RBE/CS 549 Computer Vision

HW0 - ALOHOMORA

Puneet Shetty
Robotics Engineering Department
Worcester Polytechnic Institute

Abstract

There are two components to this assignment: A) "Shake my Boundary," in which gradients, Sobel and Canny Baselines, texture, brightness, and color maps are computed to provide a probability-based edge identification method. B) "Deep Dive on Deep Learning," in which objects from the CIFAR10 BSDS500 dataset are classified by comparing several deep learning architectures.Here, I've created a boundary detection method called pb (probability of boundary), which looks for borders by analyzing data on brightness, color, and texture at various sizes. A per-pixel probability of boundary is the result. Additionally, I have used the CIFAR-10 dataset to train and evaluate a variety of models, including ResNet, ResNext, and DenseNet.

Index Terms—Edge Detection, Sobel, Canny, CIFAR10, BSDS500, ResNet, DenseNet, ResNeXt, Edge Detection, Sobel, Canny, CIFAR10, Probability-based edge detection, Convolutional Neural Networks

PHASE 1: SHAKE MY BOUNDARY

The problem statement of boundary detection is intriguing. We determine the boundary given an image by observing the way an object changes into another. Even though boundary detection appears simple to humans, boundary or edge recognition from a single image is challenging to accomplish. To obtain edges, the majority of current approaches only alter the image's intensities. In this assignment, we employ three distinct filters and a probability-based edge detection method that takes into account three distinct parameters: texture, brightness, and color changes. Oriented Derivative of Gaussian, Leung-Malik (LM), Gabor Filter-banks. The presented pb boundary detecting algorithm is implemented in this part. It employs both the intensity discontinuities and the texture and color information found in the image, which sets it apart from the widely used classical CV approaches found in Sobel and Canny Filters. The sections that follow walk you through these four steps:

1. Filter Banks
2. The Texture, Brightness, and Color maps
3. Gradients of texture, brightness, and color (Tg, Bg, and Cg)
4. Pb-lite production in conjunction with baselines

We'll be working our way through each of the previously listed steps.

Figure 1: *Original Images*

Filter Banks

In this section, the initial phase of the suggested Pb light border detection is to filter out the pictures using the collection of filter banks. For this reason,

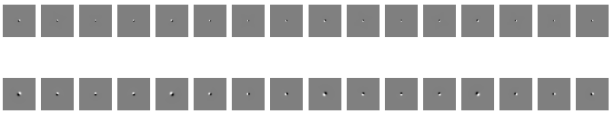

Figure 2: *DoG Filter Bank*

we have three distinct sets of filter banks below. By applying these filters on photos that include these banks, we may develop the low-level characteristics that make up texture.

1. *Oriented Derivative of Gaussian Filter Bank*
   Oriented DoG Filter Bank are created by convolving a simple Sobel Filter and a Gaussian kernel and further rotating results. Here I have total 2 scales and 16 different orientations which gives us 32 filters. Fig. 2 shows these filters.
2. *Leung-Malik (LM) Filter Bank*
   Leung-malik filter-banks are formed by multi-scale, multi- orientation filter-bank consisting 48 different filters. There are three different types of Leung-malik filters. In first type of filters, first and second derivative filters occur at the first 3 scales with an elongation factor of 3 ($\sigma_y = 3\sigma_x$). In second type of filter, Leung-malik small filters occurs at basic scales,$\sigma = 1, \sqrt{2}, 2, 2\sqrt{2}$. The third type of filter, Leung-malik large filters occurs at basic scales $\sigma = \sqrt{2}, 2, 2\sqrt{2}, 4$ Leung-Malik filters are obtained by combining 4 different combinations of filters: 1) First Derivative of Gaussian Filter 2)Second Derivative of Gaussian Filter 3) Laplacian of Gaussian Filter 4) Gaussian Filter. Fig. 3 and Fig. 4 show the LM filters.
3. *Gabor Filter Bank*
   Gabor filters are designed on the filters in the human visual system. A Gabor filter

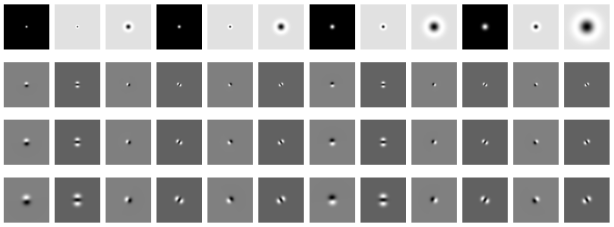

Figure 3: *LM - Small Filter*

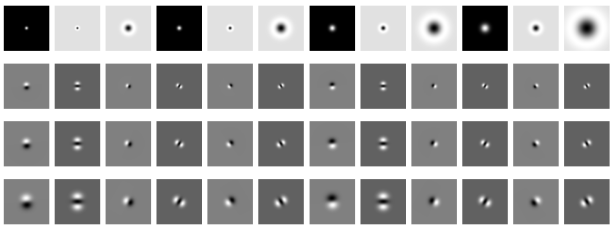

Figure 4: *LM - Large Filter*

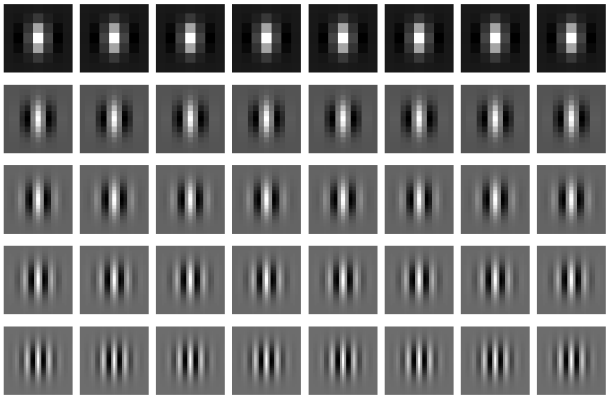

Figure 5: *Gabor Filter*

Figure 6: *a) DoG b) LMS c) LML d) Gabor filters on an Image*

is a gaussian kernel function modulated by a sinusoidal plane wave. The Gabot filter has scale of 8, 16 and 24, it also has 8 orientations with 2, 4, 6 frequencies and kernel size of 49. Fig. 5 shows these filters

Texton Brightness and Color Maps

1. Texton Map
   A vector of filter responses centered on each pixel is produced when an input picture is filtered using each member of the filter bank. It is possible to think of a distribution of these N-dimensional filter responses as encoding texture qualities. Here, we have made this format simpler by substituting a discrete Texton ID for each Ndimensional vector. To do this, we use K-means clustering to group the filter responses at each pixel in the picture into K Textons. Rather of a vector of high-dimensional, real-valued filter responses, each pixel is then represented by a one-dimensional, discrete cluster ID. After that, this is shown as a single channel picture with values between [1,2,3,...,K].K is equal to 64. Subsequently, it was noted that the pictures' filtered output had low intensity values, which led to inadequate grouping. Thus, normalizing the output picture in the 0-255 range after filtering enhanced the clustering.In order to cluster all of the answers at all pixels in the picture for K textons using Kmeans, we first capture the texture changes in the image and then use the texture variations to create an N-dimensional vector. This allows us to locate the Texton Map. Fig. 7 shows the Texton Maps of the Images

2. Brightness Map
   The only steps that is required to create a brightness map is to record the variations in image brightness. Once more, we use kmeans clustering to group the brightness values (the grayscale equivalent of the color image) into a predetermined number of clusters (K=16). The clustered result is referred to as the brightness map B. By measuring the brightness change in the image, we can create a brightness map. Then, we can use Kmeans clustering to cluster the brightness values for the grayscale counterpart of a color image by selecting a set of cluster bins. Fig. 8 shows these maps.

3. Color Map
   Retaining the image's color variations or chrominance content is the idea behind the color map. Here, again we cluster the RGB color values using kmeans clustering into a given number of clusters (K=16). The clustered output is referred to as the color map C..We find Color Map by collecting color changes or chrominance content in the image and cluster the color values (3 values per pixel (RGB color channels)) using Kmeans clustering by choosing a set of cluster bins.

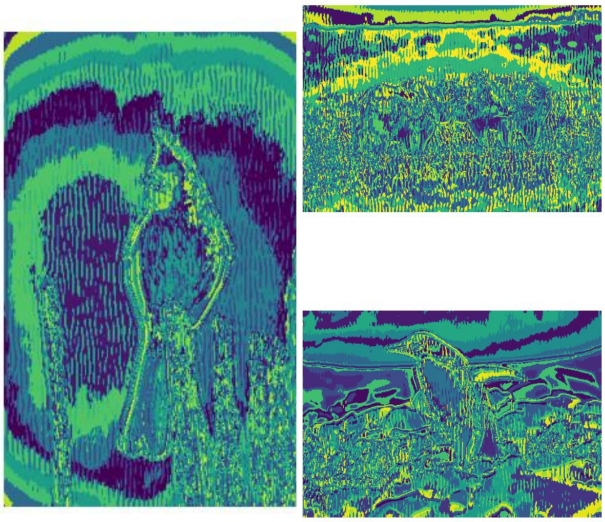

Figure 7: *Texton Map of the Images*

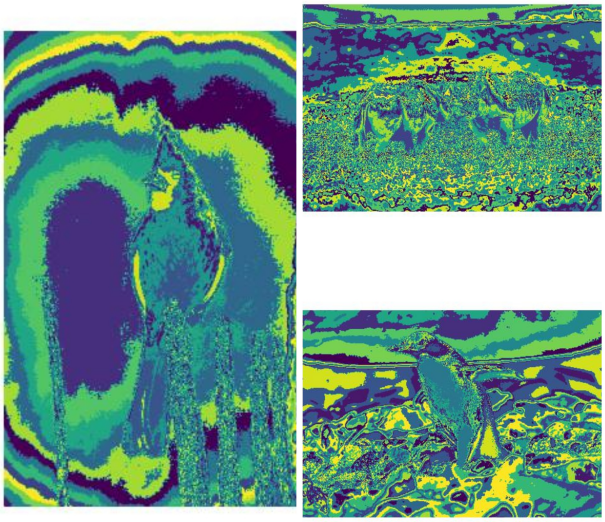

Figure 8: *Brightness Map of Images*

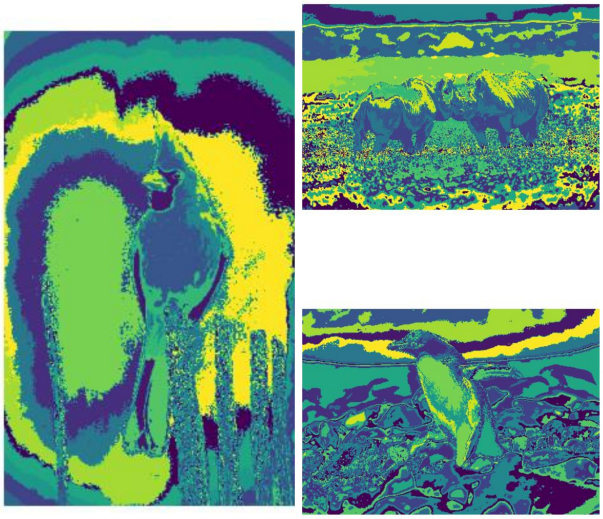

Figure 9: *Color Maps of the Images*

Texton Brightness and Color Gradients

Gradients of texture, brightness, and color indicate how much the distributions of texture, color, and brightness change at each pixel. We must calculate the differences in values across various forms and sizes in order to determine Tg, Bg, and Cg. Half-disk masks are a very effective way to accomplish this.

1. Half-Disc Masks

   The half-disc masks are simple binary images of half-discs. Using a filtering process, we can calculate the chi-square distances considerably faster than if we were to loop over every pixel neighborhood and aggregate counts for histograms. These masks are quite easy to construct. Fig. 10 displays the set of masks utilized in this work, which have three scales and eight orientations. Half-Disc Masks are binary picture pairings of half-discs that are constrained by an equation of circles to either x and y or both within a specific range and range of angles. By comparing the distributions of left/right half-disc pairs—opposing orientations of filters at the same scale—centered at a pixel, we are able to determine Tg, Bg, and Cg. The gradient ought to be modest if the distributions are similar. The gradient ought to be substantial if the distributions are not identical. We will ultimately obtain a set of local gradient measurements that encode the rate at which the texture or brightness distributions are changing at various scales and angles be-

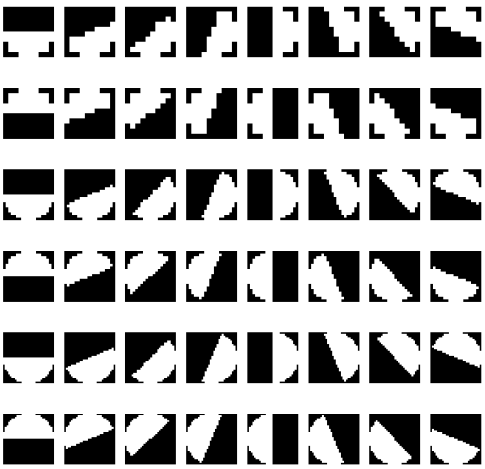

Figure 10: *Half Disk Masks*

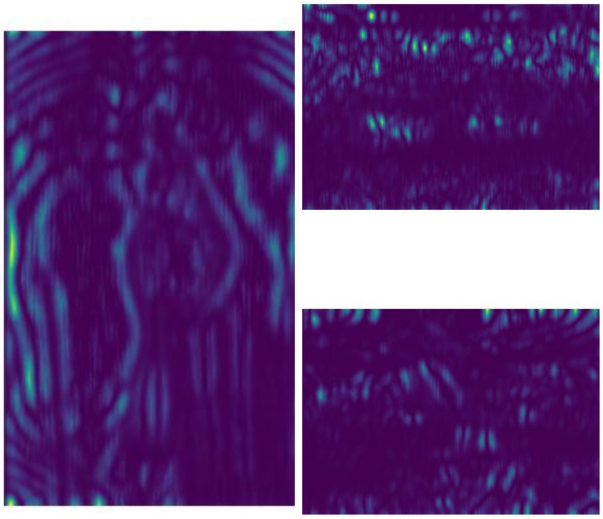

Figure 11: *Texton Gradient of the Images*

cause our half-discs cover many orientations and scales. We will use the chi-square measure to compare the distributions of texton, brightness, and color. When comparing two histograms, one commonly used statistic is the chi-square distance.

2. K-means clustering
   By minimizing inertia or the sum of squares inside a cluster, the K-means algorithm attempts to organize samples into groups of equal variance in order to cluster data. A set of N samples X is divided into K distinct clusters C using the Kmeans method, each of which is characterized by the mean ui of the samples within it. In order to create new centroids for each cluster, we first initialize the number of clusters, randomly initialize the centroid inside each cluster, then assign each point to the closest centroid until the centroid positions stay constant and are not influenced by additional iterations.
3. Chi-square Distance
   A statistical technique utilized in numerous applications, such as similar image retrieval, picture texture, and feature extraction, is the chi-square distance. It measures the similarity between two feature matrices (h, g). Because of its distributional equivalency property, it guarantees the invariance of the distances between rows and columns. Chi-sqaure distance is used to compare each map with certain bins against a half disk filter bank in order to determine the different gradient values.
   $\mathcal{X}^2 = \frac{1}{2}\sum_{i=1}^{K}\frac{g_i - h_i{}^2}{g_i + h_i}$

Probability based detection

Finally, we use chi square distances to the combined filter data to create texture brightness and color gradients. We utilize a weighted sum over Sobel and Canny baseline images for the photos to get the final edge from these gradients. The weighted total of gradients over various baselines is the final result. It is discovered that by using Sobel and Canny edge detectors, we are able to locate the edges of every object and variation present in the image, despite the fact that many approaches simply use Sobel or Canny edge detectors to find edges in an image. This is also included in many open-source online packages. We employ a rigorous technique in this assignment to apply several filter operations on the image and ultimately detect the edges of specific objects in the image.

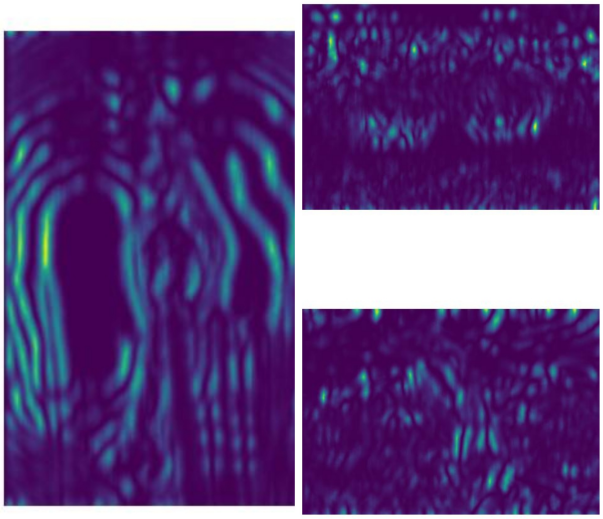

Figure 12: *Brightness Gradient of the Images*

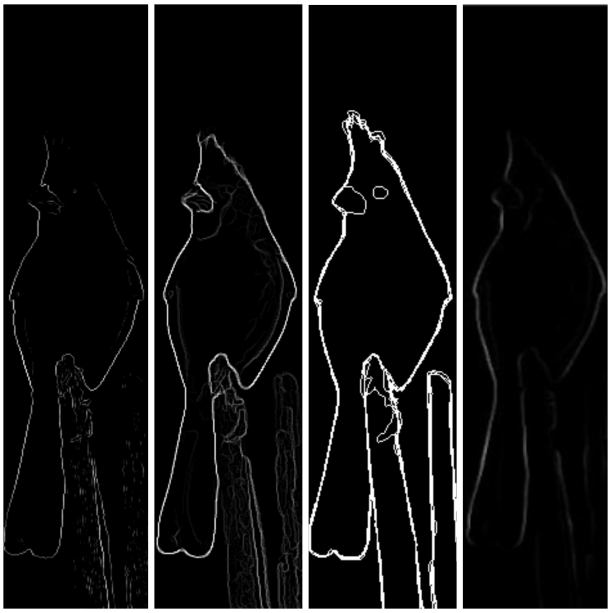

Figure 14: *a) Sobel b) Canny c) Ground Truth d) Final Result of Image*

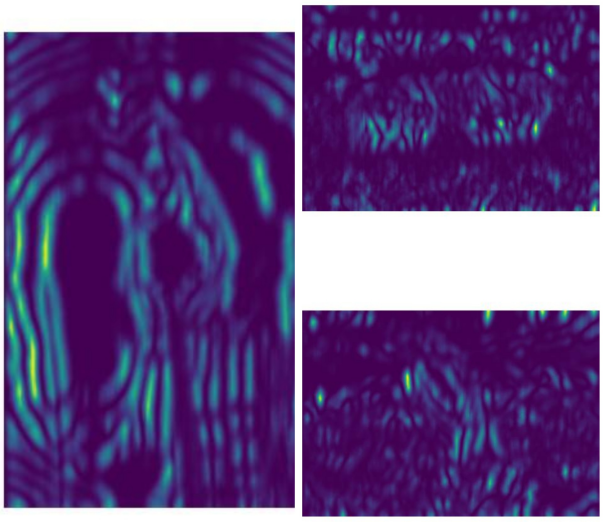

Figure 13: *Color Gradient of the Images*

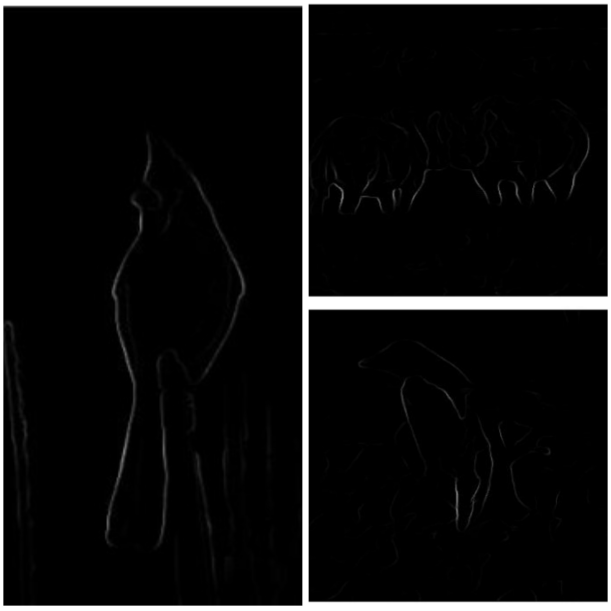

Figure 15: *Final Result of PB-Lite Detection on Images*

PHASE 2 : DEEP DIVE INTO DEEP LEARNING

In order to analyze the training and testing accuracy and loss values for training with the CIFAR-10 dataset, which comprises of 60000 32*32 color images in 10 classes with 6000 images per class, we compare many neural network designs by changing the number of parameters. There are 50,000 and 10,000 photos in the training and testing data sets, respectively. Initially, I trained for minimum epochs using a convolution neural network. For the computation of loss and learning rate of 1e-5 for every network architecture, I have employed the ADAM optimizer and the Cross Entropy function. I have used Google Colabotory's Tesla4 GPU to train and test the models. I have not utilized any standardization or normalization for the basic CNN architecture, and I have used ResNet18, DenseNet, and ResNeXt for all additional versions.

Neural Networks

1. First Neural Network

   I tried to implement the simple neural network. There were a few bugs in the starter code, but after debugging them I was able to implement the most basic Neural Network following the instructions in the assignment. The architecture for my network is shown in the figure. I used the Adam Optimizer with a learning rate of 0.001, and let the network train for 16 Epochs with a batch size of 10. I also applied MaxPooling to my layers. Furthermore, I used CrossEntropy loss function. After each epoch I calculate the mean loss and accuracy and plotted them accordingly. The number of parameters in the network are 35712. The test set was able to predict the image with an accuracy of 19.82%. This is understandable since I just simply wanted to train a neural network without any complications. The confusion matrix is shown in figure 16.

2. Improved Neural Network

   The number of epochs was kept at 16 and the mini batch size was increased to 64 in order to improve the accuracy of the train set. Furthermore, after each convolution layer, batch normalizing layers have been added. The accuracy of the test set 65.85 percent. The Adam optimizer with a 1e-3 learning rate was employed. There were 12,628 parameters in the model. Fig. 19 offer the confusion matrix for dataset

```
[ 54   0  93  78 492   1   2   0 280   0] (0)
[  7   0  24  39 829   2  10   0  89   0] (1)
[  6   0 143  89 720   4   3   0  35   0] (2)
[  0   0  29 241 720   7   0   0   3   0] (3)
[  0   0   6  21 965   0   0   0   8   0] (4)
[  0   0  46 253 672  22   1   0   6   0] (5)
[  1   0  25  66 889   0  17   0   2   0] (6)
[  0   0  20  65 897   3   0  11   4   0] (7)
[  3   0  12  45 426   0   9   0 505   0] (8)
[  2   0  30  55 820   3   1   1  88   0] (9)
  (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
```

Figure 16: *Basic Block Confusion Matrix*

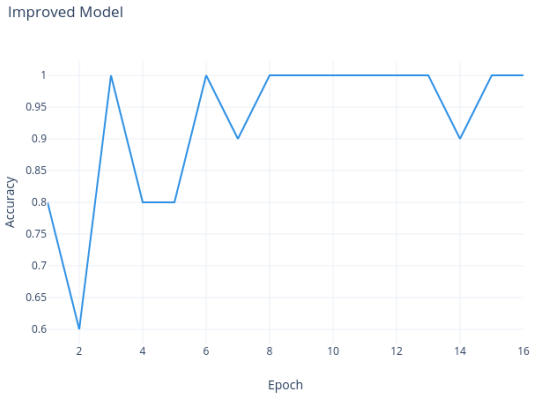

Figure 17: *Accuracy of Basic Net*

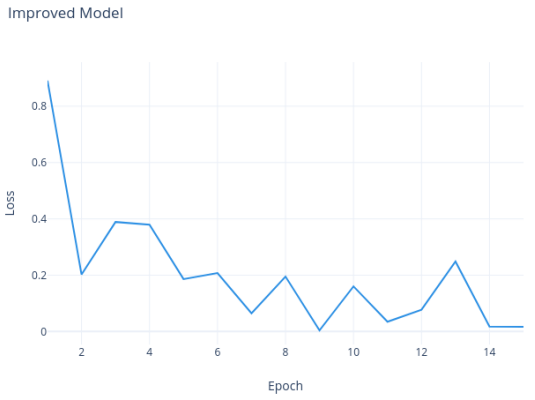

Figure 18: *Loss of Basic Net*

```
[670  19  62  26  38  11  10  29 102  33] (0)
[ 19 771   8  28   9  10  14  11  40  90] (1)
[ 51   7 498  73 135  96  47  61  22  10] (2)
[ 19  13  58 464 102 186  57  66  16  19] (3)
[ 18   9  50  67 656  57  40  91   8   4] (4)
[ 13   5  40 144  68 599  21  95   8   7] (5)
[  8  10  50  84 104  52 663  12  10   7] (6)
[  8   4  33  31  74  67   6 762   2  13] (7)
[ 50  37  16  17  18  12   6  18 798  28] (8)
[ 39 107  13  23  13  24   3  39  39 700] (9)
  (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
```

Figure 19: *Improved Block Confusion Matrix*

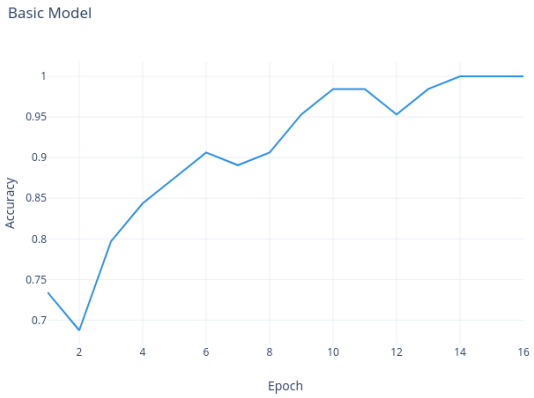

Figure 20: *Accuracy of Improved Net*

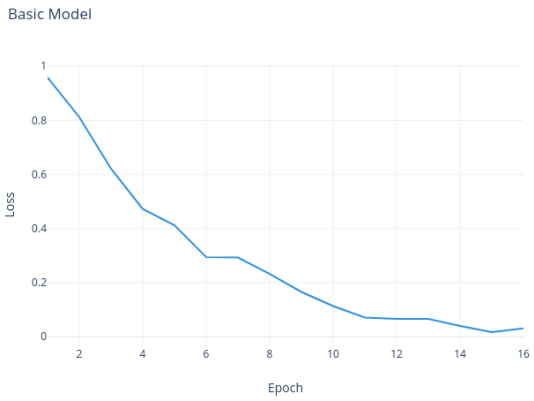

Figure 21: *Loss of Improved Net*

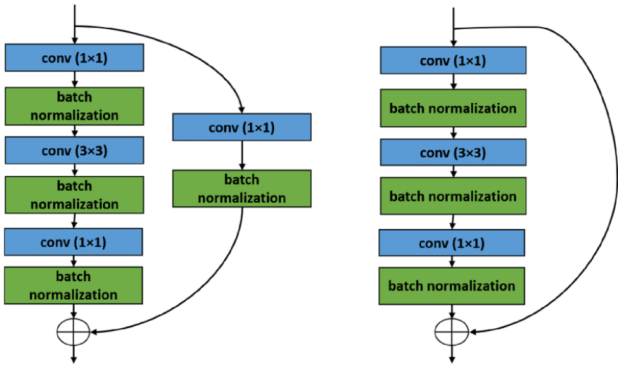

Figure 22: *Basic Block of ResNet used*

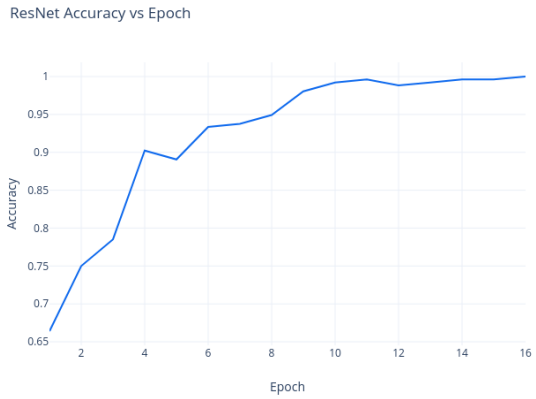

Figure 23: *ResNet Accuracy*

3. ResNet

   The input first goes through the following steps: Max Pooling, Batch Norm, Relu activation, and Convolution. The main block then appears, and it is repeated four times. The foundation of the ResNet architecture is this block. It is better explained in Figure 22. Thus, a 1 * 1 convolution occurs initially, then a 3 * 3, and finally a 1 * 1. BatchNorm and Relu layers come after these convolution layers. After that, the block's origin input and the resultant output are combined, and Relu is used as the input for the following block. The four-blocks have two of these blocks shown in in Figure 22, respectively.

4. ResNeXt

   Figure 26 shows the implemented ResNeXt architecture. The architecture of ResNeXt and ResNet are extremely similar. The first three layers inside the ResNeXt block are the standard Convolution 1*1, Batch-Norm, and Relu. A characteristic called cardinality, added to the second convolution layer (3*3), groups the convolution layer with 32 number of similar layers. After that, the out-

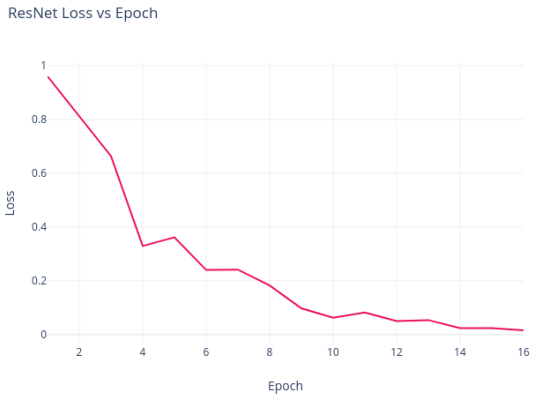

Figure 24: *ResNet Loss*

```
[839  15  40  25   9   9  11   7  31  14] (0)
[ 12 875  15   5   5   5  12   4  22  45] (1)
[ 56   6 682  69  52  44  60  23   3   5] (2)
[ 27  13  74 577  39 130 102  26   5   7] (3)
[ 26   5  96  70 657  18  70  51   6   1] (4)
[ 16   4  55 212  34 585  44  43   4   3] (5)
[  7  11  39  39  16  15 862   6   4   1] (6)
[ 20   3  28  56  39  44  17 783   6   4] (7)
[ 84  31  24  21   7   7   7   1 802  16] (8)
[ 48 124  20  18   4  12   8  15  28 723] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
```

Figure 25: *ResNet Confusion Matrix*

put is added to a downsampled input and passes through layers of BatchNorm, Relu, Convolution 1*1, and BatchNorm before passing via Relu activation. These ResNeXt blocks are contained in each of the four layer blocks.

5. DenseNet

   The two sorts of blocks are as follows: the dense block, which consists of several bottleneck levels. A 1*1 and a 3*3 convolutional layer make up the majority of each bottleneck layer. Concatenated to the input

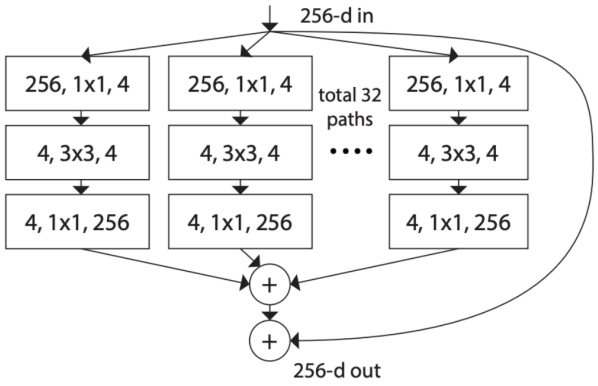

Figure 26: *ResNeXt architecture followed*

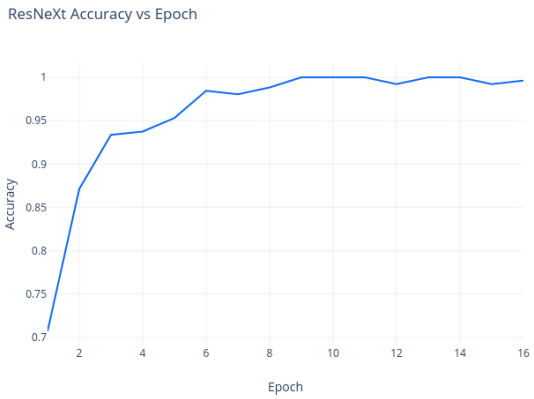

Figure 27: *ResNeXt Accuracy*

Figure 28: *ResNeXt Loss*

```
[789  21  27   8  13  16   7  22  43  54] (0)
[  3 923   0   1   0   2   4   3   4  60] (1)
[ 33   4 667  25  43 102  78  37   6   5] (2)
[  9   4  16 563  30 247  74  40   9   8] (3)
[  7   3  17  19 828  57  32  33   2   2] (4)
[  1   2   9  36  22 885  11  31   1   2] (5)
[  5   4   9   5   8  22 935  11   0   1] (6)
[  3   0   2  11  23  59   5 894   0   3] (7)
[ 23  21   6   7   2  11   7   5 893  25] (8)
[  7  36   2   1   0   7   1  15  16 915] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
```

Figure 29: *ResNeXt Confusion Matrix*

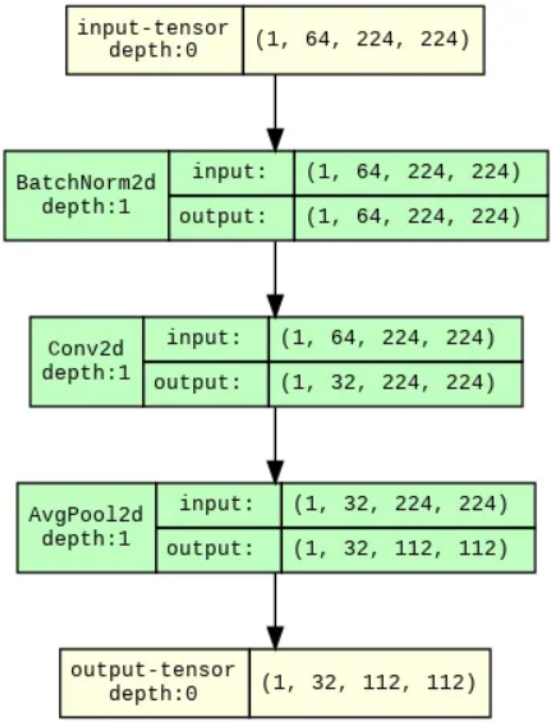

Figure 30: *Transition layer of DenseNet*

Table 1: *Comparison of all models*

Model	ResNet	ResNeXt	DenseNet
Optimizer	Adam	Adam	Adam
Learning rate	1e-3	1e-3	1e-3
Batch Size	256	256	256
Parameters	11689512	234355586	70464673
Train Accuracy	100%	99.6%	98.82%
Test Accuracy	73.85%	82.92%	76.69%
Loss	1.56%	1.94%	3.57%

of the layer is the output of the second convolution. Within the dense blocks of this architecture, there are 6, 12, 24, and 16 bottleneck layers. Also, there is a transition block (Figure 30), which is mostly formed of 1*1 convolution, and an average pool, for down- sampling between consecutive dense blocks.

Final Analysis

We may conclude from the final analysis that the ResNeXT neural network design has the best performance. All things considered, the neural network architectures that we have put in place—ResNet, ResNeXt, and DenseNet—really increase system accuracy and decrease loss.

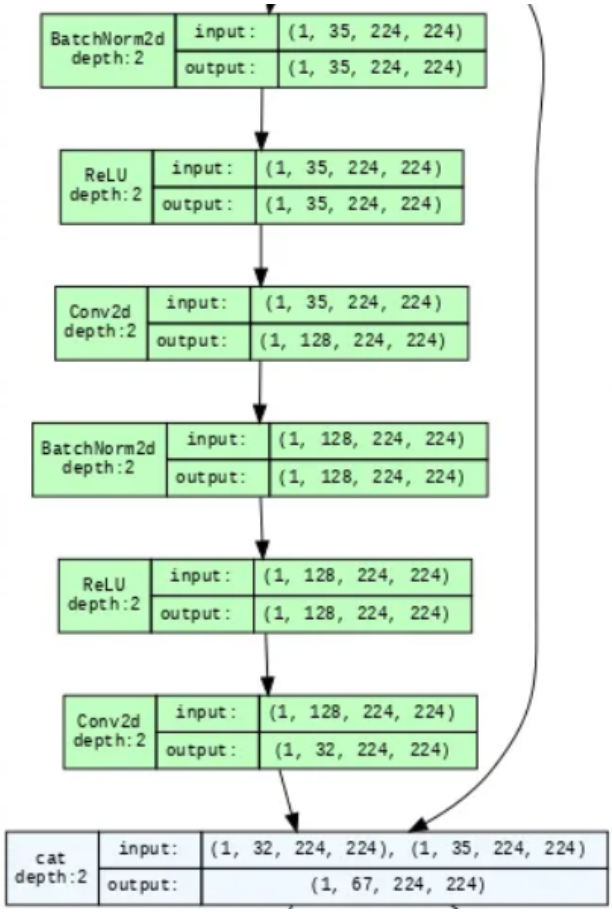

Figure 31: *DenseLayer architecture followed*

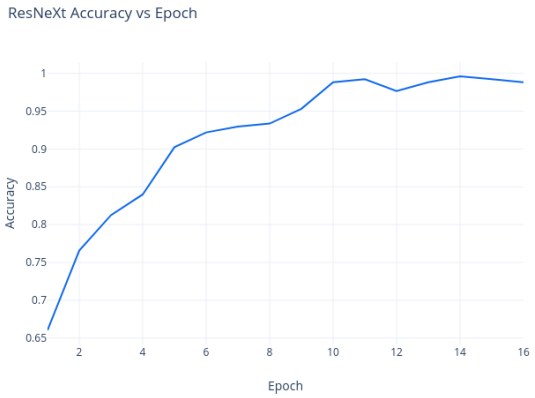

Figure 32: *DenseNet Accuracy*

Figure 33: *DenseNet Loss*

```
[766  22  43  25  37   1  12  22  47  25] (0)
[  5 885   2   3   1   1  12   7  29  55] (1)
[ 39   3 605  54 101  45  93  35  15  10] (2)
[ 12  12  37 596  85 101  72  56  12  17] (3)
[ 10   2  30  44 794  12  58  43   7   0] (4)
[  6   5  28 170  79 584  42  69   5  12] (5)
[  5   1  11  34  33  10 885   9   7   5] (6)
[  2   3   8  28  62  22   7 860   2   6] (7)
[ 44  31   8   8  17   0   7   7 861  17] (8)
[ 17  93   7   7   8   3   7   6  19 833] (9)
  (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
```

Figure 34: *DenseNet Confusion Matrix*

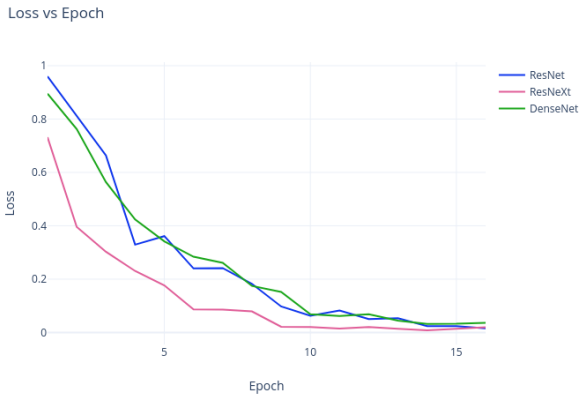

Figure 36: *Losses Compared*

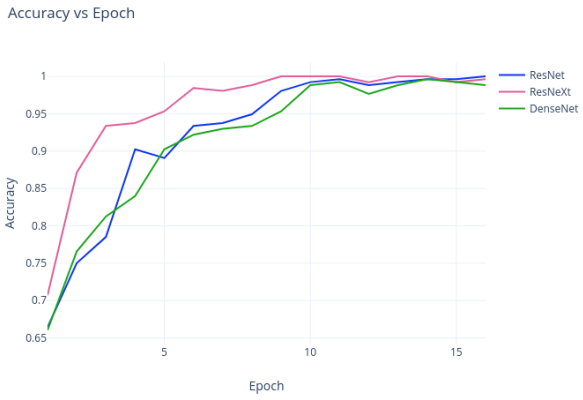

Figure 35: *Accuracies Compared*