CMSC733: Homework 0 - Alohomora!

Hossein Souri
Center for Automation Research, UMIACS
University of Maryland, College Park
Email: hsouri@umiacs.umd.edu
Use 3 late days

I. Introduction

Boundary detection and image classification are two well known computer vision problems. The challenging part of the boundary detection is when we are doing the boundary detection from a single image. Is this case we cannot use most of the recent deep learning methods. I wish you the best of success. In first part of this work I will use the most recent pb (probability of boundary) boundary detection algorithm. In the second part, I will implement multiple neural network architectures and comparing them on various criterion like number of parameters for the image classification problem.

II. Phase 1: Shake My boundary

In this section, I will develop a simplified version of pb using the texture, brightness and color information. To do this, we need to implement different filter banks.

A. Filter bank implementation

I'm implementing three kind of filter banks; oriented derivative of Gaussian (DoG) filter, Leung-Malik filter, and Gabor filter.

1) oriented derivative of Gaussian (DoG): These filters can be created by convolving a simple Sobel filter and a Gaussian kernel and then rotating the result.

Illustration of this filter bank using two scales and 5 rotations can be seen in Fig 1.

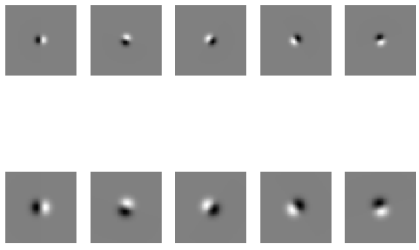

Fig. 1: DoG filters generated for 2 scales and 5 orientation

2) Leung-Malik Filters: The Leung-Malik filters or LM filters are a set of multi scale, multi orientation filter bank with 48 filters. It consists of first and second order derivatives of Gaussians, Laplacian of Gaussian (LOG), and Gaussians.

Illustration of the LM Large filter bank can be seen in Fig 2.

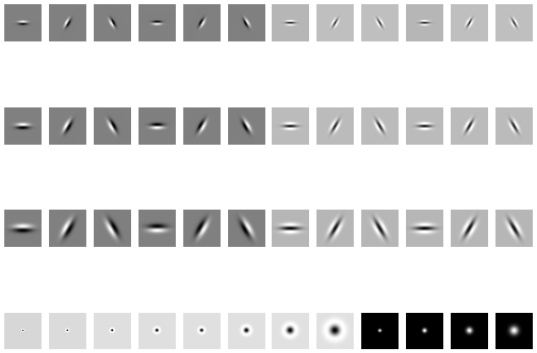

Fig. 2: Large Leung-Malik filter

3) Gabor Filters: A gabor filter is a gaussian kernel function modulated by a sinusoidal plane wave. To implement this filter bank I use the parameters: $\lambda = [8, 10, 12]$, $, \sigma = [6, 8, 14]$, $\psi = 0$, and $\gamma = 0.075$. Illustration of this filter bank using 3 scales and 3 rotations can be seen in Fig 3.

B. Texton, brightness, color map computation

In the next step, we implement texton, brightness, and color map. I generate texton map by convolving the previous filter banks with the images and map them into 64 clusters and taking the average. The concept of the brightness map is as simple as capturing the brightness changes in the image and cluster it into 16 groups. Also, The concept of the color map is to capture the color changes. Then cluster it into 16 clusters. Illustration of the generated map are shown in Fig 4-13.

C. Texture, Brightness and Color Gradients

To compute gradients, we first need to implement half-disc masks. An image of these Half-disc masks is shown in

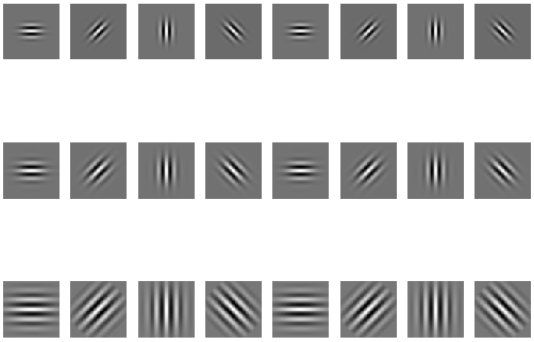

Fig. 3: Gabor filters generated at 3 different scales and 3 orientations

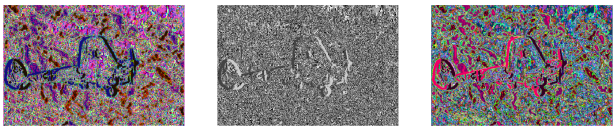

Fig. 4: Texton, Brightness and Color Map for image 1.png

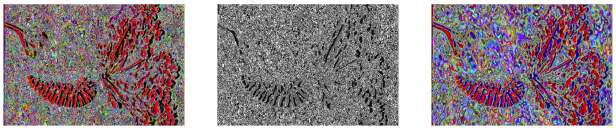

Fig. 5: Texton, Brightness and Color Map for image 2.png

Fig. 6: Texton, Brightness and Color Map for image 3.png

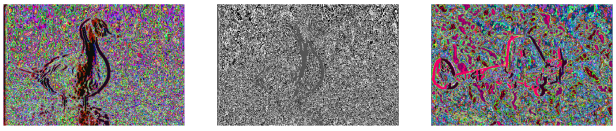

Fig. 7: Texton, Brightness and Color Map for image 4.png

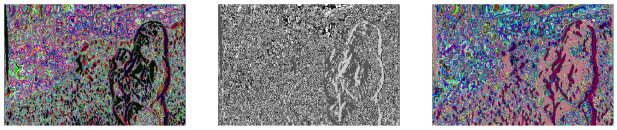

Fig. 8: Texton, Brightness and Color Map for image 5.png

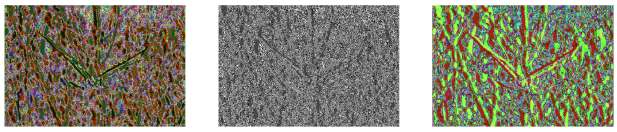

Fig. 9: Texton, Brightness and Color Map for image 6.png

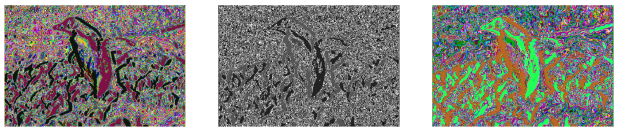

Fig. 10: Texton, Brightness and Color Map for image 7.png

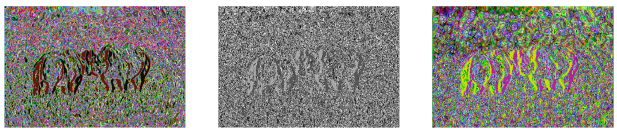

Fig. 11: Texton, Brightness and Color Map for image 8.png

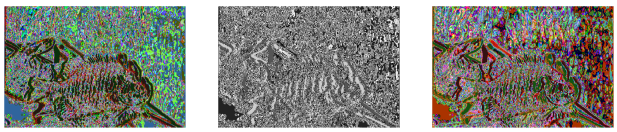

Fig. 12: Texton, Brightness and Color Map for image 9.png

Fig. 13: Texton, Brightness and Color Map for image 10.png

Fig 4. Using these half-disc masks along with the Chi-square distance, we can generate the gradient map for all texture, brightness, and color information of the images. the generated gradients are shown in Fig 15-24.

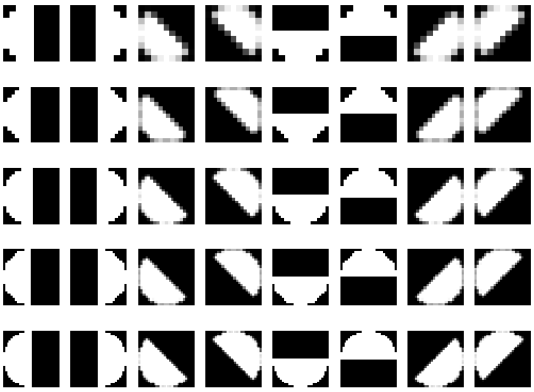

Fig. 14: Half-disc masks

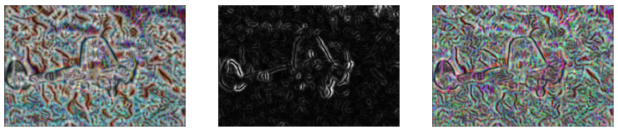

Fig. 15: Texton, Brightness and Color Gradients for image 1.png

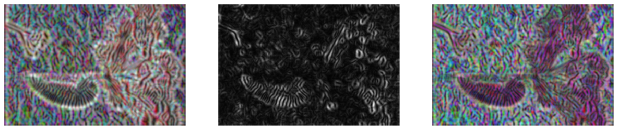

Fig. 16: Texton, Brightness and Color Gradients for image 2.png

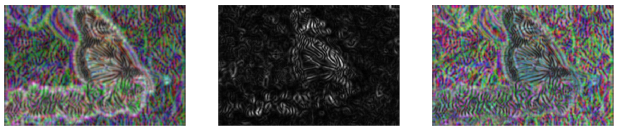

Fig. 17: Texton, Brightness and Color Gradients for image 3.png

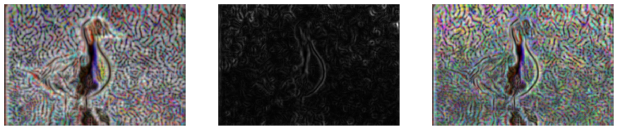

Fig. 18: Texton, Brightness and Color Gradients for image 4.png

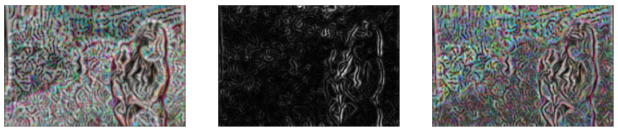

Fig. 19: Texton, Brightness and Color Gradients for image 5.png

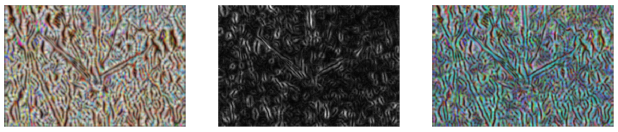

Fig. 20: Texton, Brightness and Color Gradients for image 6.png

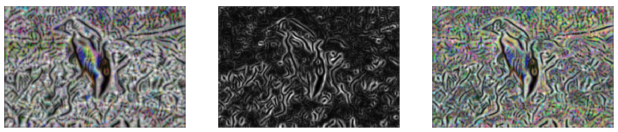

Fig. 21: Texton, Brightness and Color Gradients for image 7.png

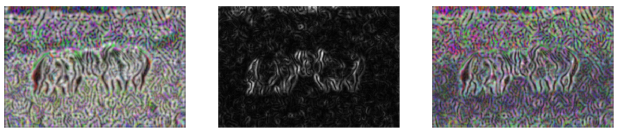

Fig. 22: Texton, Brightness and Color Gradients for image 8.png

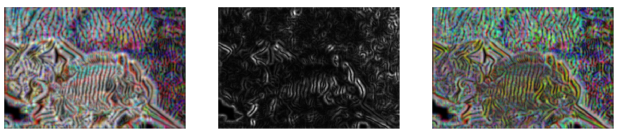

Fig. 23: Texton, Brightness and Color Gradients for image 9.png

Fig. 24: Texton, Brightness and Color Gradients for image 10.png

D. Boundary detection

The final step to get the boundaries is to combine information from the features with Sobel or Canny edge detection by computing the dot product between the average of gradients and the average of Canny ans Sobel. The results are shwon in fig 25-34.

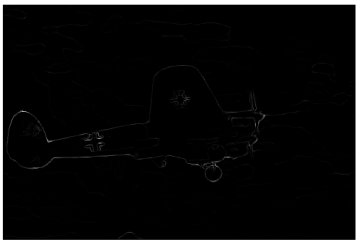

Fig. 25: pb-lite result of image 1

Fig. 26: pb-lite result of image 2

Fig. 27: pb-lite result of image 3

Fig. 28: pb-lite result of image 4

Fig. 29: pb-lite result of image 5

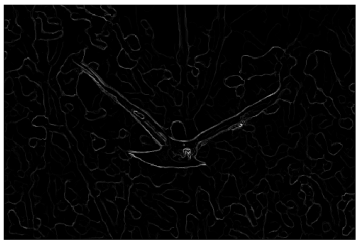

Fig. 30: pb-lite result of image 6

Fig. 31: pb-lite result of image 7

Fig. 32: pb-lite result of image 8

Fig. 33: pb-lite result of image 9

Fig. 34: pb-lite result of image 10

E. Analysis

The result is not as clear as the Canny baseline, however, it's better than the Sobel base line. In some degree, the Pb-lite is better than Canny since it does not have false positive results. We can see a lot of incorrect edges in the Canny which are not presented in the Pb-lite. It should be also mentioned that it's a possible that I don't chose the best filter bank to do it, even though I tried different sizes and filters. you can see Fig to cpmpare Sobel, Canny, and pb-lite results.

III. Deep Dive on Deep Learning

In this section we implement different neural network architectures and try to train them for the image classification. For the image classification I use CIFAR-10 with 50000 training images and 10000 test images.

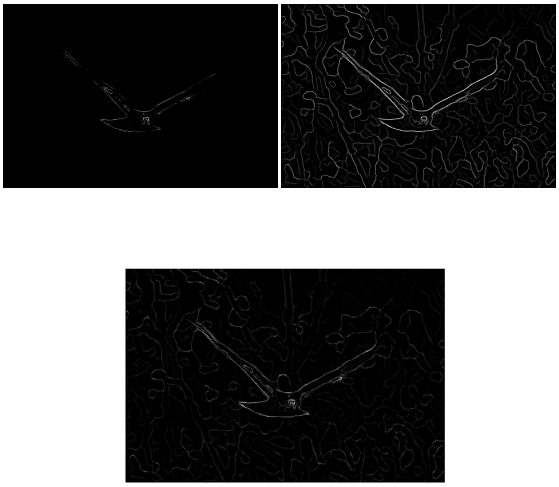

Fig. 35: Sobel, Canny, and pb-lite result of image 6.png

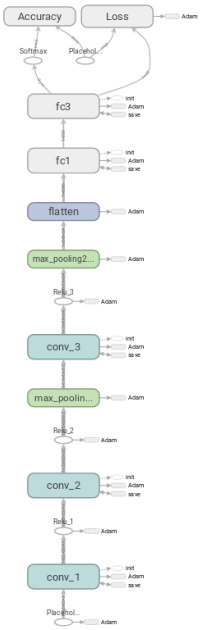

Fig. 36: Simple network architecture

A. My first neural network

In this section I use 3 convolutional layers with Relu activation function. After it there is a max pooling layer and three fully connected layers with total 1075754 number of parameters. The network architecture is shown in Fig 36.

I use Adam optimizer with learning rate 1e-4, batch size is 64, and number of epochs is 24. The train and test accuracy and train loss over epochs is plotted in figure 37. Also, in the same figure we can see the confusion matrix. It can be seen that the model is overfitted so it has a very low accuracy on the test set. The possible next step to improve it is to use data augmentation, batch normalization layer.

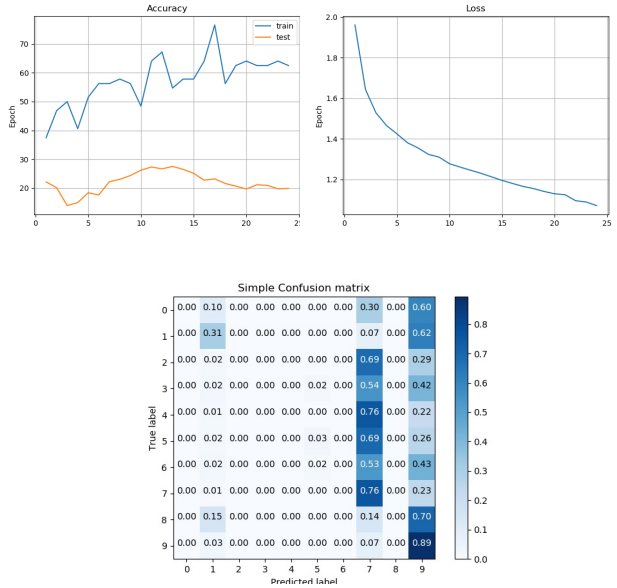

Fig. 37: Accuracy, loss, and confusion matrix of the Simple Network

B. Improving network and training

In this section I use 3 convolutional layers with batch normalization and Relu activation function. After them there is a max pooling layer and three fully connected layers with total 1090506 number of parameters. I also used the standardization and random horizontal flip as data augmentation. The network architecture is shown in Fig 38.

The train and test accuracy and train loss over epochs is plotted in figure 39. Also, in the same figure we can see the confusion matrix. Using the improved network we can see that we have better performance and it's because of using more complex network with batch normalization and data augmentation. Hereafter, I use data augmentation and batch normalization during training.

C. ResNet

I implement a simple model of the ResNet using one convolution layer and three residual block each consisted of three convolution layers. The total number of the parameters in this network is 615946 . Fig 40 shows the architecture of this network. The first convolutional layer is 3 by 3. And the convolutional layers in the residual blocks are 5 by 5.

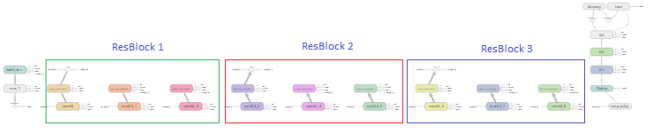

Fig. 40: ResNet architecture

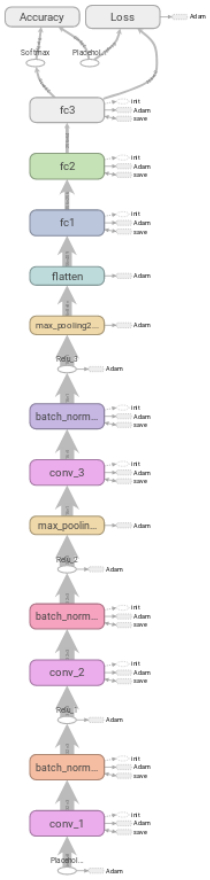

Fig. 38: Improved network architecture

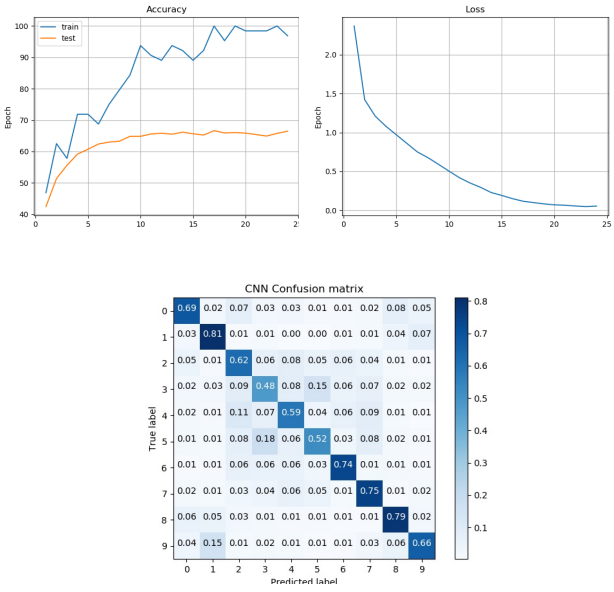

Fig. 39: Accuracy, loss, and confusion matrix of the improved Network

The train and test accuracy and train loss over epochs is plotted in figure 41. Also, in the same figure we can see the confusion matrix. As we can see the result of ResNet is better than the previous model. I will elaborate more on this in the Analysis section.

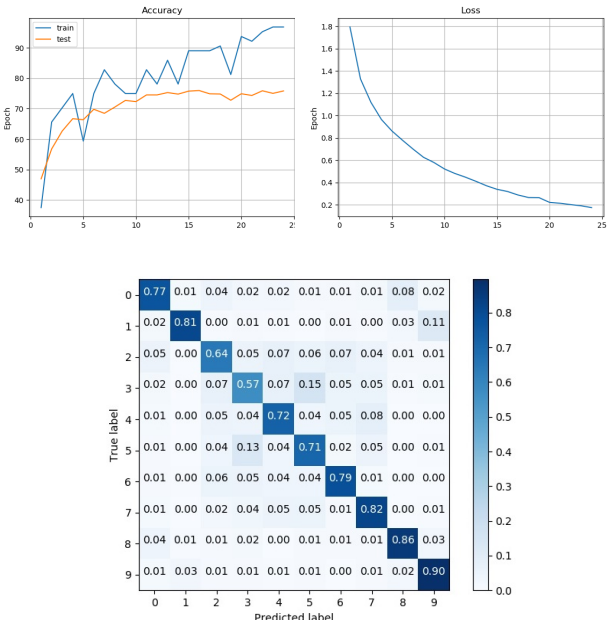

Fig. 41: Accuracy, loss, and confusion matrix of the ResNet Network

D. ResNext

I implement a simple model of the ResNext using one resNext block consisted of 5 residual block and three fully connected layers. The total number of the parameters in this network is 2222442 . Fig 42 shows the architecture of this network.

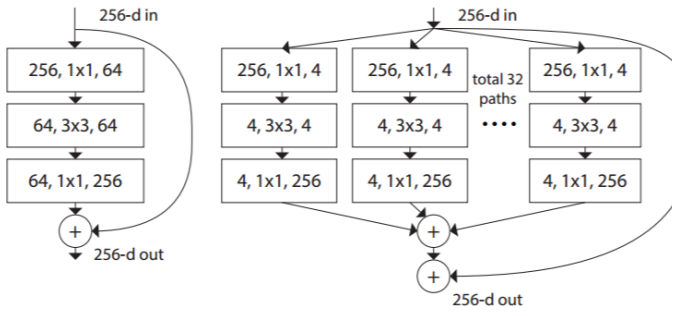

Fig. 42: ResNext architecture

The train and test accuracy and train loss over epochs is plotted in figure 43. Also, in the same figure we can see the confusion matrix.

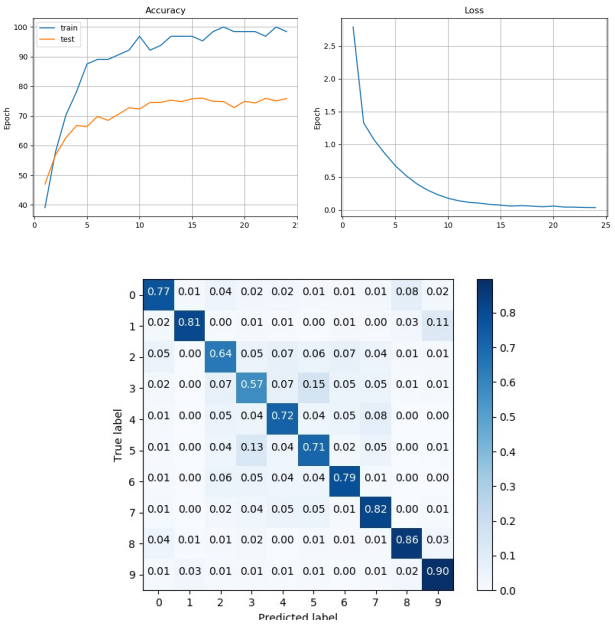

Fig. 43: Accuracy, loss, and confusion matrix of the ResNext Network

E. DenseNet

I implement a simple model of the DenseNet using one resNext block consisted of 5 convolutions layer and three fully connected layers. The total number of the parameters in this network is 3368666 . Fig 44 shows the architecture of this network.

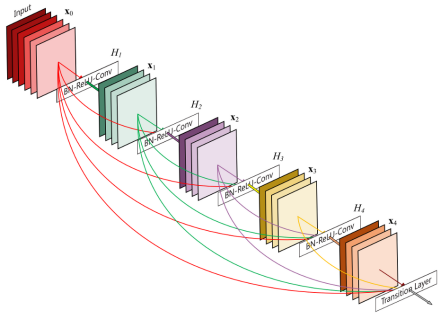

Fig. 44: DenseNext architecture

The train and test accuracy and train loss over epochs is plotted in figure 45. Also, in the same figure we can see the confusion matrix.

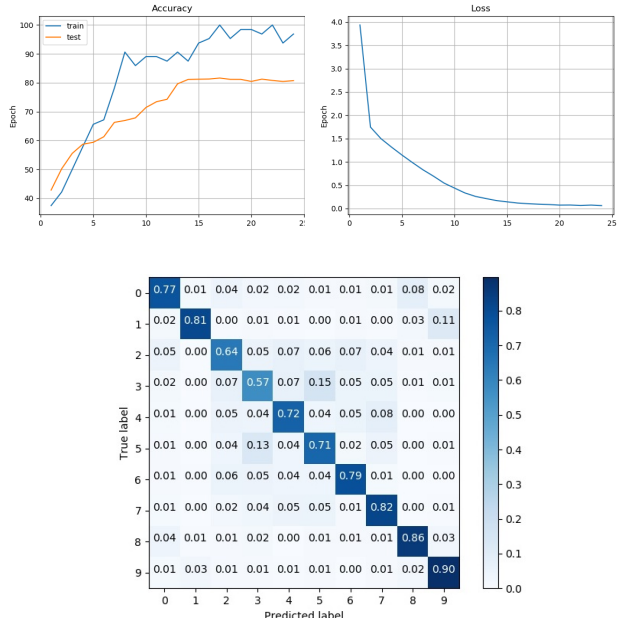

Fig. 45: Accuracy, loss, and confusion matrix of the DenseNext Network

F. Analysis

In Phase 2 I implement different networks and train them for the task of image classification. First, we see that the size of the network is really important because the small networks over fit very soon after a few epochs and their test accuracy is not good. Second, Using batch normalization and suitable data augmentation can boost the network and give better results. It has been show that modern architectures give us better results since they try to resolve the weakness of old networks like gradient vanishing. ResNet address this issue bu using residual blocks. Also, DenseNet concatenate the out put of consecutive layers which prevents gradient vanishing. We can see that DenseNet has the best performance near 80%. However, the number of the parameters is this network is much more that the ResNet. I also, observed that hyperparameters are really important and we should find the suitable hyperparameters by experimenting a lot. In the following table we can see the results of testing on different models.

Model Testing		
Network	Parameters	Test Acc(%)
Simple	1075754	20
Improved	1090506	66.48
ResNet	615946	75.84
ResNext	2222442	74.74
DenseNet	3368666	80.74

TABLE I: Summary of All Models