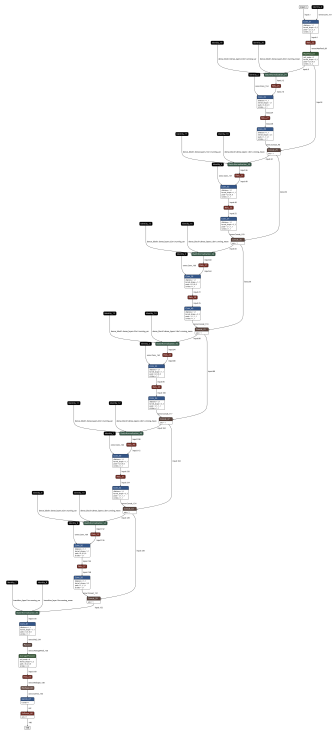


Fig. 37: Network Architecture for DenseNet

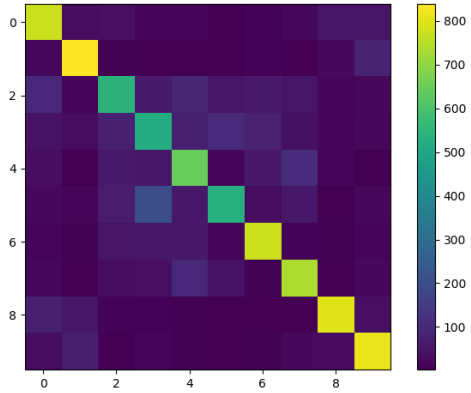


Fig. 38: Confusion Matrix for DenseNet

REFERENCES

[1] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik, "Contour detection and hierarchical image segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 33, no. 5, pp. 898–916, 2010. 1

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385 5

[3] S. Xie, R. B. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," *CoRR*, vol. abs/1611.05431, 2016. [Online]. Available: http://arxiv.org/abs/1611.05431 5

[4] G. Huang, Z. Liu, and K. Q. Weinberger, "Densely connected convolutional networks," *CoRR*, vol. abs/1608.06993, 2016. [Online]. Available: http://arxiv.org/abs/1608.06993 7

Model	Params	Train Accuracy	Test Accuracy	Inference(ms)
Basic CNN	133,538	60.2%	56.13%	0.282
CNN	133,582	70.63	63.51%	0.341
ResNet	1,294,026	79.21%	67.5%	0.848
ResNeXt	15,018	52%	51.36 %	1.664
DenseNet	393,738	79.03%	69.7 %	1.551

TABLE I: Comparison of network performance on CIFAR-10 dataset

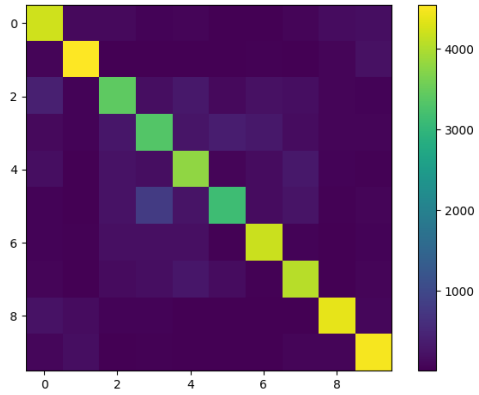


Fig. 39: Training Confusion Matrix for DenseNet

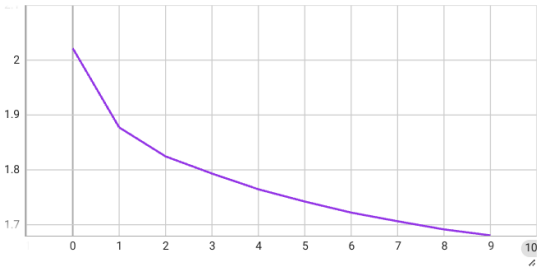


Fig. 40: Training Loss vs Epochs for DenseNet

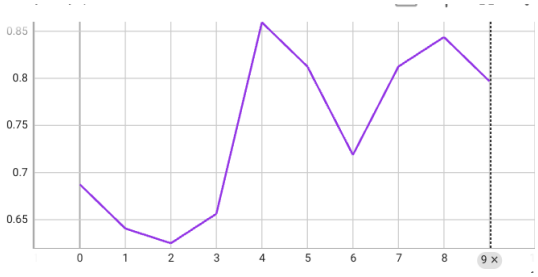


Fig. 41: Training Accuracy vs Epochs for DenseNet

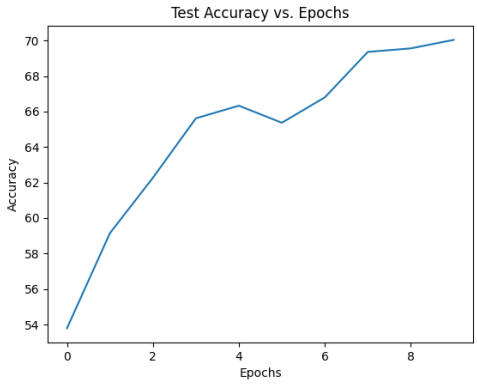


Fig. 43: Testing Accuracy vs Epochs for DenseNet

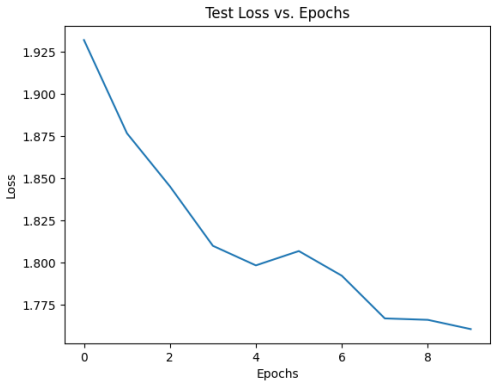


Fig. 42: Testing Loss vs Epochs for DenseNet