RBE 549- HomeWork0 :Alohomora

Ankit Mittal
Department of Robotics Engineering
Worcester Polytechnic Institute
Email: amittal@wpi.edu

***Abstract*—(USING 1 LATE DAY) This report outlines the outcomes of two distinct phases in image processing and analysis. Phase I focuses on traditional methods for edge detection, delving into the creation of various filter kernels through mathematical equations and associated techniques. These kernels are then applied to extract edges from images. Phase II shifts to modern approaches in object classification, emphasizing the use of deep Convolutional Neural Networks (CNNs). This phase involves the implementation of various models and provides an in-depth analysis of their performance.**

I. Phase 1: Shaken My Boundary

A. Introduction

In this section, we will create a streamlined version of the pb lite algorithm. This algorithm detects boundaries by analyzing data on brightness, color, and texture at various scales, accommodating different object sizes and image dimensions. The result of this process will be a probability of boundary assigned to each pixel.The initial stage of the process involves applying a series of filter banks to the image. Following this filtration, a texton, along with brightness and color maps, are produced by clustering the responses from these filters. Subsequently, the image gradients are computed to display the variations in texture, brightness, and color at each pixel. In the final step, these gradient outcomes are integrated with the Sobel and Canny baseline methods, employing specific weights, to achieve the desired result.

B. Generating Filters Blanks

To analyze the texture of an image, we will employ three distinct sets of filter banks. After applying these filters to the image, we will create a texton map. This map visually represents the image's texture by grouping together similar responses from the filters.

1) Oriented DoG filters: A simple but effective filter bank is a collection of oriented Derivative of Gaussian (DoG) filters. These filters can be created by convolving a simple Sobel filter and a Gaussian kernel and then rotating the result.

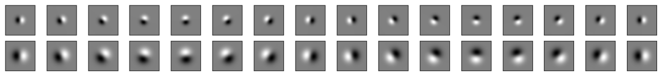


Fig. 1: Generated DoG filter

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 at 6 orientations and 3 scales making a total of 36; 8 Laplacian of Gaussian (LOG) filters; and 4 Gaussians. 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., (σx=σ and σy=3σx).

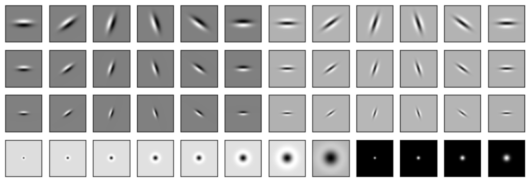


Fig. 2: Generated LM filter

3) Gabor Filters: Gabor Filters are designed based on the filters in the human visual system. A gabor filter is a gaussian kernel function modulated by a sinusoidal plane wave. The Gabor filter is a linear filter used for texture analysis, which essentially means that it analyzes whether there is any specific frequency content in the image in specific directions in a localized region around the point or region of analysis.

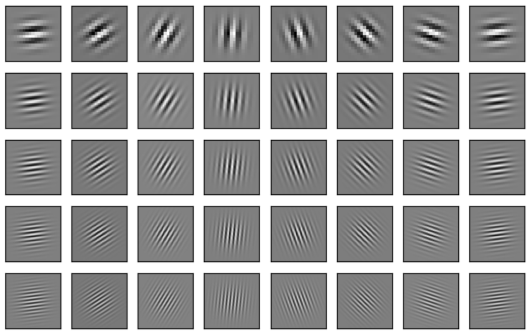
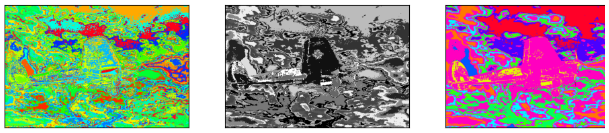
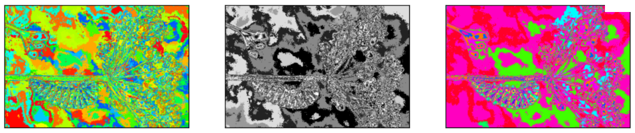


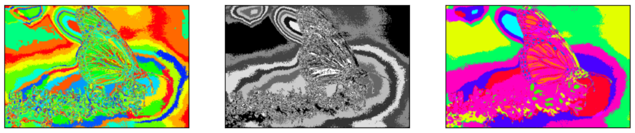
Fig. 3: Generated LM filter

C. Texton, Brightness, and Color Map

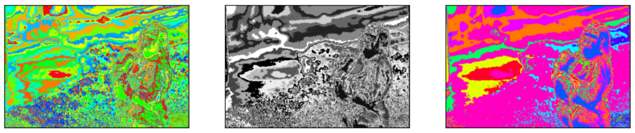
Texton maps are generated by applying a comprehensive set of 120 filters to an image, which results in a collection of output layers. The objective is to classify pixels with si texture characteristics together and assign a unique texto to each. This classification is achieved through the KM clustering technique, where pixels with comparable value grouped into the same cluster. For this process, we use 64 distinct cluster centers to categorize each pixel. Similar to the texton map, brightness maps and color maps are also prod for the same image. The brightness map is derived fron gray-scale version of the original image, while the color is created using the RGB (Red, Green, Blue) version o image.16 clusters are used in case of both brightness and color maps.

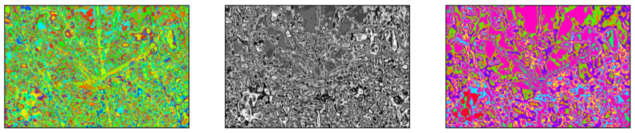

(a) Texton, brightness and color map of image 1


(b) Texton, brightness and color map of image 2

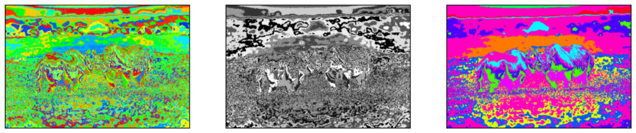

(c) Texton, brightness and color map of image 3

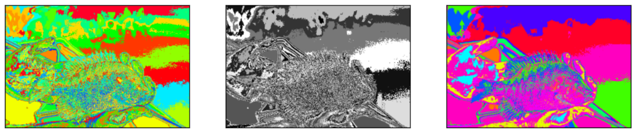
(d) Texton, brightness and color map of image 4


(e) Texton, brightness and color map of image 5


(f) Texton, brightness and color map of image 6

(g) Texton, brightness and color map of image 7


(a) Texton, brightness and color map of image 8


(b) Texton, brightness and color map of image 9

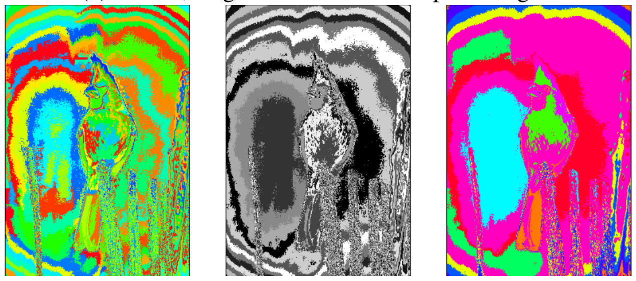

(c) Texton, brightness and color map of image 10

Fig. 5: Texton, brightness and color map

D. Texton, Brightness, and Color gradients

To compute the Texton, Brightness, and Color gradients we need to compute differences of values across different shapes and sizes. This can be achieved very efficiently by the use of Half-disc masks. The half-disc masks are simply (pairs of) binary images of half-discs. This is very important because it will allow us to compute the chi-square distances (finally obtain values of T_g,B_g,C_g) using a filtering operation, which is much faster than looping over each pixel neighborhood and aggregating counts for histograms. Forming these masks is quite trivial. A set of masks generated (8 orientations, 3 scales) is shown in Fig.

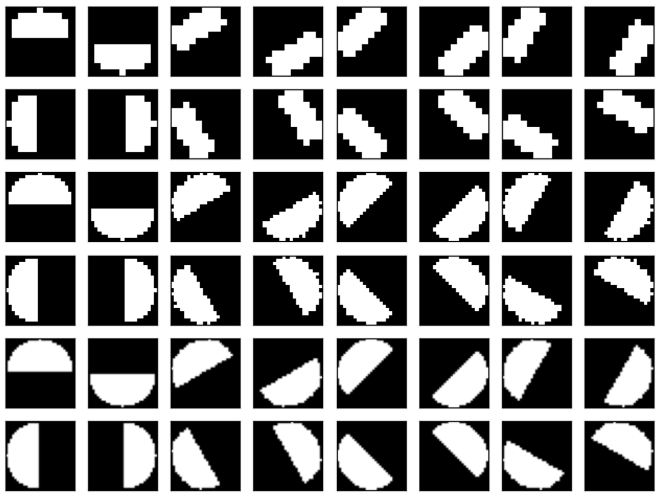
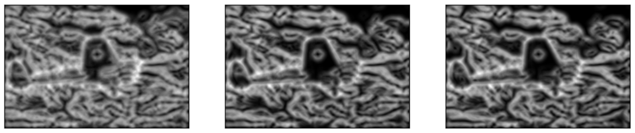
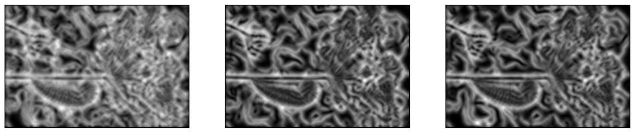
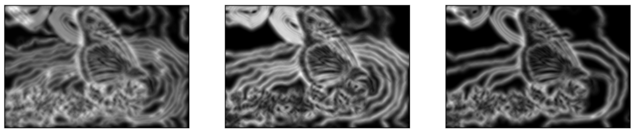
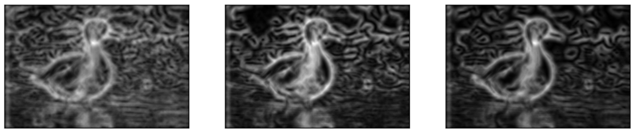


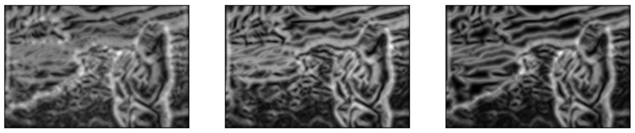
Fig. 6: Half disc mask

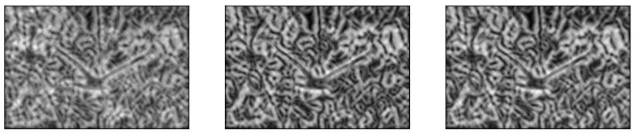

(a) Texton, brightness and color gradients of image 1

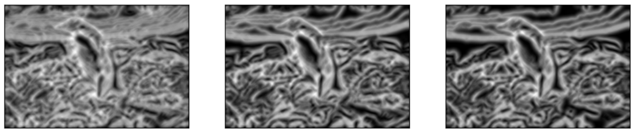

(b) Texton, brightness and color gradients of image 2

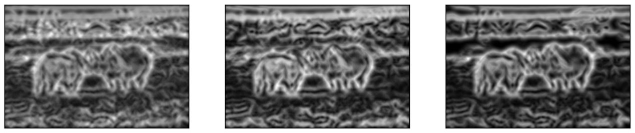

(c) Texton, brightness and color gradients of image 3

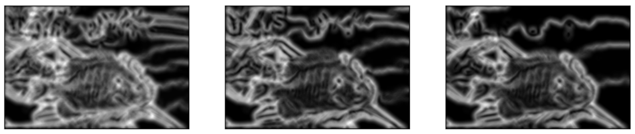

(d) Texton, brightness and color gradients of image 4

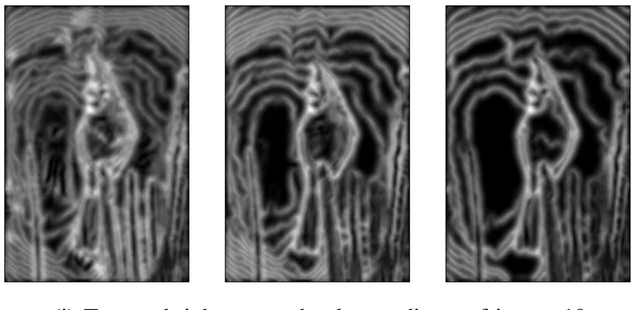

(e) Texton, brightness and color gradients of image 5


(f) Texton, brightness and color gradients of image 6


(g) Texton, brightness and color gradients of image 7


(h) Texton, brightness and color gradients of image 8


(i) Texton, brightness and color gradients of image 9

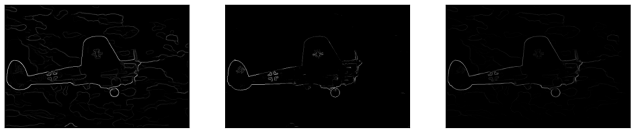

(j) Texton, brightness and color gradients of image 10

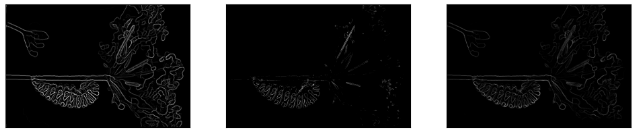
E. Pb-lite Output

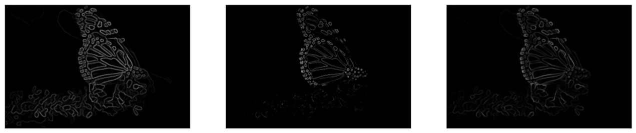
The final step is to combine information from the features with a baseline method (based on Sobel or Canny edge detection or an average of both) using a simple equation

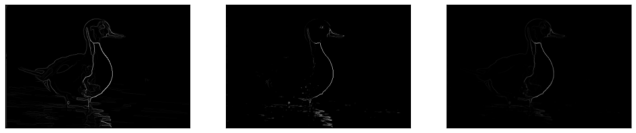
$$P_bEdges = \frac{(T_g + B_g + C_g)}{3} \circ (w_1 * cannnyPb + w_2 * sobelPb)$$

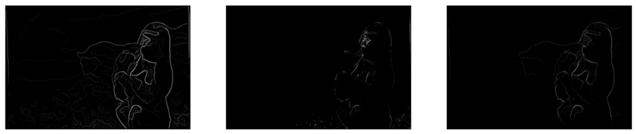
Here$\circ$ is the Hadamard product operator. And w_1, w_2 is choosen as 0.5 The comparisonbbetween Canny baseline, Sobel baseline and Pb-lite output is shown below.

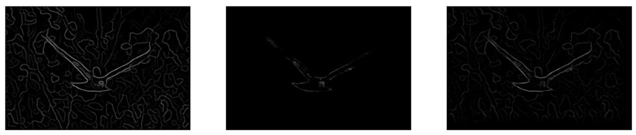

(a) Canny, Sobel and Pb-lite of image 1

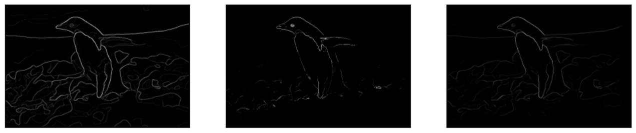

(b) Canny, Sobel and Pb-lite of image 2

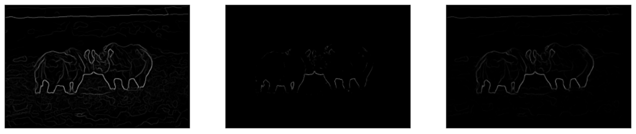

(c) Canny, Sobel and Pb-lite of image 3

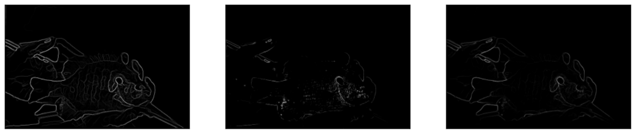

(d) Canny, Sobel and Pb-lite of image 4


(e) Canny, Sobel and Pb-lite of image 5


(f) Canny, Sobel and Pb-lite of image 6


(g) Canny, Sobel and Pb-lite of image 7


(a) Canny, Sobel and Pb-lite of image 8


(b) Canny, Sobel and Pb-lite of image 9

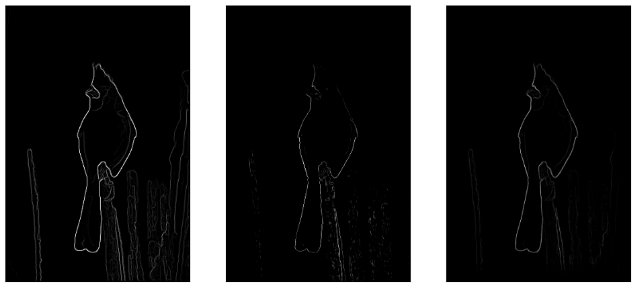

(c) Canny, Sobel and Pb-lite of image 10

Fig. 9: Canny, Sobel and Pb-lite

F. Result

In the comparative analysis, it's observed that the Canny baseline tends to produce an excessive number of false positives. On the other hand, the output from the Sobel baseline appears overly subdued. The performance of the Pb-lite output strikes a balance between these extremes. It effectively moderates the output, ensuring that it doesn't generate as many false positives as the Canny baseline, while also not obscuring critical features like the Sobel baseline.

II. Phase 2: Deep dive in deep learning

In this part of homework, modern techniques for image classification using Deep Convolutional Networks have been implemented.Various neural network architectures were applied to the CIFAR-10 dataset, and their performances were evaluated and compared in terms of loss and accuracy. The CIFAR-10 dataset comprises 50,000 training images and 10,000 test images, each of 32x32 pixel resolution. These images are categorized into a total of 10 distinct classes.

A. Baseline model

For the baseline model, a neural network architecture inspired by VGG is used. In this design, the number of convolutional filters doubles as the depth of the network increases. Meanwhile, to capture more detailed features in the images, the size of the activation maps is halved using Max Pooling with a 2x2 size, positioned between the convolutional layers. Each convolutional layer is followed by a Rectified Linear Unit (ReLU) activation function. After the convolutional layers, a collection of detailed activation maps is flattened and connected to a linear layer, which then feeds into the classification layer.

Hyper-parameter	**Value**
Optimizer	Adam
Learning Rate	1e-3
Epochs	20
Batch Size	128

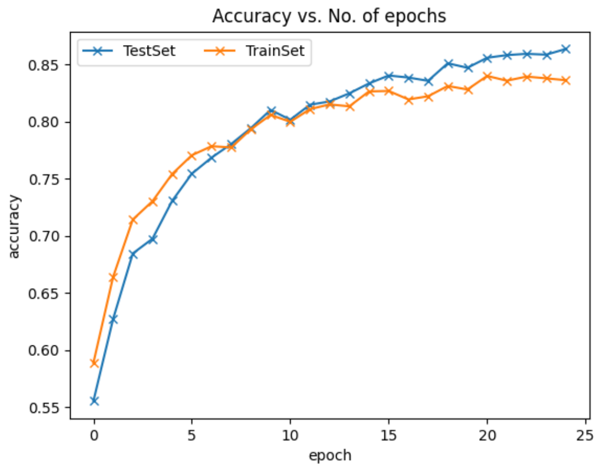


Fig. 10: Base Model - Accuracy vs epochs

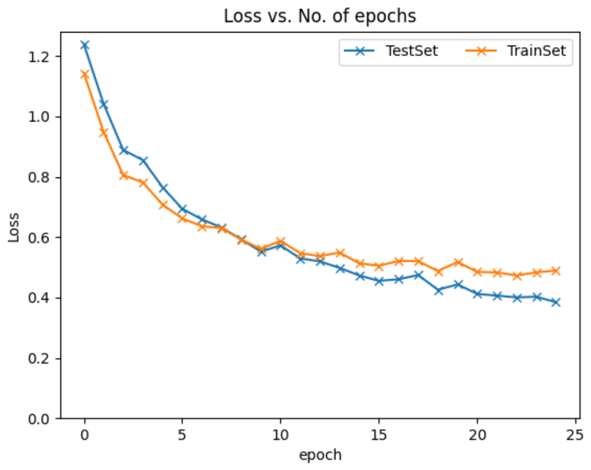
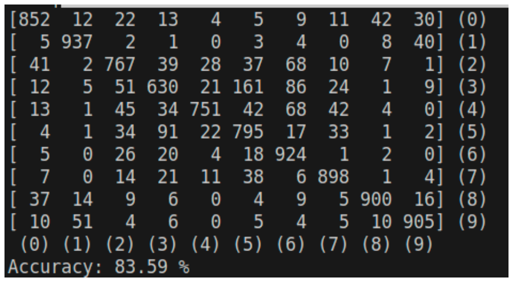
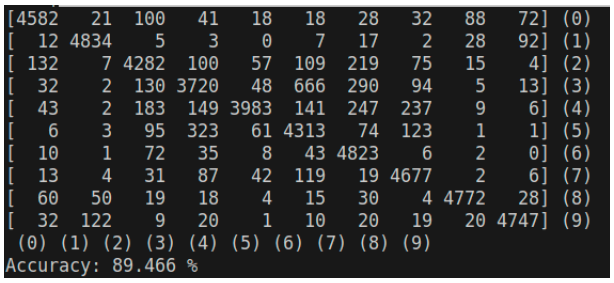


Fig. 11: Base Model - loss vs epochs



```
[852  12  22  13   4   5   9  11  42  30] (0)
[  5 937   2   1   0   3   4   0   8  40] (1)
[ 41   2 767  39  28  37  68  10   7   1] (2)
[ 12   5  51 630  21 161  86  24   1   9] (3)
[ 13   1  45  34 751  42  68  42   4   0] (4)
[  4   1  34  91  22 795  17  33   1   2] (5)
[  5   0  26  20   4  18 924   1   2   0] (6)
[  7   0  14  21  11  38   6 898   1   4] (7)
[ 37  14   9   6   0   4   9   5 900  16] (8)
[ 10  51   4   6   0   5   4   5  10 905] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 83.59 %
```

Fig. 12: Base Model - Test Set confusion Matrix



```
[4582   21  100   41   18   18   28   32   88   72] (0)
[  12 4834    5    3    0    7   17    2   28   92] (1)
[ 132    7 4282  100   57  109  219   75   15    4] (2)
[  32    2  130 3720   48  666  290   94    5   13] (3)
[  43    2  183  149 3983  141  247  237    9    6] (4)
[   6    3   95  323   61 4313   74  123    1    1] (5)
[  10    1   72   35    8   43 4823    6    2    0] (6)
[  13    4   31   87   42  119   19 4677    2    6] (7)
[  60   50   19   18    4   15   30    4 4772   28] (8)
[  32  122    9   20    1   10   20   19   20 4747] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 89.466 %
```

Fig. 13: Base Model - Train Set confusion Matrix

B. *Improvements to Baseline model*

1) Data Augmentation: Image data is standardized at the start of both the training and testing phases to align with a predetermined mean and variance. This process of normalizing the information on a per-pixel basis is crucial for maintaining consistency, ensuring that the model's weights do not have to adjust to variable targets. In addition to this standardization, the images are upscaled from their original size of 32x32 pixels to a larger resolution of 64x64 pixels.

2) Batch normalization: In addition to implementing data augmentation techniques, a batch normalization layer has been incorporated between convolutional layers. This inclusion ensures that each layer receives standardized inputs, thereby stabilizing the learning process and preventing excessive fluctuations in the model's weights.

Hyper-parameter	**Value**
Optimizer	Adam
Learning Rate	1e-3
Epochs	20
Batch Size	128

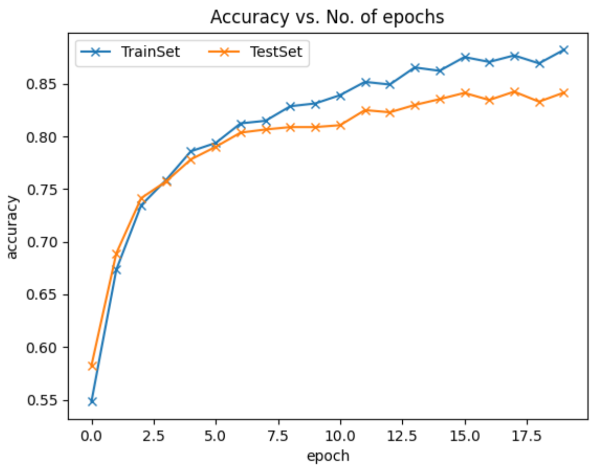


Fig. 14: Base Model - Accuracy vs epochs

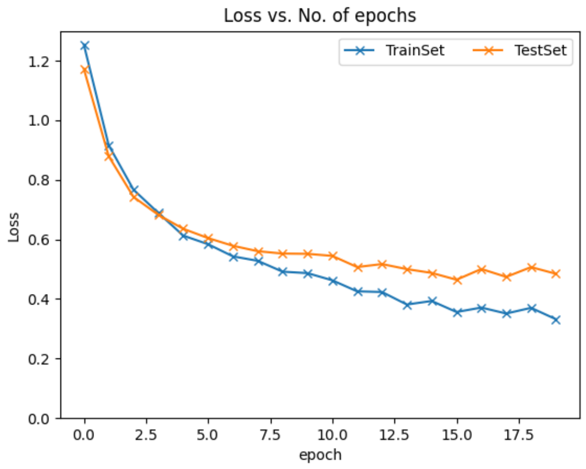
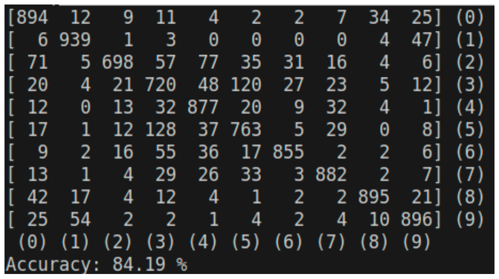
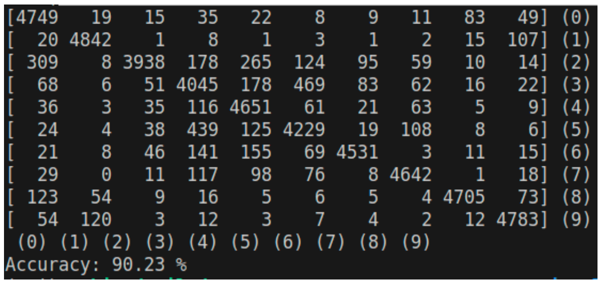


Fig. 15: Base Model - loss vs epochs



```
[894  12   9  11   4   2   2   7  34  25] (0)
[  6 939   1   3   0   0   0   0   4  47] (1)
[ 71   5 698  57  77  35  31  16   4   6] (2)
[ 20   4  21 720  48 120  27  23   5  12] (3)
[ 12   0  13  32 877  20   9  32   4   1] (4)
[ 17   1  12 128  37 763   5  29   0   8] (5)
[  9   2  16  55  36  17 855   2   2   6] (6)
[ 13   1   4  29  26  33   3 882   2   7] (7)
[ 42  17   4  12   4   1   2   2 895  21] (8)
[ 25  54   2   2   1   4   2   4  10 896] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 84.19 %
```

Fig. 16: Base Model - Test Set confusion Matrix



```
[4749   19   15   35   22    8    9   11   83   49] (0)
[  20 4842    1    8    1    3    1    2   15  107] (1)
[ 309    8 3938  178  265  124   95   59   10   14] (2)
[  68    6   51 4045  178  469   83   62   16   22] (3)
[  36    3   35  116 4651   61   21   63    5    9] (4)
[  24    4   38  439  125 4229   19  108    8    6] (5)
[  21    8   46  141  155   69 4531    3   11   15] (6)
[  29    0   11  117   98   76    8 4642    1   18] (7)
[ 123   54    9   16    5    6    5    4 4705   73] (8)
[  54  120    3   12    3    7    4    2   12 4783] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 90.23 %
```

Fig. 17: Base Model - Train Set confusion Matrix

C. *ResNet,ResNeXt,DenseNet*

1) ResNet: The core concept behind the ResNet architecture involves incorporating skip connections from previous layers. These skip connections, also known as identity connections, ensure that the model's performance does not deteriorate, and may even improve. They enable the model to leverage features from preceding layers. In this homework, the ResNet 9 architecture is employed, which consists of three consecutive residual blocks. These blocks are interspersed with convolution and pooling layers.

Hyper-parameter	**Value**
Optimizer	Adam
Learning Rate	1e-3
Epochs	20
Batch Size	128
Drop out	0.2
Weight Decay	1e-4

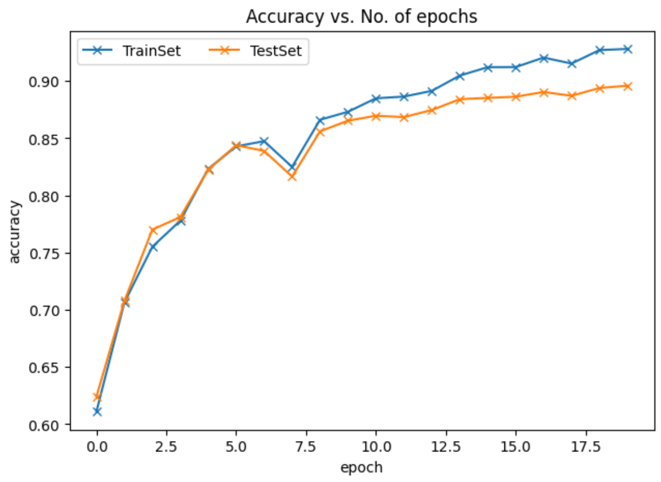


Fig. 18: ResNet Model - Accuracy vs epochs

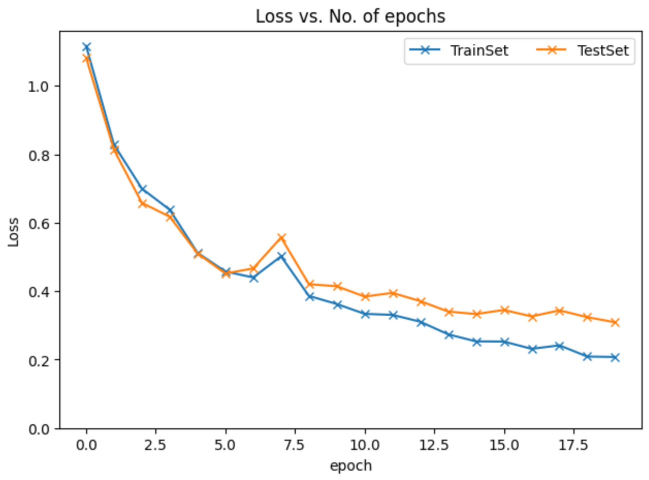
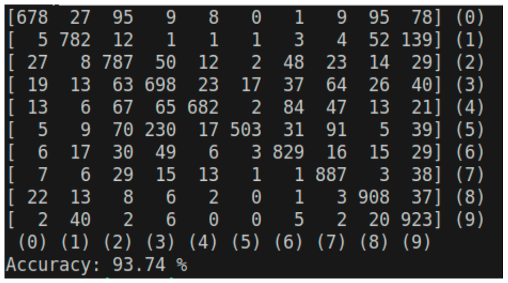
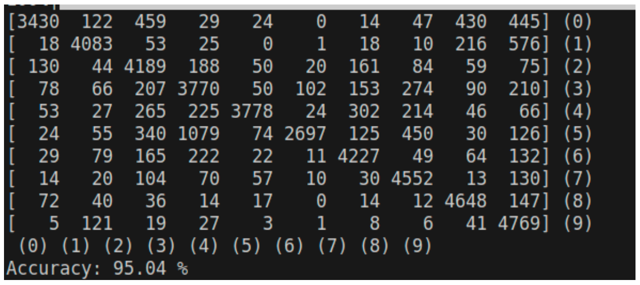


Fig. 19: ResNet Model - loss vs epochs



```
[678  27  95   9   8   0   1   9  95  78] (0)
[  5 782  12   1   1   1   3   4  52 139] (1)
[ 27   8 787  50  12   2  48  23  14  29] (2)
[ 19  13  63 698  23  17  37  64  26  40] (3)
[ 13   6  67  65 682   2  84  47  13  21] (4)
[  5   9  70 230  17 503  31  91   5  39] (5)
[  6  17  30  49   6   3 829  16  15  29] (6)
[  7   6  29  15  13   1   1 887   3  38] (7)
[ 22  13   8   6   2   0   1   3 908  37] (8)
[  2  40   2   6   0   0   5   2  20 923] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 93.74 %
```

Fig. 20: ResNEt Model - Test Set confusion Matrix



```
[3430  122  459   29   24    0   14   47  430  445] (0)
[  18 4083   53   25    0    1   18   10  216  576] (1)
[ 130   44 4189  188   50   20  161   84   59   75] (2)
[  78   66  207 3770   50  102  153  274   90  210] (3)
[  53   27  265  225 3778   24  302  214   46   66] (4)
[  24   55  340 1079   74 2697  125  450   30  126] (5)
[  29   79  165  222   22   11 4227   49   64  132] (6)
[  14   20  104   70   57   10   30 4552   13  130] (7)
[  72   40   36   14   17    0   14   12 4648  147] (8)
[   5  121   19   27    3    1    8    6   41 4769] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 95.04 %
```

Fig. 21: ResNet Model - Train Set confusion Matrix