RBE 549: Homework 0 - Alohomora

Mandeep Singh
M.S. Robotics Engineering
Worcester Polytechnic Institute (WPI)
Email: msingh2@wpi.edu

***Abstract*—There are two phases of this homework, Phase 1 and Phase 2 respectively. Phase 1 is about developing pb (probability of boundary) boundary detection computer vision algorithm which significantly outperforms the classical methods of edge detection like Canny and Sobel baselines. Phase 2 of the homework focuses on image classification using Deep learning approach which has become famous in recent times. Various Neural networks are trained on CIFAR-10 dataset and their performances in terms of accuracy and losses have been compared.**

***Index Terms*—Computer vision, Boundary detection, Deep learning.**

I. Phase 1 : Shake My Boundary

A. Introduction

In Phase 1 of homework, a simplified version of Probability of boundary detection algorithm is developed, which finds boundaries by examining brightness, color, and texture information across multiple scales in the image. This boundary detector outperforms the Canny and Sobel detectors as it suppresses the false positives that the other methods produce in textured regions. The implementation of the algorithm has been tested on Berkeley Segmentation Dataset 500.

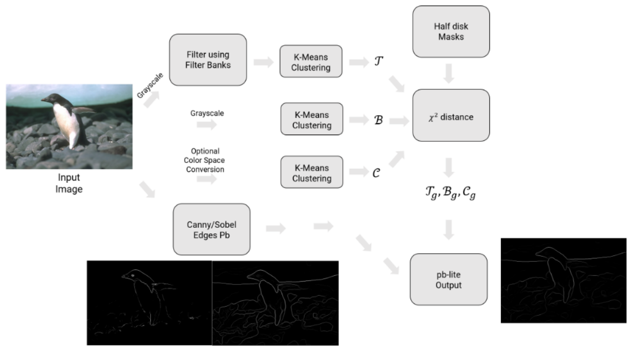


Fig. 1: Overview of the Pb lite pipeline.

The first step of the pipeline is to filter the image with a set of filter banks. Once filtered a texton, brightness and color map is generated by clustering the filter responses. Then the image gradients can be calculated which show how much the texture, brightness and color distributions are changing at a pixel. Finally, the gradient outputs are combined with Sobel and Canny baselines using appropriate weights to get the required result.

B. Filters

Images are filtered using a collection of filters to measure texture properties and to aggregate regional texture and brightness distributions. Three different sets of filter banks are created for this purpose.

1) Oriented Derivative of Gaussian (DoG) Filters: A simple DoG filter is created by convolving a Sobel filter and a Gaussian kernel.This filter can then be rotated at different scales to find a series of oriented DoG filters.

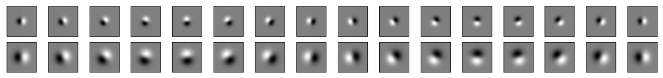


Fig. 2: Oriented DoG filter bank with kernel size = 50 and σ = [3, 5]

2) Leung-Malik Filters: The Leung-Malik filters are a set of multi scale, multi orientation filter bank with 48 filters. Filters consists of first and second order derivatives of Gaussian with 3 different scales and 6 orientations, 8 Laplacian filters and 4 basic gaussian kernels.

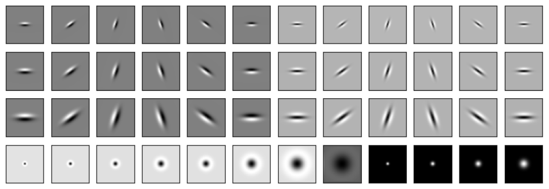


Fig. 3: Larger Leung-Malik filters

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

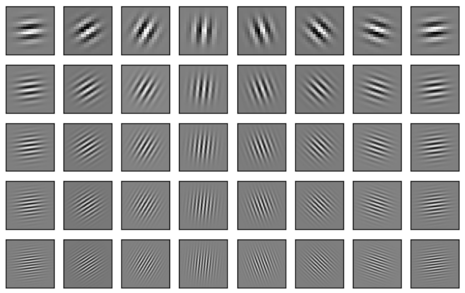
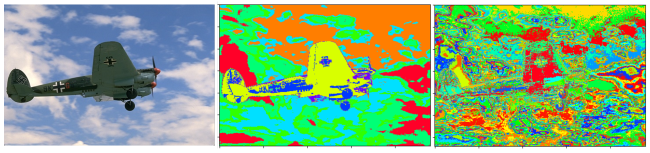
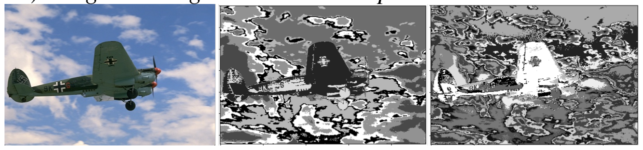


Fig. 4: Gabor filter bank with σ = [3, 5, 7, 9, 12]

C. Texton, Brightness, and Color Map

To create Texton maps all the filters (total 120) are applied to the image and a stack of resultant outputs are obtained. Now the task is to group the pixels having similar texture properties and give a discrete texton ID to each pixel. This is done by using KMeans clustering method where the similar pixel values will be assigned to one cluster. In this case, 64 cluster centers are being used to categorize each pixel. More number of clusters means more detail about the texture and vice-versa. In Fig. 5 the differnce in the texton maps output by using different number of clusters can be seen.


a) Original image and texton maps with K = 8 and 64.


b) Original image and brightness maps with K = 8 and 16

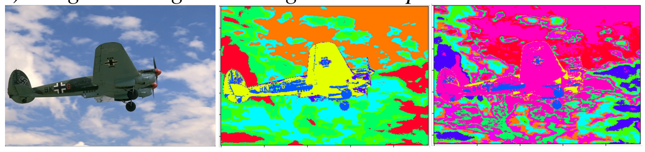
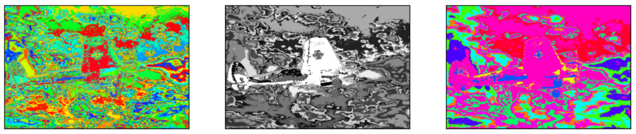
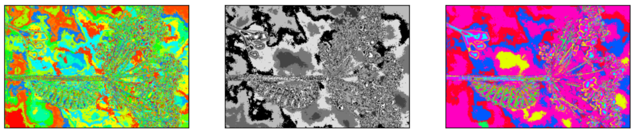
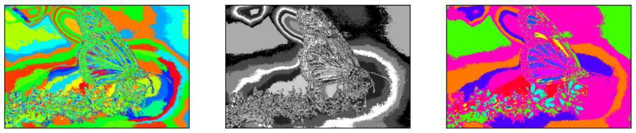

c) Original image and color maps with K = 8 and 16

Fig. 5: Difference between maps by varying number of clusters

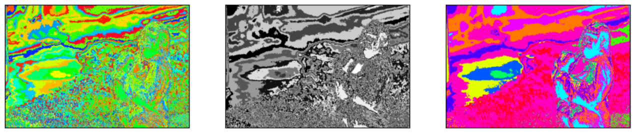
Similar to the texton map, brightness maps and color maps are created for the same image. For the brightness map grayscale version of the original image is used whereas for the color map RGB image is used. 16 clusters are used in case of both brightness and color maps. Fig 5 b) and Fig 5 c) can be referred to see the difference between brightness and color maps with cluster size 8 and 16. For better results 64, 16 and 16 clusters are finalized for texton, brightness and color map respectively. Results on the BSDS500 dataset can be seen in Fig 6.

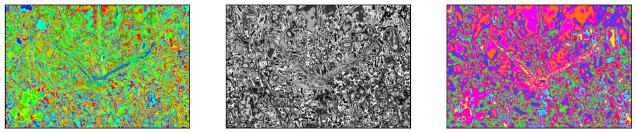

a) Texton, brightness and color maps for image 1

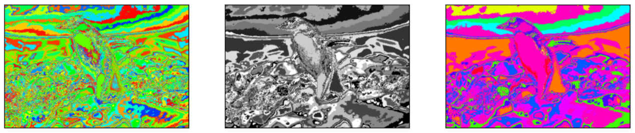

b) Texton, brightness and color maps for image 2

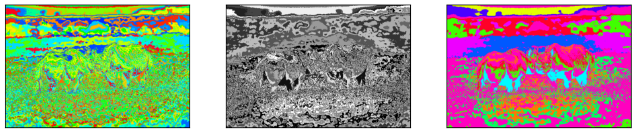

c) Texton, brightness and color maps for image 3

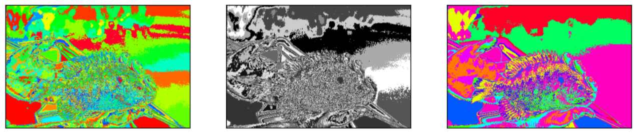
d) Texton, brightness and color maps for image 4


e) Texton, brightness and color maps for image 5


f) Texton, brightness and color maps for image 6


g) Texton, brightness and color maps for image 7


h) Texton, brightness and color maps for image 8


i) Texton, brightness and color maps for image 9

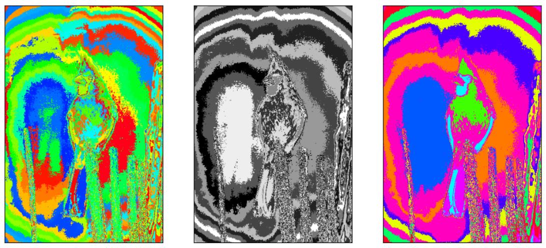

j) Texton, brightness and color maps for image 10

Fig. 6: Texton, brightness and color map of all images

D. Texton, Brightness, and Color Gradients

To calculate the gradients of maps, half disc masks of different scales and sizes are used. These masks helps to calculate chi-square distances using a filtering operation.Fig 7 shows half disc masks.

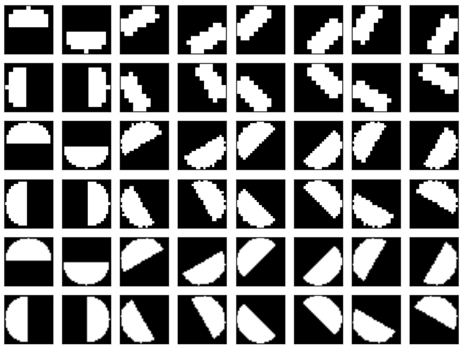
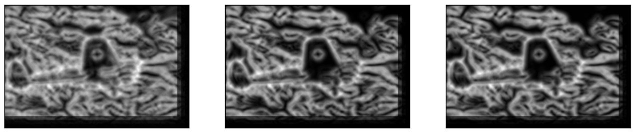
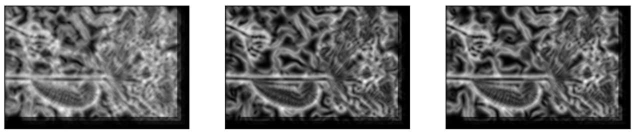
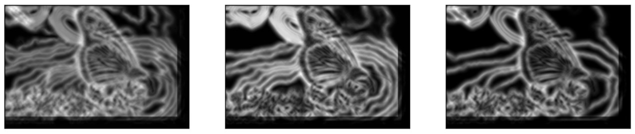


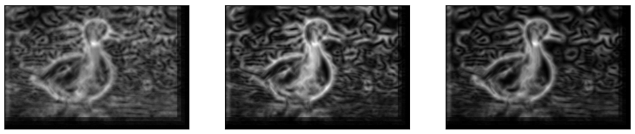
Fig. 7: Half disc masks with σ = [5, 10, 15]

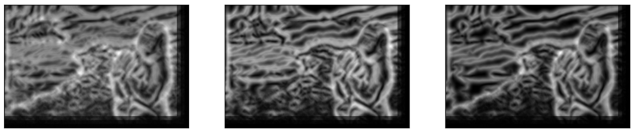
The texture, brightness and color gradient shows how much the texture, brightness and color distributions are changing at a pixel. Gradients are calculated by comparing the distributions in left/right half-disc pairs. Fig 8 can be referred for the outputs of texton, brightness and color gradients for all the images.

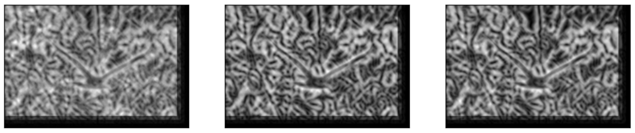

a) Texton, brightness and color gradients for image 1

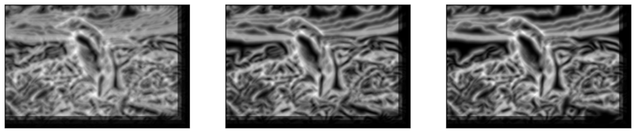

b) Texton, brightness and color gradients for image 2

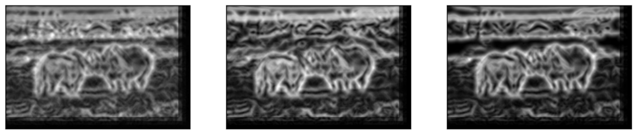

c) Texton, brightness and color gradients for image 3

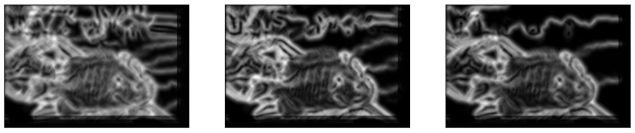

d) Texton, brightness and color gradients for image 4


e) Texton, brightness and color gradients for image 5


f) Texton, brightness and color gradients for image 6


g) Texton, brightness and color gradients for image 7


h) Texton, brightness and color gradients for image 8


i) Texton, brightness and color gradients for image 9

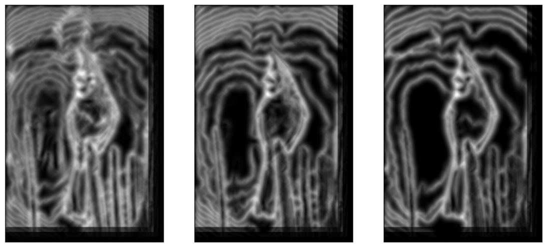

j) Texton, brightness and color gradients for image 10

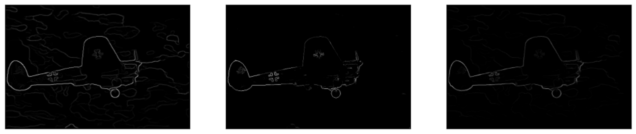
Fig. 8: Texton, brightness and color gradient of all images

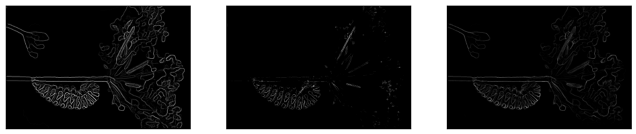
E. Pb-lite Output

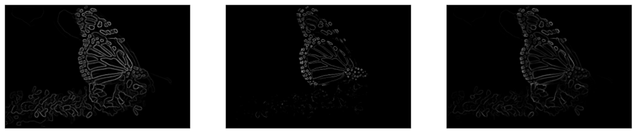
The final step in the pipeline is to combine the gradients with features from Sobel and Canny detectors by using the simple equation below:

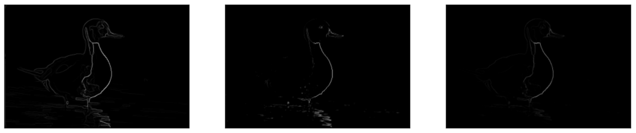
$$PbEdges = \frac{(\mathcal{T}_g + \mathcal{B}_g + \mathcal{C}_g)}{3} \odot (w_1 * cannyPb + w_2 * sobelPb)$$

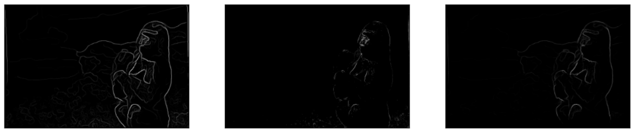
Here w1 and w2 are taken as 0.5 each. The comparison between Canny baseline, Sobel baseline and Pb-lite output is shown in Fig.9.

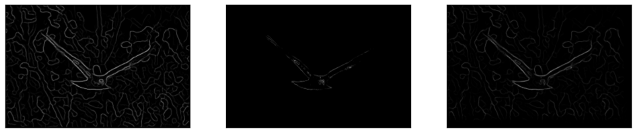

a) Canny, Sobel and Pb-lite output of image 1

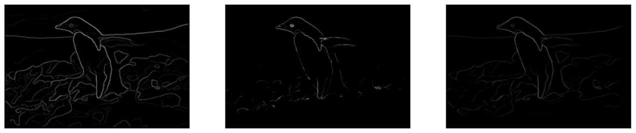

b) Canny, Sobel and Pb-lite output of image 2

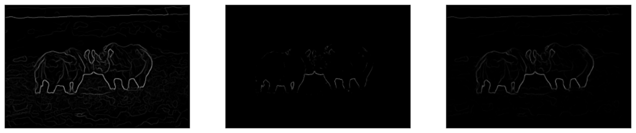

c) Canny, Sobel and Pb-lite output of image 3

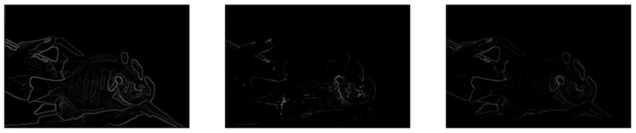

d) Canny, Sobel and Pb-lite output of image 4


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


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


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


h) Canny, Sobel and Pb-lite output of image 8


i) Canny, Sobel and Pb-lite output of image 9

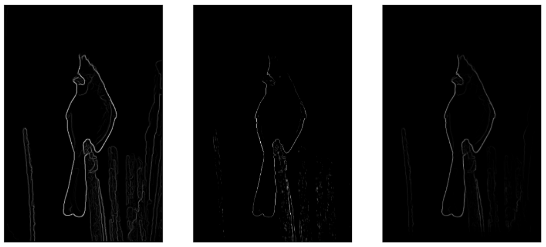

j) Canny, Sobel and Pb-lite output of image 10

Fig. 9: Canny, Sobel and Pb-lite outputs

F. Results analysis

It can be noticed in the comparison that Canny baseline gives too many false positives whereas Sobel baseline output is a little too suppressed. The Pb-lite output performance lies in between these two where output is just enough suppressed that it neither gives any false positives like canny baseline and nor does it hide the important features as is the case in sobel. Except image 3 and 9 the Pb-lite edge detector has done quite a good job. Maybe if the parameters are optimized further even better results can be obtained.

II. Phase 2: Deep Dive on Deep Learning

A. Introduction

In Phase 2 of this homework multiple neural network architectures are implemented on CIFAR-10 dataset and their performance on the basis of loss and accuracy are compared against each other. Also, various methods to improve this classification task such as data augmentation, normalization, regularization, etc to increase the accuracy and at the same time reduce over-fitting on the training data. CIFAR-10 dataset consists of 32x32 size 50,000 training and 10,000 test images. All the images are classified into total 10 classes.

B. First Neural Network - LeNet

Convolutional Neural networks are the most commonly employed models for image classification. So, as a first approach to the CIFAR-10 dataset a very basic neural network architecture called the LeNet is implemented. The basic architecture of the LeNet can be seen in the Fig. 10. there are total 3 convolutional layers of the network. Activation fucntion used after each layer is tanh and Average pooling is used to downsample the feature maps after each convolutional layer. At the end linear layers are used to classify the images into 10 classes.

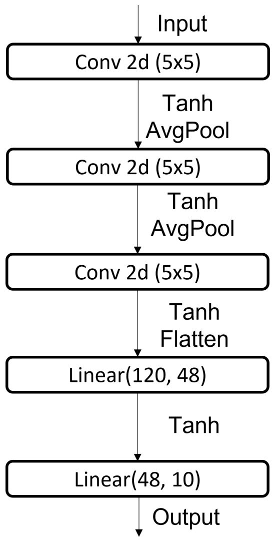


Fig. 10: LeNet architecture

TABLE I: Hyperparameters for LeNet architecture

Hyper-parameter	Value
Optimizer	Adam
Learning rate	1e-3
Mini Batch Size	128
Epochs	30

Table 1 can be referred for the hyperparameters used with LeNet architecture. Withour any data normalization and data augmentation the accuracy and loss values were very poor. Accuracy and loss plots over train set and test set can be seen in Fig. 11 and Fig. 12. The total number of parameters for the model are 57,290. Also, the confusion matrix for the train set and test can be referred from Fig 13 and Fig 14.

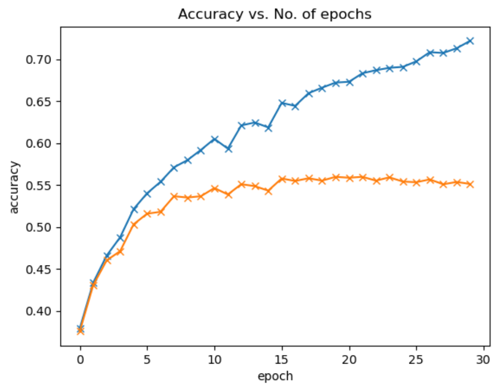


Fig. 11: LeNet model - Accuracy vs Epochs

The train set accuracy was over 0.7 in about 30 epochs and it was still increasing but test set accuracy kind of remain unchanged after 10 epochs which clearly shows the problem of overfitting. Also, overfitting trend can be noticed in loss vs epoch plot in Fig 12 where loss starts to increase after a certain number of epochs.

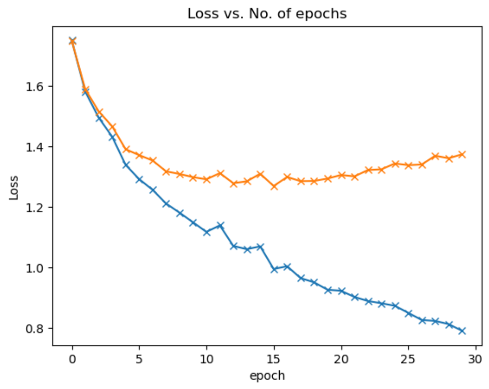
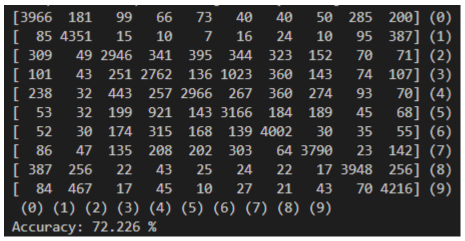
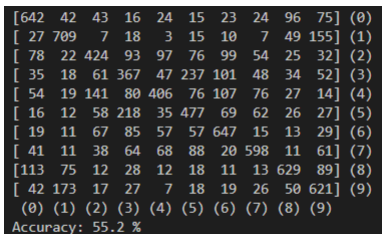


Fig. 12: LeNet model - Loss vs Epochs



```
[3966  181   99   66   73   40   40   50  285  200] (0)
[  85 4351   15   10    7   16   24   10   95  387] (1)
[ 309   49 2946  341  395  344  323  152   70   71] (2)
[ 101   43  251 2762  136 1023  360  143   74  107] (3)
[ 238   32  443  257 2966  267  360  274   93   70] (4)
[  53   32  199  921  143 3166  184  189   45   68] (5)
[  52   30  174  315  168  139 4002   30   35   55] (6)
[  86   47  135  208  202  303   64 3790   23  142] (7)
[ 387  256   22   43   25   24   22   17 3948  256] (8)
[  84  467   17   45   10   27   21   43   70 4216] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 72.226 %
```

Fig. 13: LeNet model - Confusion matrix on train set



```
[642  42  43  16  24  15  23  24  96  75] (0)
[ 27 709   7  18   3  15  10   7  49 155] (1)
[ 78  22 424  93  97  76  99  54  25  32] (2)
[ 35  18  61 367  47 237 101  48  34  52] (3)
[ 54  19 141  80 406  76 107  76  27  14] (4)
[ 16  12  58 218  35 477  69  62  26  27] (5)
[ 19  11  67  85  57  57 647  15  13  29] (6)
[ 41  11  38  64  68  88  20 598  11  61] (7)
[113  75  12  28  12  18  11  13 629  89] (8)
[ 42 173  17  27   7  18  19  26  50 621] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 55.2 %
```

Fig. 14: LeNet model - Confusion matrix on test set

C. Improved neural network

First of all to reduce the overfit problem it was decided to augment the data and generate some randomness in the model. Also, the data was normalized as per the mean and standard deviation values for the CIFAR-10 dataset. The model architecture was also changed, network was made more deep with additional convolutional layers with added dropout layer to regularize the gradients and reduce overfitting as well. The model was inspired from VGG-11 but has some changes to it as per the CIFAR-10 dataset.
Augmmentation methods used are Random crop after padding the image by 4 and then random horizontal flip with a probablity of 0.5, which are commonly used data augmentation methods for CIFAR-10.

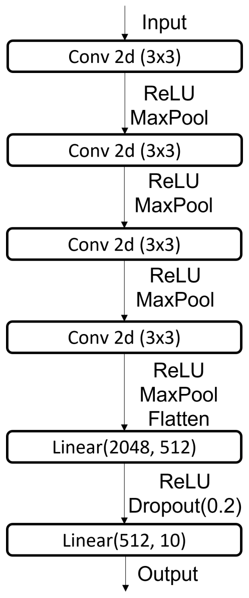


Fig. 15: Modified model architecture

TABLE II: Hyperparameters for Modified model architecture

Hyper-parameter	Value
Optimizer	Adam
Learning rate	1e-3
Mini Batch Size	128
Epochs	20
Dropout	0.2

Table II can be referred for the hyperparameters used with modified architecture. Significant improvement can be noticed after data augmentation and a deep neural network. Accuracy and loss plots over train set and test set can be seen in Fig. 16 and Fig. 17. The total number of parameters for the model are 2,605,194. Also, the confusion matrix for the train set and test can be referred from Fig 18 and Fig 19.

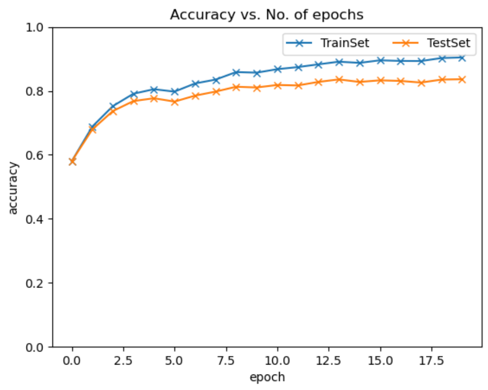


Fig. 16: Modified model - Accuracy vs Epochs

The model worked really well and reached 0.80 accuracy within just 3 epochs. the train accuracy reached about 0.9 and still increasing after 20 epochs but the test set accuracy became consistent and there is a little overfitting in the model as can be seen in Fig. 16 and fig. 17 but this much overfit is expected on the train set.

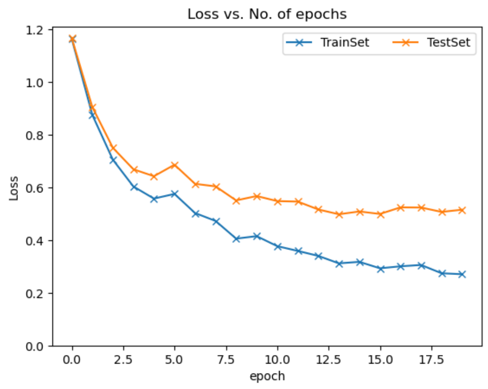
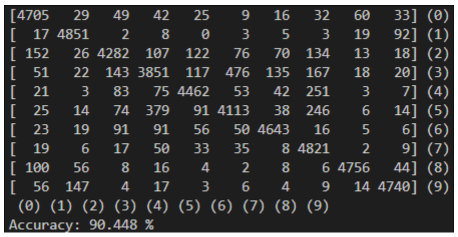
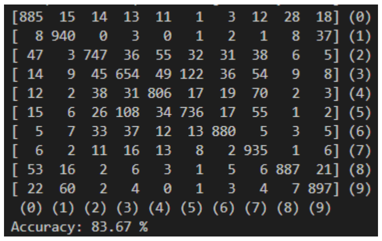


Fig. 17: Modified model - Loss vs Epochs



```
[4705   29   49   42   25    9   16   32   60   33] (0)
[  17 4851    2    8    0    3    5    3   19   92] (1)
[ 152   26 4282  107  122   76   70  134   13   18] (2)
[  51   22  143 3851  117  476  135  167   18   20] (3)
[  21    3   83   75 4462   53   42  251    3    7] (4)
[  25   14   74  379   91 4113   38  246    6   14] (5)
[  23   19   91   91   56   50 4643   16    5    6] (6)
[  19    6   17   50   33   35    8 4821    2    9] (7)
[ 100   56    8   16    4    2    8    6 4756   44] (8)
[  56  147    4   17    3    6    4    9   14 4740] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 90.448 %
```

Fig. 18: Modified model - Confusion matrix on train set



```
[885  15  14  13  11   1   3  12  28  18] (0)
[  8 940   0   3   0   1   2   1   8  37] (1)
[ 47   3 747  36  55  32  31  38   6   5] (2)
[ 14   9  45 654  49 122  36  54   9   8] (3)
[ 12   2  38  31 806  17  19  70   2   3] (4)
[ 15   6  26 108  34 736  17  55   1   2] (5)
[  5   7  33  37  12  13 880   5   3   5] (6)
[  6   2  11  16  13   8   2 935   1   6] (7)
[ 53  16   2   6   3   1   5   6 887  21] (8)
[ 22  60   2   4   0   1   3   4   7 897] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 83.67 %
```

Fig. 19: Modified model - Confusion matrix on test set

D. ResNet

As per the general behaviour of neural networks it is assumed that as we go deeper and deeper into the networks the performance should increase. But many other problems arises as we build a model with deep layers. First problem is the bigger the network more difficult it is to train and second problem is of vanishing gradients as the gradient is back-propagated to the earlier layers. Due to vanishing gradients the model becomes saturated because of the negligible gradients it doesn't learn anything new.

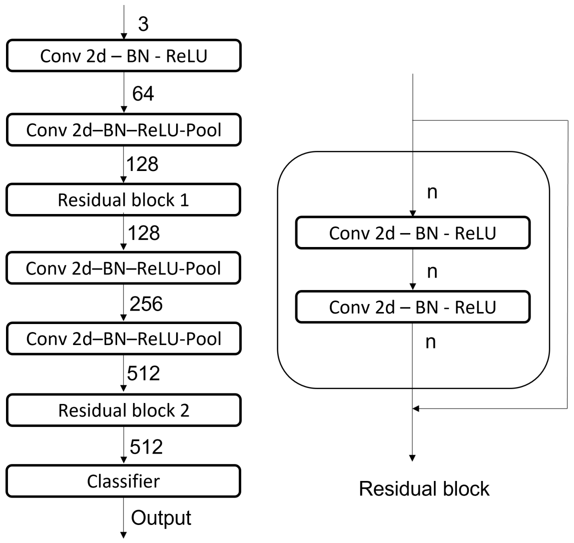


Fig. 20: ResNet architecture

The main idea of the ResNet architecture is to add a skip connection from previous layer as shown in Fig. 20. Adding skip connection (identity connections) makes sure that the model performance doesn't degrade if not increasing as such. Also, the model learns from the features from previous layers.

In this homework ResNet 9 architecture has been used where three such residual blocks are stacked one after the other with convolution and pooling layers in between.

TABLE III: Hyperparameters for ResNet architecture

Hyper-parameter	Value
Optimizer	Adam
Learning rate	1e-3
Mini Batch Size	128
Epochs	50
Dropout	0.2
Weight decay	1e-4

Table 1 can be referred for the hyperparameters used with modified architecture. Significant improvement can be seen from the earlier networks. Accuracy and loss plots over train set and test set can be seen in Fig. 21 and Fig. 22. The total number of parameters for the model are 6,575,370. Also, the confusion matrix for the train set and test can be referred from Fig 23 and Fig 24.

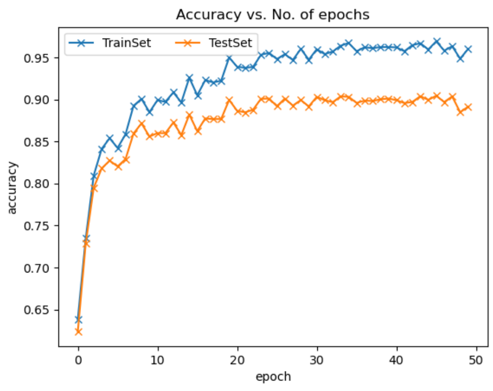


Fig. 21: ResNet model - Accuracy vs Epochs

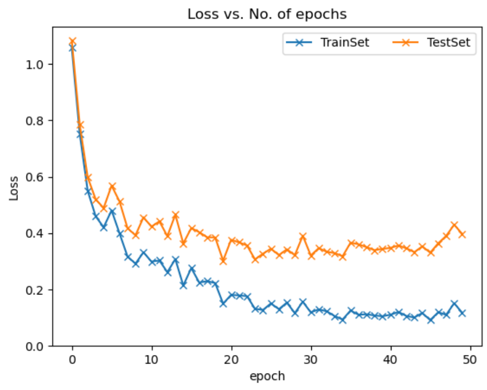
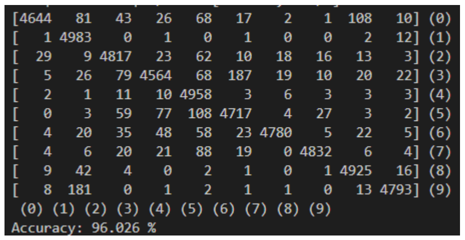


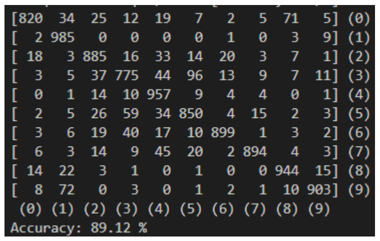
Fig. 22: ResNet model - Loss vs Epochs

The model worked really well and reached 0.95 training accuracy after training over 30 epochs. The test accuracy reached about 0.9 and became saturated hinting that the model started to overfit after that point.



```
[4644   81   43   26   68   17    2    1  108   10] (0)
[   1 4983    0    1    0    1    0    0    2   12] (1)
[  29    9 4817   23   62   10   18   16   13    3] (2)
[   5   26   79 4564   68  187   19   10   20   22] (3)
[   2    1   11   10 4958    3    6    3    3    3] (4)
[   0    3   59   77  108 4717    4   27    3    2] (5)
[   4   20   35   48   58   23 4780    5   22    5] (6)
[   4    6   20   21   88   19    0 4832    6    4] (7)
[   9   42    4    0    2    1    0    1 4925   16] (8)
[   8  181    0    1    2    1    1    0   13 4793] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 96.026 %
```

Fig. 23: ResNet model - Confusion matrix on train set



```
[820  34  25  12  19   7   2   5  71   5] (0)
[  2 985   0   0   0   0   1   0   3   9] (1)
[ 18   3 885  16  33  14  20   3   7   1] (2)
[  3   5  37 775  44  96  13   9   7  11] (3)
[  0   1  14  10 957   9   4   4   0   1] (4)
[  2   5  26  59  34 850   4  15   2   3] (5)
[  3   6  19  40  17  10 899   1   3   2] (6)
[  6   3  14   9  45  20   2 894   4   3] (7)
[ 14  22   3   1   0   1   0   0 944  15] (8)
[  8  72   0   3   0   1   2   1  10 903] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 89.12 %
```

Fig. 24: ResNet model - Confusion matrix on test set

E. *DenseNet*

DenseNet architecture employs the basic idea of skip connections (adding earlier layers) but the only difference is that in each DenseNet block output of all the preceding layers is fed into the current layer. Also one more main difference is that unlike residual block the outputs are stacked in dense block and not just added as done in residual blocks. This can be better understood by the dense block configuration shown in Fig. 25.

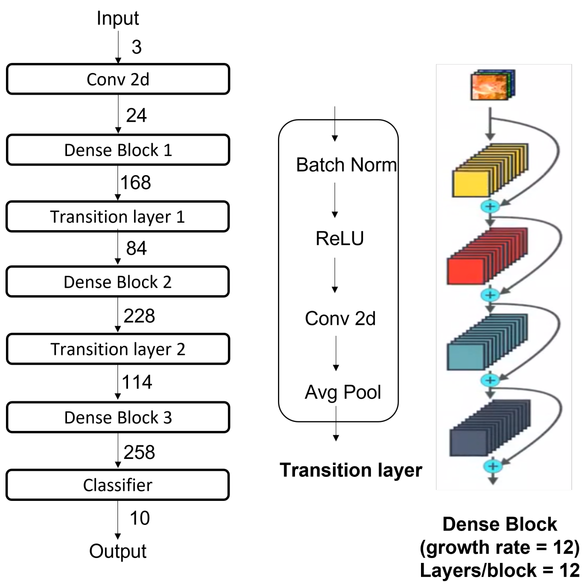


Fig. 25: DenseNet - 40 architecture

For the ease of adding connections the image size is not changed. That's why to downsample the data, transition layers mainly consisting of convolution and pooling layers are employed. One more advantage of DenseNet is that it uses very fewer parameters than ResNets but still give equivalent results. The DenseNet employed in this paper is DenseNet-40 which has 3 denseblocks each having 12 layers and in between each block the growth rate of the width of the network is also 12. For better understandinfg Fig. 25 can be referred.

TABLE IV: Hyperparameters for DenseNet architecture

Hyper-parameter	Value
Optimizer	Adam
Learning rate	1e-3
Mini Batch Size	128
Epochs	25
Weight decay	1e-4

Table 1 can be referred for the hyperparameters used with DenseNet architecture. Accuracy and loss plots over train set and test set can be seen in Fig. 26 and Fig. 27. The total number of parameters for the model are 489,112. Also, the confusion matrix for the train set and test can be referred from Fig 28 and Fig 29.

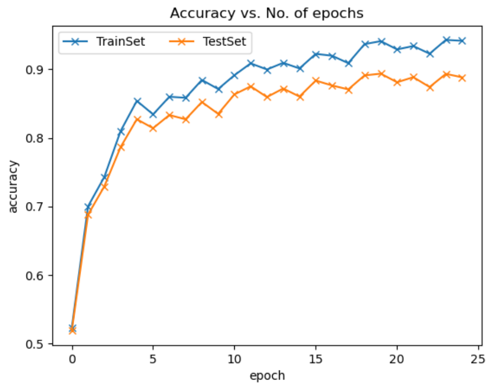


Fig. 26: DenseNet model - Accuracy vs Epochs

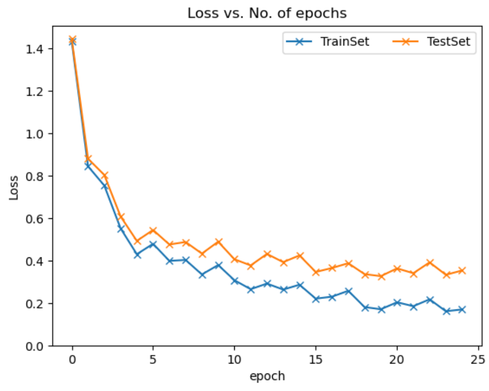
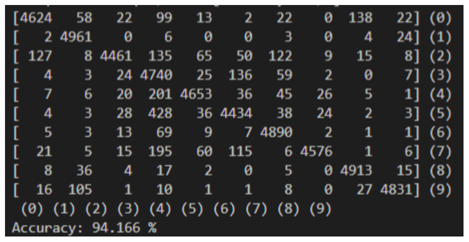


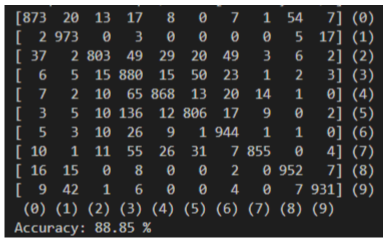
Fig. 27: DenseNet model - Loss vs Epochs

The train set accuracy crossed 0.95 mark whereas best test accuracy is around 0.9.



```
[4624   58   22   99   13    2   22    0  138   22] (0)
[   2 4961    0    6    0    0    3    0    4   24] (1)
[ 127    8 4461  135   65   50  122    9   15    8] (2)
[   4    3   24 4740   25  136   59    2    0    7] (3)
[   7    6   20  201 4653   36   45   26    5    1] (4)
[   4    3   28  428   36 4434   38   24    2    3] (5)
[   5    3   13   69    9    7 4890    2    1    1] (6)
[  21    5   15  195   60  115    6 4576    1    6] (7)
[   8   36    4   17    2    0    5    0 4913   15] (8)
[  16  105    1   10    1    1    8    0   27 4831] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 94.166 %
```

Fig. 28: DenseNet model - Confusion matrix on train set



```
[873  20  13  17   8   0   7   1  54   7] (0)
[  2 973   0   3   0   0   0   0   5  17] (1)
[ 37   2 803  49  29  20  49   3   6   2] (2)
[  6   5  15 880  15  50  23   1   2   3] (3)
[  7   2  10  65 868  13  20  14   1   0] (4)
[  3   5  10 136  12 806  17   9   0   2] (5)
[  5   3  10  26   9   1 944   1   1   0] (6)
[ 10   1  11  55  26  31   7 855   0   4] (7)
[ 16  15   0   8   0   0   2   0 952   7] (8)
[  9  42   1   6   0   0   4   0   7 931] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 88.85 %
```

Fig. 29: DenseNet model - Confusion matrix on test set

F. *ResNeXT*

ResNeXt architecture is based on the combination of Resnet and Inception module as it follows split-tranform-merge principle from Inception module and residual blocks like Resnets. According to ResNeXt paper instead of forming deep neural network we can achiwve the same performance by increasing the cardinality and training the same data by dividing into groups and then merging their outputs. ResNeXt architecture uses very less model parameters as compared to other well known networks. Fig. 15 can be referred for the modified neural network architecture.

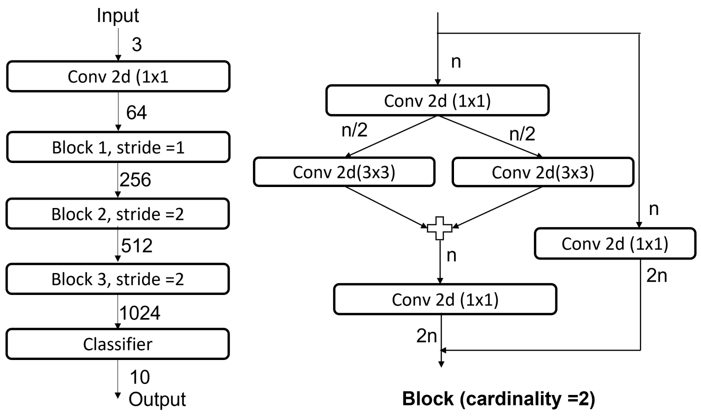


Fig. 30: ResNeXT architecture

In this homework, the simplest ResNeXt architecture is implemented using cardinality as 2. The width (channels) of the network are increased after each block. One thing to note in this architecture is that here stride is used to downsample the data. For e.g. stride of 2 is used to reduce the image size by half.

TABLE V: Hyperparameters for ResNeXT architecture

Hyper-parameter	Value
Optimizer	Adam
Learning rate	1e-3
Mini Batch Size	128
Epochs	30
Weight decay	1e-4

Table V can be referred for the hyperparameters used with ResNeXt architecture. Accuracy and loss plots over train set and test set can be seen in Fig. 31 and Fig. 32. The total number of parameters for the model are 3,270,794. Also, the confusion matrix for the train set and test can be referred from Fig 33 and Fig 34.

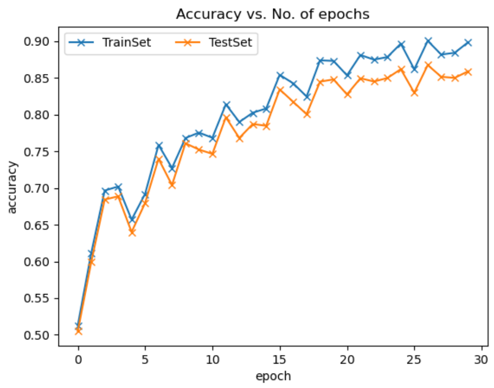


Fig. 31: ResNeXT model - Accuracy vs Epochs

The train and test accuracy is around 0.85 but it can further increase if the model is allowed trained for further epochs.

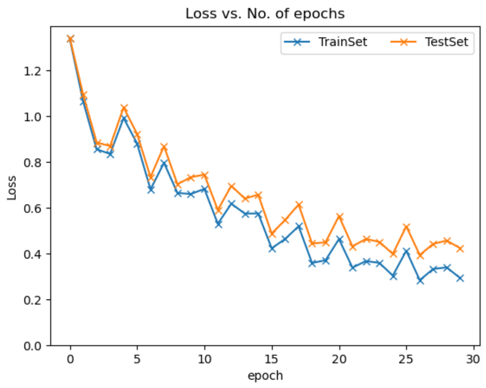
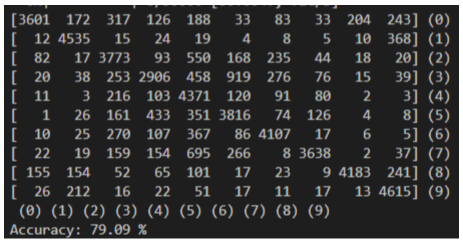


Fig. 32: ResNeXT model - Loss vs Epochs

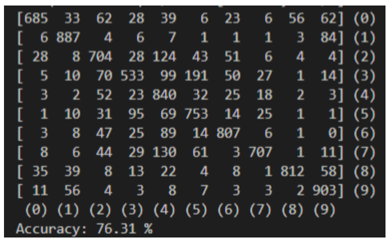
G. *Comparison of all Networks*

A detailed comparison on the basis of accuracy, and model parameters is done. All the models vary in their depth of network architecture and their key principles. Although this comparison can't be exact as most of the models if trained



```
[3601  172  317  126  188   33   83   33  204  243] (0)
[  12 4535   15   24   19    4    8    5   10  368] (1)
[  82   17 3773   93  550  168  235   44   18   20] (2)
[  20   38  253 2906  458  919  276   76   15   39] (3)
[  11    3  216  103 4371  120   91   80    2    3] (4)
[   1   26  161  433  351 3816   74  126    4    8] (5)
[  10   25  270  107  367   86 4107   17    6    5] (6)
[  22   19  159  154  695  266    8 3638    2   37] (7)
[ 155  154   52   65  101   17   23    9 4183  241] (8)
[  26  212   16   22   51   17   11   17   13 4615] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 79.09 %
```

Fig. 33: ResNeXT model - Confusion matrix on train set



```
[685  33  62  28  39   6  23   6  56  62] (0)
[  6 887   4   6   7   1   1   1   3  84] (1)
[ 28   8 704  28 124  43  51   6   4   4] (2)
[  5  10  70 533  99 191  50  27   1  14] (3)
[  3   2  52  23 840  32  25  18   2   3] (4)
[  1  10  31  95  69 753  14  25   1   1] (5)
[  3   8  47  25  89  14 807   6   1   0] (6)
[  8   6  44  29 130  61   3 707   1  11] (7)
[ 35  39   8  13  22   4   8   1 812  58] (8)
[ 11  56   4   3   8   7   3   3   2 903] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 76.31 %
```

Fig. 34: ResNeXT model - Confusion matrix on test set

for more time can be achieve better accuracy. But still the main concepts that had to be learned can be interpreted from this data. Table VI can referred for the detailed comparison.

TABLE VI: Comparison of all CNN's

Network	Test Accuracy	Model Parameters
LeNet	55.2%	57,290
VGG-11	83.67%	2,605,194
ResNet-9	89.12%	6,575,370
DenseNet-40	88.85%	489,112
ResNeXt	76.31%	3,270,794

It can be seen how initial LeNet without must depth and any data augmentation method performed poorly on the model. Then VGG-11 inspired model improved the accuracy which had employed a deeper network and data augmentation and L2 regularization. The next ResNet 9 model increased accuracy further but has also the maximum number of model parameters crossing 6 million. DenseNet architecture as discussed also provided results comparable to ResNet but has parameters only 1/10 of those of ResNet. ResNeXt model also has almost half the model parameters as compared to ResNet and its accuracy can be further increased by increasing cardinality and training for more time.