Homework 0: Alohomora

Using 1 Late Day
Pradnya Sushil Shinde
RBE 549: Computer Vision
Worcester Polytechnic Institute
Email: pshinde1@wpi.edu

***Abstract*—The objective of the homework is to implement an edge detection task using the classical pb (probability of boundary) detection algorithm and an image classification task using a deep learning approach. As a part of the classical approach, the BSDS500 dataset has been passed through a filter bank consisting of Oriented Derivatives of Gaussian (DoG) filters, Leung-Malik filters, and Gabor filters. Further, each of these images has been graded map and gradient features based on Texture, Brightness, and Color properties. As a final step, each of the gradients of images has been averaged out and combined with the Canny and Sobel baselines. Image classification in the second phase has been performed using a Basic Network, ResNet, ResNeXt, and Dense Network architecture. Various hyperparameters such as learning rate, and number of epochs have been varied to obtain test and train accuracy and loss.**

I. Phase 1: Shake My Boundary

A. *Filter Banks*

1) Oriented DoG Filters: We will generate Derivatives of Guassians at 12 orientations with two scales each scale having different kernel sizes. Following is the flow of the algorithm:

- Get a 2D gaussian kernel using:

$$G = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

- Define Sobel operators

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad S_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

- Convolve Sobel operators and Gaussian kernel to generate filters in X-Y directions

$$G_x = S_x * G$$

$$G_y = S_y * G$$

- Rotate the filters at given orientations

$$G_{oriented} = G_x \cos\theta + G_y \sin\theta$$

Visualization of the filters is shown in Figure 1.

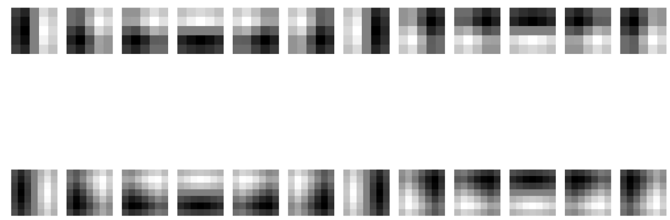


Fig. 1: Oriented DoG Filters

2) Leung-Malik Filters: Two types of Leung-Malik filters have been generated: Leung-Malik Small and Leung-Malik Large with In LM Small (LMS), the filters occur at basic scales $\sigma = \{1, \sqrt{2}, 2, 2\sqrt{2}\}$. The first and second derivative filters occur at the first three scales with an elongation factor of 3, i.e.$\sigma_x = \sigma$ & $\sigma_y = 3\sigma_x$. 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 $\sigma = \{\sqrt{2}, 2, 2\sqrt{2}, 4\}$. Following is the flow of the algorithm to generate LM filters:

- Generate 1D Gaussian kernels in both X-Y directions.

$$G_{\text{1D}} = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot \exp\left(-\frac{x^2}{2\sigma^2}\right)$$

- Generate first order and second order derivates of the Gaussian kernel

$$G' = (-\frac{x}{\sigma^2}) \cdot G_{\text{1D}}$$

$$G'' = (\frac{x^2}{\sigma^4} - \frac{1}{\sigma^2}) \cdot G'$$

- Get 2D Gaussian kernels as calculated for Oriented DoG filters
- Generate Laplacians of Gaussains

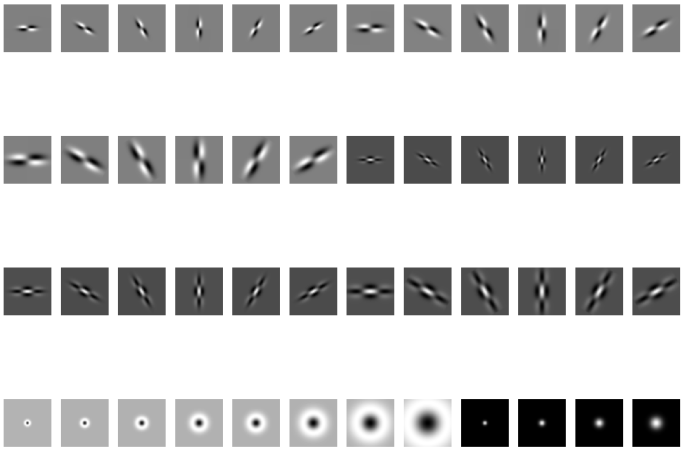
$$\text{LoG} = \frac{1}{\sqrt{\pi\sigma^2}} \left(\frac{x^2+y^2}{\sigma^4} - \frac{1}{\sigma^2}\right) \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

- Add all the filters to generate LM filter bank.
Visualization of filters is shown in Fig 2.

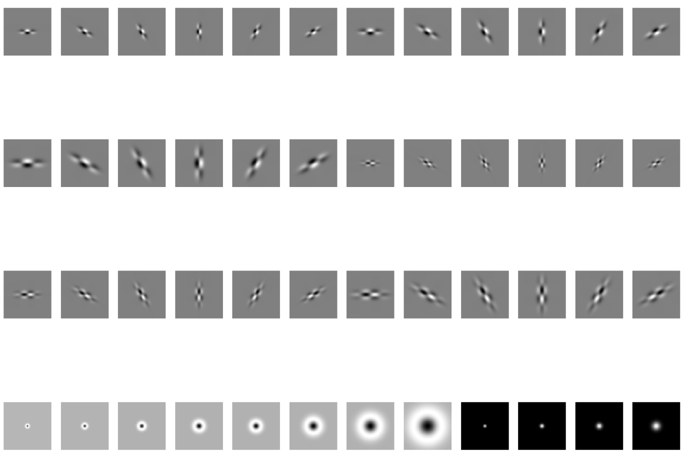
3) Gabor Filters: A gabor filter is a gaussian kernel function modulated by a sinusoidal plane wave.

- Generate Gabor filter using the following equation.

$$\text{Gb} = \exp\left(-\frac{1}{2}\left(\frac{x^2+(\gamma^2 y^2)}{\sigma^2}\right)\right)\cos\left(2\pi\left(\frac{x}{\lambda}\right)+\psi\right)$$



(a) LM Large



(b) LM Small

Fig. 2: Leung-Malik Filter Bank

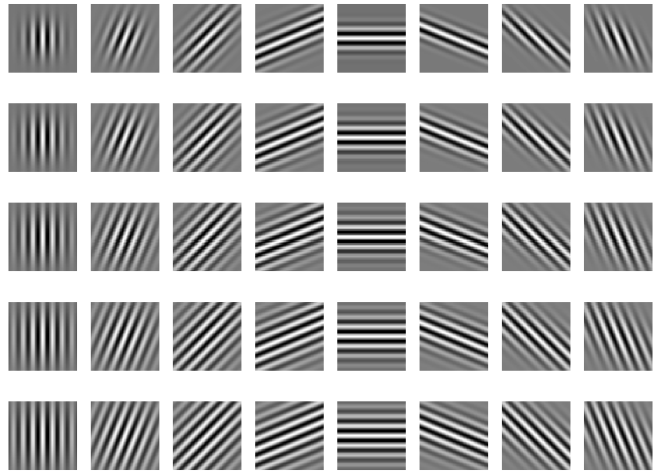


Fig. 3: Gabor Filters

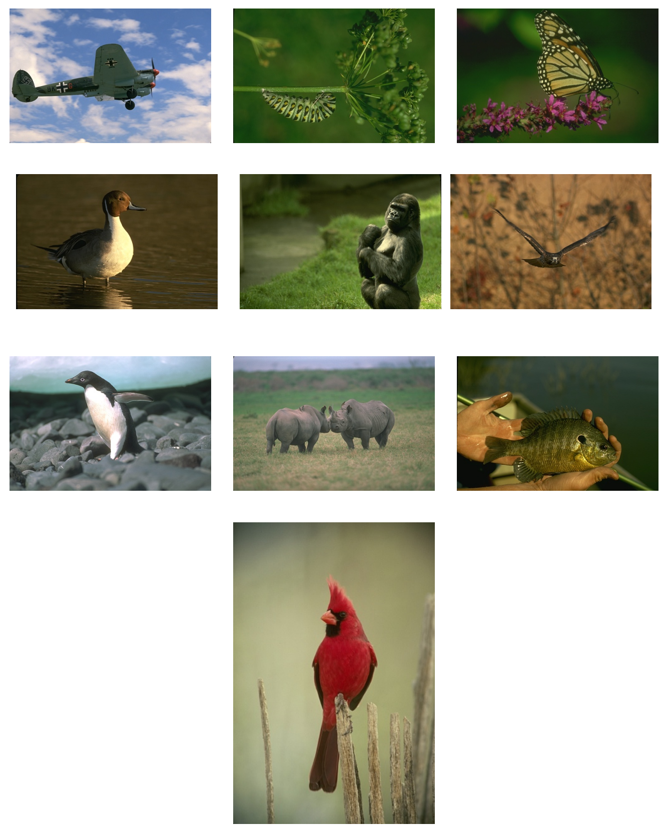


Fig. 4: Input Images

- A fixed value of $\lambda = 7, \psi = 0$ and $\gamma = 1$ has been used to calculate the filters. The σ values have been varied with $\{9, 11, 13, 15, 17\}$
- Visualization of Gabor filters is shown in Figure 3.

B. Feauture Maps

To generate , the Texture, Brightness, and Color properties of the image are considered. Firstly, each image is converted to grayscale and passed through the curated filter bank consisting of Oriented DoG, Leung-Malik, and Gabor filters. The resulting filtered images are collected as filter responses and are reshaped to $W \times H \times N$ where W is the width of the image, H is the height of the image and N is the number of filters. They are then clustered into a given number of clusters at all pixels using the KMeans algorithm.

1) Texton Map: For generating texton maps, we will use clusters = 64. See Fig. 5

2) Brightness Map: For generating texton maps, we will use clusters = 16. See Fig. 6.

3) Color Map: For generating texton maps, we will use clusters = 16. See Fig. 7

C. Half Disc Masks

We will now generate Half-Disc Masks which are essentially binary images where the semi-circular discs are oriented at different angles. Every mask consists of two pairs of such discs which are mirrored to each other. For visualization, see Fig. 8.

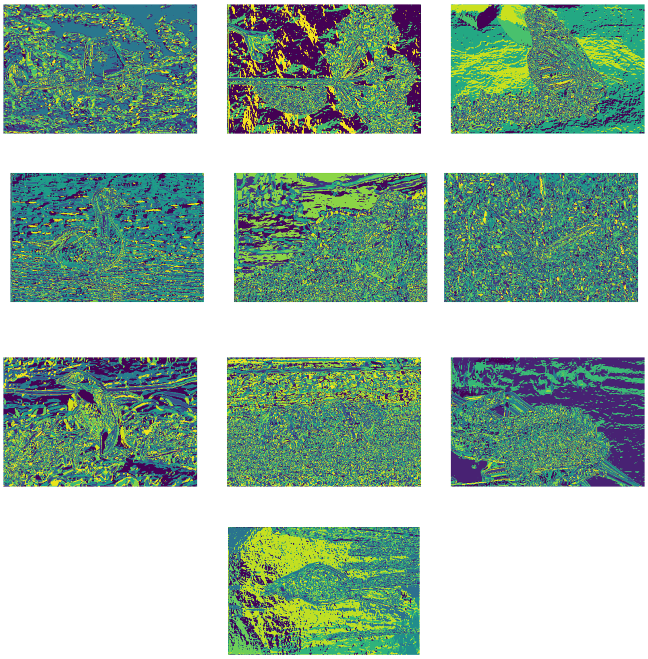


Fig. 5: Texton Maps

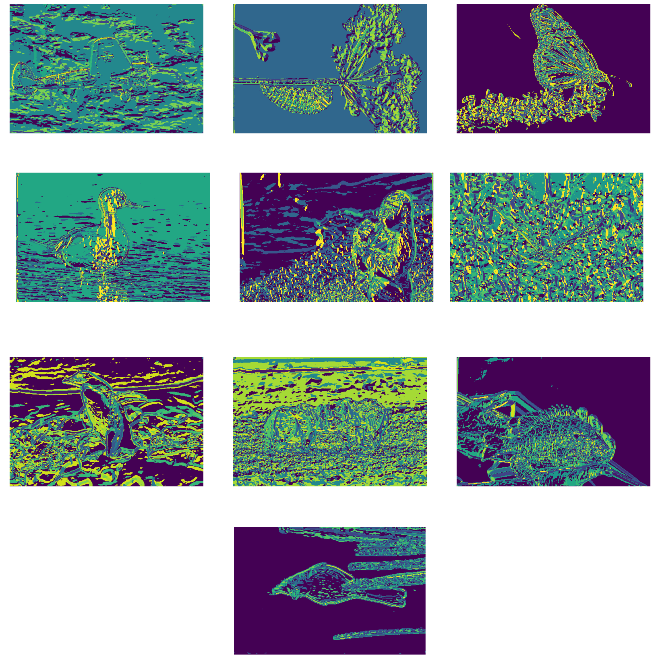


Fig. 7: Color Maps

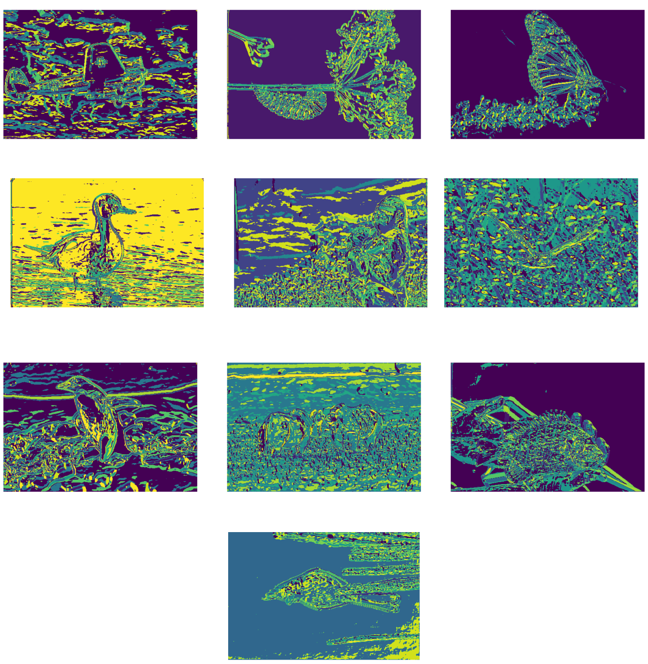


Fig. 6: Brightness Maps

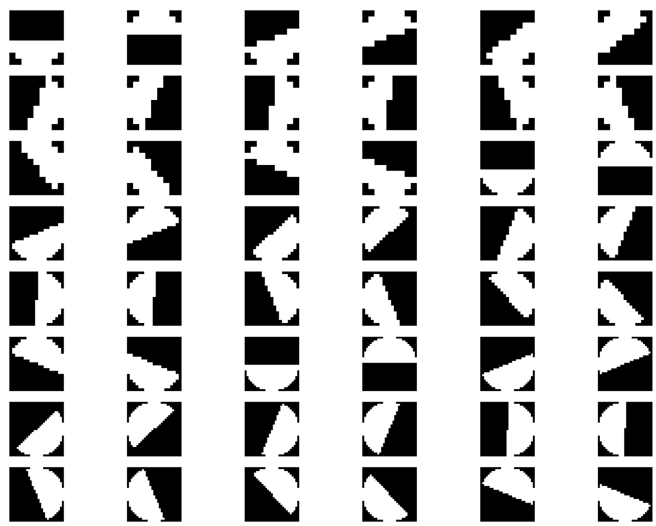


Fig. 8: Half-Disc Masks

D. *Feauture Gradients*

Now that we have the Half-Disc Masks, we will move forward with generating gradients for each of the texture, brightness, and color properties of the images. The procedure for the same is as follows:

- Divide the half-disc masks into a set of left masks and right masks.
- Perform convolution of maps obtained in the previous section with the left and right masks individually.

$$g_i = FeatureMap * LeftMask$$

$$h_i = FeautureMap * RightMask$$

- Calculate chi-square distance.

$$\text{dst} = \frac{(g_i - h_i)^2}{2(g_i + h_i + 1 \times 10^{-10})}$$

- Update chi-square values and return their mean as the feature gradient

1) Texture Gradient: To generate texton gradients, we will use the number of bins as 64 to calculate the chi-square distance. See Fig. 9 for visualization of texture gradients.

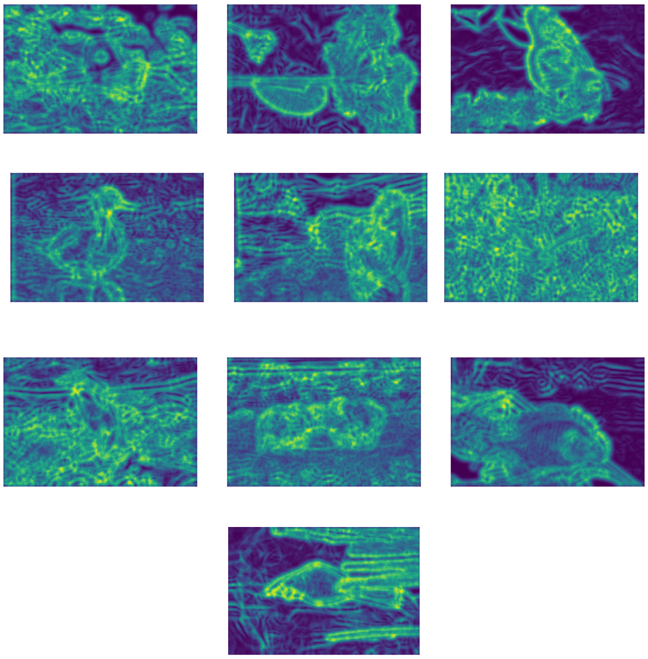


Fig. 9: Texton Gradients

2) Brightness Gradient: To generate brightness gradients, we will use the number of bins as 16 to calculate the chi-square distance. See Fig. 10 for visualization of brightness gradients.

3) Color Gradients: To generate color gradients, we will use the number of bins as 16 to calculate the chi-square distance. See Fig. 11 for visualization of color gradients.

E. Boundary Detection

We have been provided with Canny and Sobel results for Image BOundary Detection. Out next and final step is to combine the results of image gradients obtained previously with the Canny and Sobel baseline to obtain the classical Pb-Lite edge detection algorithm.

1) Canny Baseline: Image boundary detection using the Canny Baseline has been shown in Fig 12.

2) Sobel Baseline: Image boundary detection using the SObel Baseline has been shown in the following Fig 13.

3) Pb-Lite: The probability of Boundary (Pb) lite edge detection can be computed by averaging out both the gradients and Canny-Sobel baselines and then eventually multiplying them.

$$\text{pb\_lite} = (0.33^* T_g + 0.33^* B_g + 0.33^* C_g) \odot (0.5^*(sPb + cPb))$$

Image boundary detection using the Pb-Lite algorithm has been shown in Fig 14.

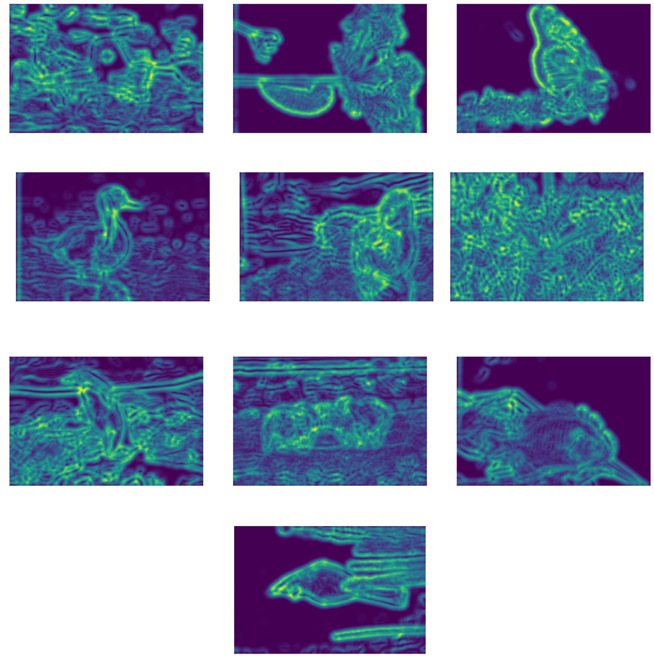


Fig. 10: Brightness Gradients

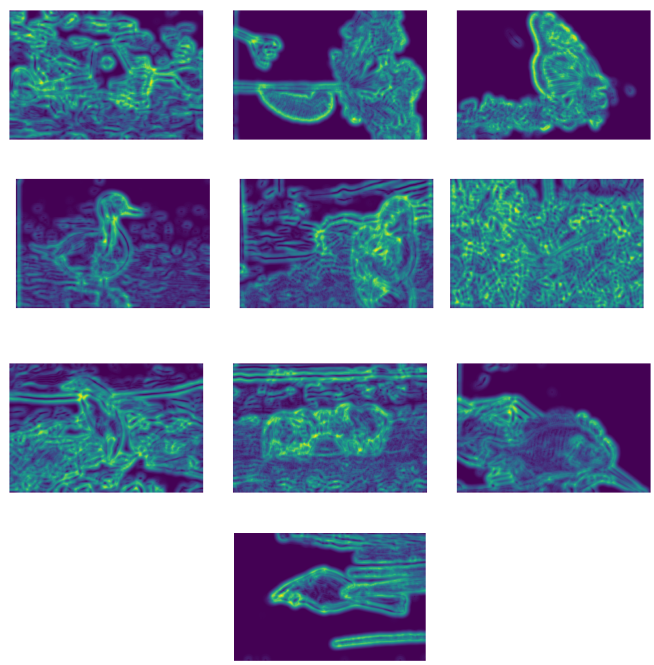


Fig. 11: Color Gradients

II. Phase II: Deep Dive on Deep Learning

A. Training a Basic Network

1) Implementation: The task in this part is to train a convolutional neural network on PyTorch for the task of classification. The input is a single CIFAR-10 image and the output is the probabilities of 10 classes. Following are the

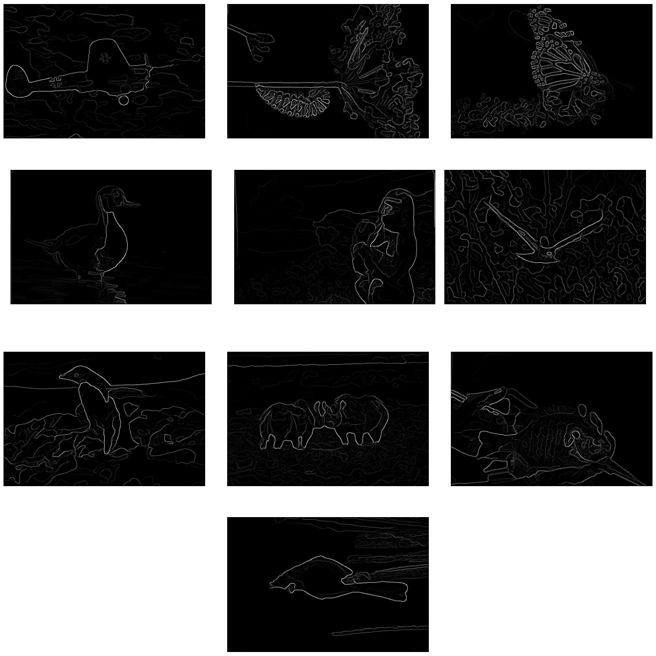


Fig. 12: Canny Baseline

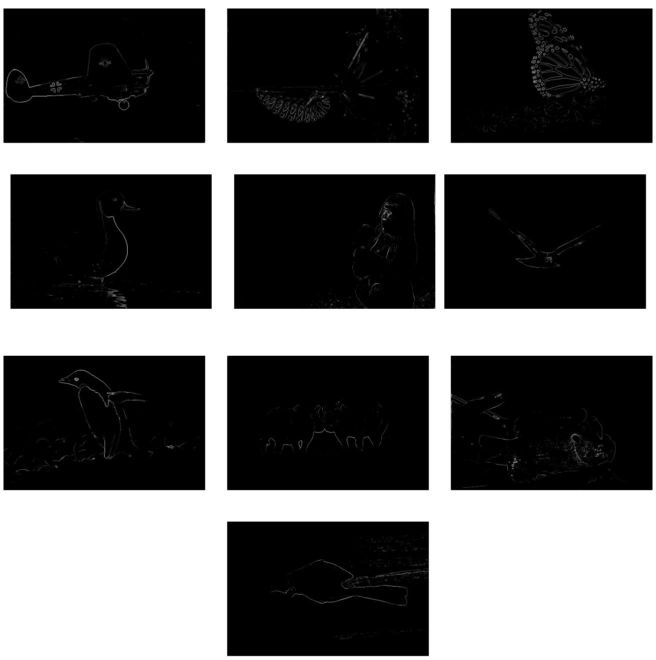


Fig. 13: Sobel Baseline

details concerning the implementation of the network. See Fig 15. for the architecture of the Neural Network.

Image transformation used is:

- Random Horizontal Flip:

transforms.RandomHorizontalFlip()

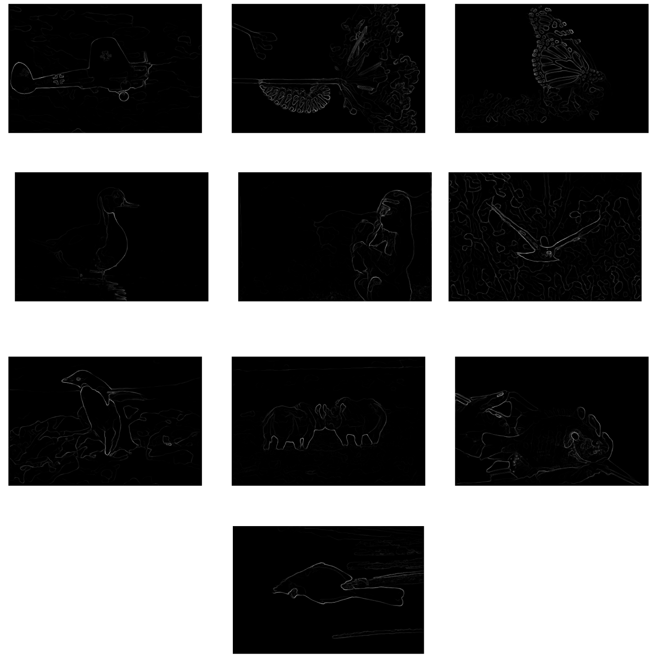


Fig. 14: Pb-Lite

TABLE I: Basic Neural Network Configuration

Hyper Parameter	Value
Optimizer	SGD
Momentum	0.9
Learning Rate	0.001
Num of Epochs	20
Batch Size	50

- Random Rotation:

transforms.RandomRotation(10)

This rotates the image by a random angle in the range $[-10, 10]$ degrees.
- Convert to Tensor:

transforms.ToTensor()

This converts the image to a PyTorch tensor.

2) Evaluation: The train and test accuracy plots were obtained during the training phase of the above network configuration.

—Test Accuracy
—Train Accuracy
Loss over epochs has been plotted as shown in Fig 17.
—Test Loss
—Train Loss

TABLE II: Basic Network Model Test Output

Parameter Count	545,226
Training Accuracy	86.406%
Testing Accuracy	74.27%

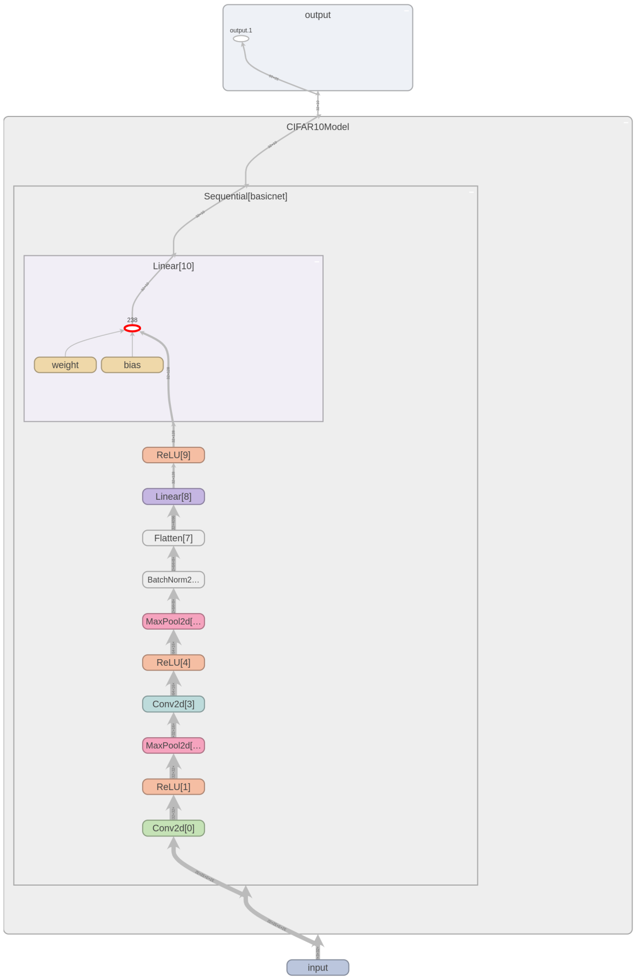


Fig. 15: Basic CNN: SGD Optimizer

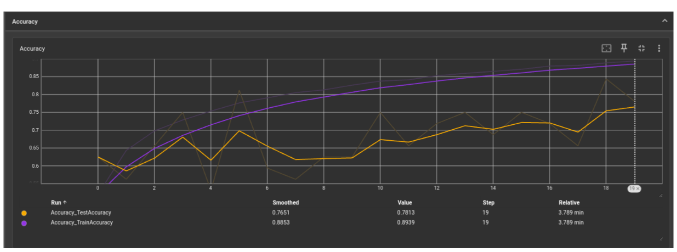


Fig. 16: Basic Network: Accuracy Plot

B. *Improving Accuracy*

1) Implementation: The previous network used Stochastic Gradient Descent with Momentum as an optimizer which updates the model parameters based on the negative gradient descent of the loss concerning each parameter multiplied by a fixed learning rate. On the other hand, AdamW has an adaptive learning rate that updates model parameters based on historical information. Moreover, we will also reduce the batch size and number of epochs to avoid overfitting. Following is the configuration of the IMPROVED Basic Neural Network. The

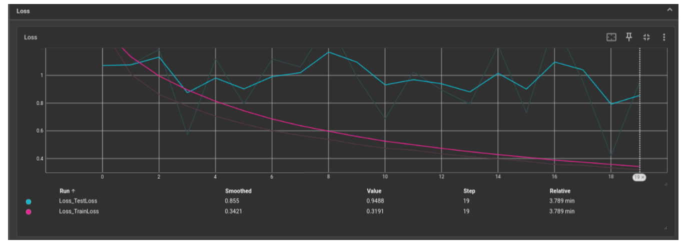


Fig. 17: Basic Network: Loss Plot

Basic Network Confusion Matrix of Testing Data

$$\begin{bmatrix} 768 & 10 & 28 & 21 & 18 & 11 & 8 & 15 & 78 & 43 \\ 22 & 808 & 3 & 11 & 3 & 3 & 8 & 4 & 36 & 102 \\ 57 & 6 & 630 & 53 & 77 & 66 & 42 & 39 & 25 & 5 \\ 28 & 5 & 59 & 485 & 76 & 189 & 66 & 49 & 22 & 21 \\ 15 & 2 & 71 & 44 & 699 & 37 & 41 & 70 & 18 & 3 \\ 14 & 2 & 41 & 120 & 50 & 676 & 25 & 47 & 15 & 10 \\ 7 & 5 & 39 & 40 & 32 & 29 & 826 & 8 & 10 & 4 \\ 14 & 3 & 20 & 16 & 48 & 38 & 5 & 828 & 6 & 22 \\ 42 & 22 & 4 & 12 & 6 & 8 & 7 & 2 & 877 & 20 \\ 25 & 51 & 5 & 10 & 5 & 4 & 7 & 14 & 49 & 830 \end{bmatrix}$$

Basic Network Confusion Matrix of Training Data

$$\begin{bmatrix} 4415 & 22 & 79 & 30 & 52 & 37 & 13 & 32 & 250 & 70 \\ 51 & 4521 & 5 & 12 & 7 & 9 & 8 & 14 & 108 & 265 \\ 199 & 7 & 4044 & 115 & 197 & 147 & 111 & 93 & 62 & 25 \\ 89 & 15 & 163 & 3383 & 183 & 667 & 182 & 163 & 71 & 84 \\ 49 & 2 & 217 & 122 & 4169 & 110 & 82 & 197 & 34 & 18 \\ 22 & 5 & 128 & 312 & 113 & 4129 & 78 & 148 & 36 & 29 \\ 24 & 10 & 130 & 105 & 84 & 42 & 4546 & 17 & 26 & 16 \\ 21 & 3 & 50 & 58 & 141 & 88 & 8 & 4574 & 19 & 38 \\ 85 & 26 & 12 & 12 & 6 & 13 & 6 & 9 & 4799 & 32 \\ 64 & 94 & 13 & 25 & 18 & 15 & 10 & 26 & 112 & 4623 \end{bmatrix}$$

architecture of the network has not been changed.

Image transformation used is:

transforms.ToTensor()

2) Evaluation: The train and test accuracy plots were obtained during the training phase of the above network configuration.

—Test Accuracy

—Train Accuracy

Loss over epochs has been plotted as shown in Fig 19.

—Test Loss

—Train Loss

TABLE III: Basic Neural Network Configuration (IMPROVED)

Parameters	**Value**
Optimizer	AdamW
Learning Rate	0.001
Num of Epochs	15
Batch Size	50

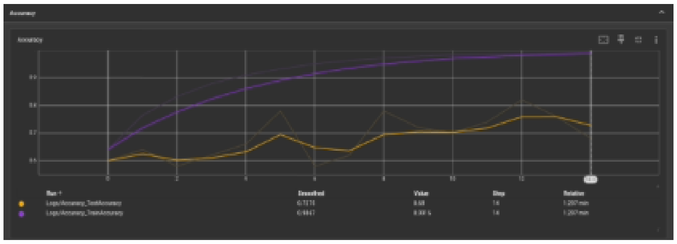


Fig. 18: Basic Network Improved: Accuracy Plot

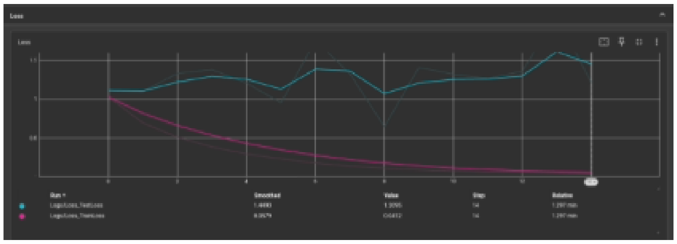


Fig. 19: Basic Network Improved: Loss Plot

$$\text{Improved Network Confusion Matrix of testing data}$$
$$\begin{bmatrix} 678 & 24 & 69 & 26 & 24 & 14 & 17 & 14 & 100 & 34 \\ 18 & 828 & 12 & 7 & 3 & 7 & 11 & 5 & 34 & 75 \\ 57 & 7 & 557 & 67 & 99 & 77 & 72 & 38 & 21 & 5 \\ 18 & 21 & 74 & 475 & 76 & 198 & 75 & 27 & 15 & 21 \\ 24 & 5 & 69 & 76 & 647 & 43 & 60 & 60 & 13 & 3 \\ 21 & 8 & 45 & 152 & 50 & 609 & 46 & 56 & 7 & 6 \\ 5 & 7 & 40 & 51 & 44 & 30 & 801 & 12 & 8 & 2 \\ 9 & 3 & 37 & 46 & 82 & 70 & 13 & 720 & 8 & 12 \\ 40 & 46 & 21 & 17 & 9 & 10 & 12 & 7 & 822 & 16 \\ 27 & 113 & 13 & 26 & 4 & 15 & 16 & 17 & 33 & 736 \end{bmatrix}$$

$$\text{Improved Network Confusion Matrix of training data}$$
$$\begin{bmatrix} 4701 & 14 & 68 & 28 & 27 & 19 & 9 & 9 & 114 & 11 \\ 4 & 4946 & 4 & 5 & 1 & 1 & 3 & 1 & 19 & 16 \\ 15 & 2 & 4768 & 34 & 68 & 34 & 54 & 11 & 12 & 2 \\ 9 & 5 & 32 & 4709 & 37 & 109 & 64 & 22 & 8 & 5 \\ 15 & 3 & 41 & 18 & 4849 & 22 & 26 & 19 & 4 & 3 \\ 2 & 3 & 25 & 33 & 19 & 4873 & 23 & 17 & 4 & 1 \\ 3 & 3 & 20 & 9 & 11 & 6 & 4944 & 1 & 2 & 1 \\ 4 & 5 & 14 & 21 & 61 & 27 & 12 & 4853 & 1 & 2 \\ 18 & 17 & 10 & 6 & 2 & 4 & 6 & 3 & 4927 & 7 \\ 10 & 82 & 7 & 8 & 1 & 5 & 11 & 6 & 27 & 4843 \end{bmatrix}$$

C. *ResNet*

1) Implementation: A Residual Network has been implemented to further improve the overall performance of Image Classification. ResNet uses a "shortcut" which can be defined as the identity of input. This identity bypasses one more layer and helps overcome the issue of vanishing gradient. Vanishing gradients are common as the depth of the network is increased.

TABLE IV: Improved Network Model Test Output

Parameter Count	545,226
Training Accuracy	96.826%
Testing Accuracy	68.73%

See Fig 20. for the architecture of the Neural Network.

TABLE V: Residual Network

Parameters	Value
Optimizer	AdamW
Learning Rate	0.001
Num of Epochs	20
Batch Size	200
Configuration	$[2, 2, 2]$

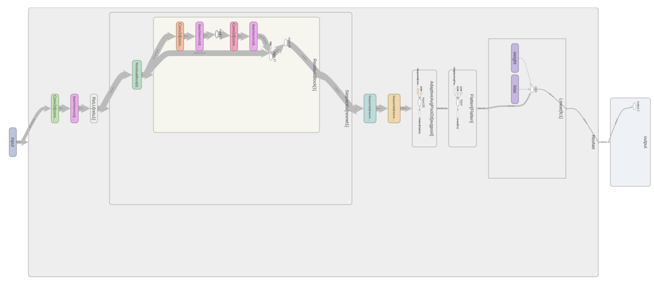


Fig. 20: ResNet

2) Evaluation: The train and test accuracy plots were obtained during the training phase of the above network configuration.

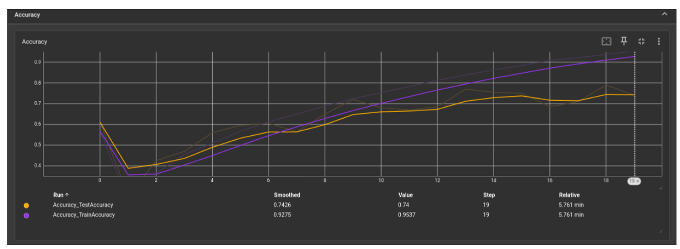


Fig. 21: ResNet: Accuracy Plot

—Test Accuracy
—Train Accuracy
Loss over epochs has been plotted as shown in Fig 22.

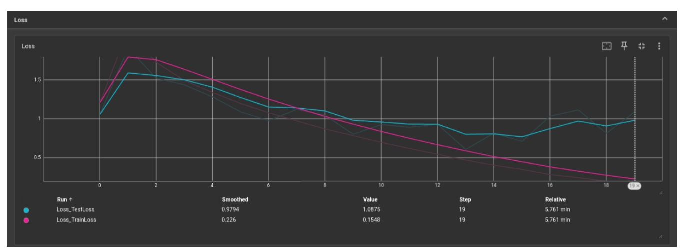


Fig. 22: ResNet: Loss Plot

—Test Loss
—Train Loss

D. *ResNeXt*

1) Implementation: Residual NeXt implements a similar "shortcut" but in a parallel network manner. It introduces the

TABLE VI: ResNet Model Test Output

Parameter Count	697738
Training Accuracy	93.828 %
Testing Accuracy	74.19%

ResNet Confusion Matrix of Testing Data

$$\begin{bmatrix} 828 & 7 & 32 & 27 & 11 & 6 & 8 & 11 & 45 & 25 \\ 17 & 826 & 5 & 8 & 1 & 4 & 13 & 3 & 24 & 99 \\ 77 & 2 & 579 & 68 & 73 & 55 & 91 & 40 & 6 & 9 \\ 28 & 6 & 44 & 565 & 54 & 154 & 76 & 49 & 13 & 11 \\ 34 & 1 & 55 & 57 & 676 & 41 & 56 & 69 & 6 & 5 \\ 14 & 3 & 32 & 182 & 37 & 622 & 37 & 60 & 5 & 8 \\ 7 & 6 & 25 & 65 & 40 & 18 & 825 & 6 & 6 & 2 \\ 23 & 2 & 15 & 40 & 38 & 56 & 5 & 806 & 1 & 14 \\ 77 & 16 & 9 & 20 & 3 & 2 & 7 & 6 & 835 & 25 \\ 36 & 41 & 6 & 17 & 1 & 8 & 7 & 16 & 11 & 857 \end{bmatrix}$$

ResNet Confusion Matrix of Training Data

$$\begin{bmatrix} 4880 & 1 & 33 & 13 & 6 & 7 & 7 & 10 & 29 & 14 \\ 44 & 4761 & 1 & 13 & 0 & 2 & 15 & 2 & 16 & 146 \\ 161 & 1 & 4346 & 101 & 92 & 77 & 149 & 56 & 13 & 4 \\ 30 & 2 & 35 & 4429 & 67 & 224 & 138 & 51 & 12 & 12 \\ 80 & 0 & 64 & 97 & 4552 & 35 & 66 & 97 & 7 & 2 \\ 14 & 0 & 14 & 263 & 44 & 4541 & 47 & 74 & 1 & 2 \\ 6 & 3 & 19 & 52 & 23 & 18 & 4867 & 5 & 5 & 2 \\ 12 & 0 & 12 & 31 & 30 & 34 & 9 & 4867 & 0 & 5 \\ 127 & 12 & 4 & 19 & 2 & 4 & 5 & 2 & 4800 & 25 \\ 52 & 15 & 3 & 22 & 3 & 5 & 5 & 6 & 18 & 4871 \end{bmatrix}$$

concept of cardinality and employs multiple parallel paths in skip connection.

See Fig 23. for the architecture of the Neural Network.

2) Evaluation: The train and test accuracy plots were obtained during the training phase of the above network configuration.

—Test Accuracy

—Train Accuracy

Loss over epochs has been plotted as shown in Fig 25.

—Test Loss

—Train Loss

E. DenseNet

DenseNet introduces the concept of feature reuse and dense connectivity. It also provides output concatenation with the input of subsequent layers. This provides a strong feature propagation. The architecture and configuration used for DenseNet has been provided below.

1) Evaluation: The train and test accuracy plots were obtained during the training phase of the above network configuration.

TABLE VII: Residual Next Network

Parameters	**Value**
Optimizer	AdamW
Learning Rate	0.001
Num of Epochs	15
Batch Size	128
Configuration	$[3, 4, 6, 3]$

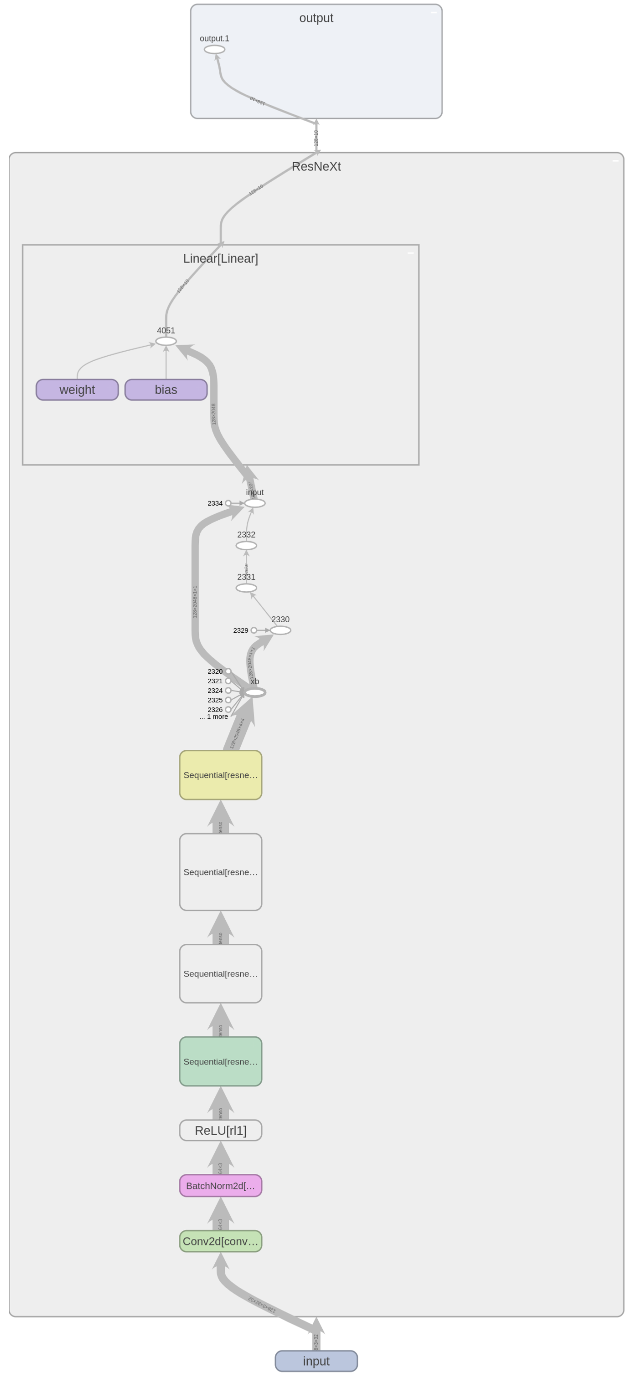


Fig. 23: ResNeXt Architiecture

—Test Accuracy

—Train Accuracy

Loss over epochs has been plotted as shown in Fig 28.

—Test Loss

—Train Loss

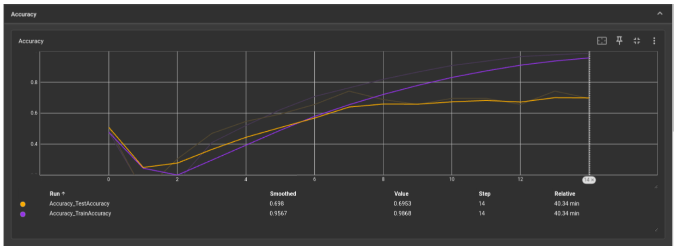


Fig. 24: ResNeXt: Accuracy Plot

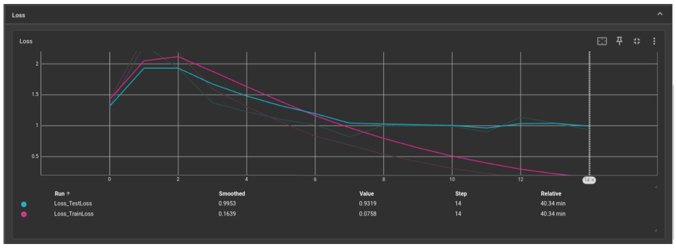


Fig. 25: ResNeXt: Loss Plot

$$\text{ResNeXt Confusion Matrix of Testing Data}$$
$$\begin{bmatrix}
749 & 13 & 62 & 40 & 19 & 7 & 10 & 6 & 79 & 15 \\
13 & 856 & 8 & 7 & 2 & 9 & 6 & 2 & 29 & 68 \\
52 & 3 & 582 & 117 & 101 & 53 & 66 & 8 & 15 & 3 \\
16 & 7 & 52 & 646 & 65 & 120 & 56 & 11 & 15 & 12 \\
13 & 6 & 74 & 77 & 663 & 45 & 68 & 44 & 6 & 4 \\
9 & 1 & 27 & 286 & 60 & 568 & 18 & 22 & 7 & 2 \\
6 & 3 & 35 & 103 & 36 & 20 & 780 & 3 & 10 & 4 \\
19 & 8 & 30 & 87 & 83 & 72 & 9 & 683 & 1 & 8 \\
59 & 22 & 18 & 21 & 4 & 8 & 8 & 1 & 844 & 15 \\
33 & 72 & 8 & 34 & 7 & 6 & 6 & 3 & 25 & 806
\end{bmatrix}$$

$$\text{ResNeXt Confusion Matrix of Training Data}$$
$$\begin{bmatrix}
4768 & 10 & 45 & 41 & 35 & 8 & 4 & 4 & 81 & 4 \\
13 & 4909 & 2 & 6 & 1 & 0 & 3 & 2 & 25 & 39 \\
55 & 1 & 4620 & 127 & 74 & 38 & 60 & 5 & 19 & 1 \\
6 & 0 & 32 & 4803 & 43 & 55 & 43 & 6 & 6 & 6 \\
5 & 0 & 46 & 84 & 4733 & 34 & 58 & 31 & 9 & 0 \\
6 & 5 & 26 & 370 & 56 & 4478 & 32 & 19 & 5 & 3 \\
6 & 7 & 29 & 92 & 33 & 16 & 4809 & 0 & 5 & 3 \\
16 & 6 & 34 & 132 & 99 & 107 & 13 & 4571 & 12 & 10 \\
24 & 7 & 11 & 21 & 6 & 3 & 3 & 1 & 4917 & 7 \\
46 & 46 & 8 & 47 & 3 & 3 & 5 & 4 & 19 & 4819
\end{bmatrix}$$

F. Analysis

To conclude, AdamW showed a better performance compared to Stochastic Gradient DenseNet with momentum and hence AdamW was chosen for further training. Following is the analysis of the four networks implemented:

- The basic network was able to achieve a higher training accuracy but comparatively lower testing accuracy. A possible cause for this can be overfitting where the model learns efficiently but fails to generalize the new dataset.
- ResNet and ResNext showcase a drastic increase in training accuracy which means the model is very well efficient in learning. However, the gap between training and testing accuracy has reduced but still remains. Increasing the depth and standardizing the data may help overcome the gap.
- DenseNet performs exceptionally well in bridging the gap between training and testing accuracy. This shows that it is more like to generalize new data better as compared to other networks. One thing to note here is the number of epochs in training has been subsequently reduced

TABLE VIII: ResNeXt Model Test Output

Parameter Count	23019146
Training Accuracy	94.854 %
Testing Accuracy	71.77 %

TABLE IX: Dense Network

Parameters	Value
Optimizer	AdamW
Learning Rate	0.001
Num of Epochs	15
Batch Size	64
Growth Rate	12
Depth	32
Reduction Factor	0.5

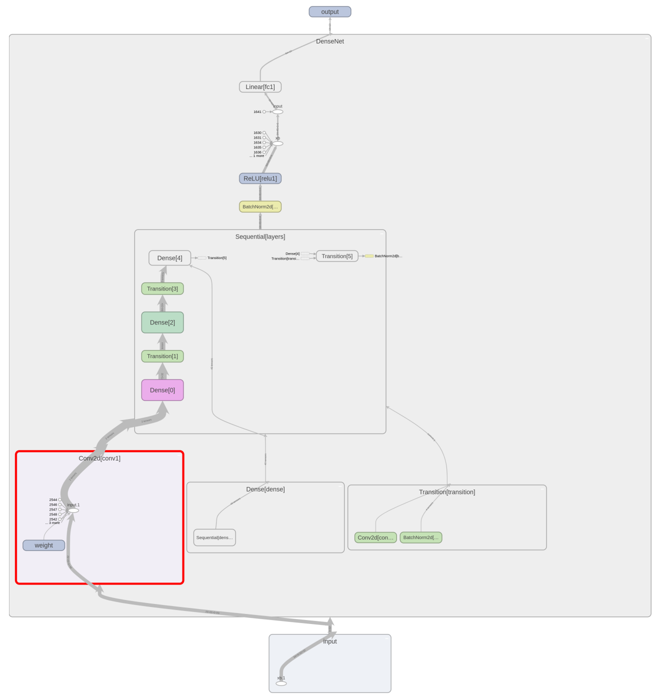


Fig. 26: DenseNet

TABLE X: DenseNet Model Test Output

Parameter Count	105214
Training Accuracy	84.854 %
Testing Accuracy	78.23 %

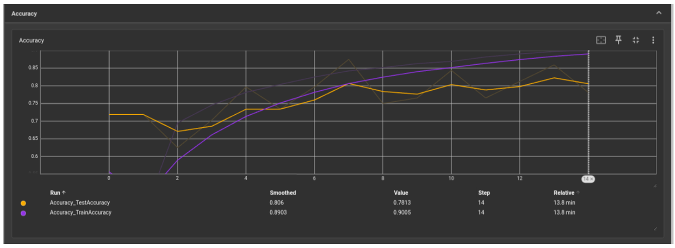


Fig. 27: DenseNet: Accuracy Plot

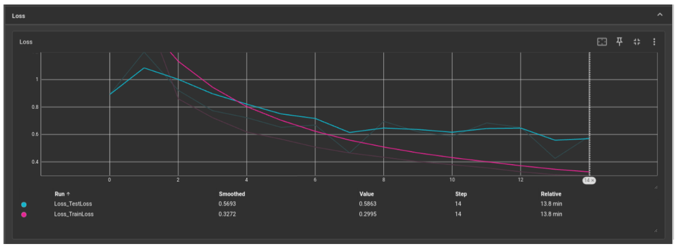


Fig. 28: DenseNet: Loss Plot

$$\text{DenseNet Confusion Matrix of testing data}$$
$$\begin{bmatrix} 722 & 13 & 69 & 64 & 20 & 6 & 12 & 17 & 21 & 56 \\ 6 & 860 & 3 & 7 & 1 & 3 & 14 & 4 & 5 & 97 \\ 34 & 2 & 668 & 73 & 82 & 72 & 41 & 17 & 4 & 7 \\ 5 & 1 & 38 & 695 & 50 & 155 & 23 & 24 & 0 & 9 \\ 5 & 0 & 31 & 63 & 801 & 32 & 16 & 46 & 3 & 3 \\ 2 & 0 & 15 & 139 & 33 & 770 & 5 & 30 & 0 & 6 \\ 2 & 1 & 32 & 88 & 51 & 22 & 793 & 6 & 0 & 5 \\ 3 & 1 & 12 & 45 & 44 & 52 & 4 & 824 & 1 & 14 \\ 66 & 20 & 24 & 36 & 6 & 12 & 11 & 7 & 756 & 62 \\ 6 & 33 & 7 & 13 & 1 & 2 & 2 & 1 & 1 & 934 \end{bmatrix}$$

$$\text{DenseNet Confusion Matrix of training data}$$
$$\begin{bmatrix} 3931 & 25 & 250 & 253 & 79 & 63 & 43 & 100 & 37 & 219 \\ 10 & 4550 & 10 & 28 & 3 & 11 & 28 & 8 & 10 & 342 \\ 106 & 2 & 3868 & 282 & 285 & 248 & 127 & 62 & 5 & 15 \\ 9 & 0 & 90 & 3987 & 138 & 616 & 68 & 70 & 2 & 20 \\ 20 & 0 & 79 & 212 & 4366 & 134 & 32 & 141 & 6 & 10 \\ 3 & 0 & 48 & 539 & 152 & 4124 & 26 & 101 & 0 & 7 \\ 4 & 1 & 126 & 286 & 245 & 119 & 4186 & 14 & 2 & 17 \\ 0 & 2 & 36 & 158 & 108 & 192 & 6 & 4473 & 5 & 20 \\ 248 & 65 & 75 & 198 & 16 & 31 & 86 & 16 & 3992 & 273 \\ 13 & 36 & 4 & 33 & 5 & 10 & 8 & 14 & 3 & 4874 \end{bmatrix}$$

due to computation limitations. Therefore, increasing the number of epochs may help improve the accuracy. For comparison, see below table.

TABLE XI: Model Comparison

Networks	Metrics		
	Parameter Count	**Training Accuracy**	**Testing Accuracy**
Basic Network	545,226	86.406%	74.27%
Improved Network	545,226	96.826%	68.73%
ResNet	697,738	93.828%	74.19%
ResNeXt	23,019,146	94.854%	71.77%
DenseNet	105,214	84.854%	78.23%

References

[1] P. Arbeláez, M. Maire, C. Fowlkes and J. Malik, ”Contour Detection and Hierarchical Image Segmentation,” in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 33, no. 5, pp. 898-916, May 2011, doi: 10.1109/TPAMI.2010.161.

[2] J. Canny, ”A Computational Approach to Edge Detection,” in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-8, no. 6, pp. 679-698, Nov. 1986, doi: 10.1109/TPAMI.1986.4767851.

[3] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. ArXiv. /abs/1512.03385

[4] Xie, S., Girshick, R., Dollár, P., Tu, Z., & He, K. (2016). Aggregated Residual Transformations for Deep Neural Networks. ArXiv. /abs/1611.05431

[5] Huang, G., Liu, Z., & Weinberger, K. Q. (2016). Densely Connected Convolutional Networks. ArXiv. /abs/1608.06993