Homework 0 - Alohamora!

*

Sayan Brahma
M.Engg Robotics(using 2 late days)
University of Maryland - College Park
sbrahma@terpmail.umd.edu

***Abstract*—This homework has two phases. Phase 1 is about implementation of pb (probability of boundary) algorithm using the texture, color and brightness information combined with the canny and sobel baselines features. Phase 2 is the implementation of multiple deep learning architectures: ResNet, ResNeXt and DenseNet on CIFAR10 Dataset. Few approaches for improving train and test dataset accuracies are discussed and implemented.**

I. Introduction

Boundary detection and Image classification are two fundamental problems that are answered using image processing techniques. Thus, anyone looking to start learning in the field of Computer Vision needs to know about these problems and should attempt to learn different techniques required to solve the two. This document is divided into two phases in order to address the two problems with different techniques: Phase1-Probability of Boundary Technique for edge and boundary detection and Phase2-A deep learning approach for image classification on CIFAR-10 dataset.

II. Phase 1: Shake My Boundary

Boundary detection is an important, well-studied computer vision problem. Clearly it would be nice to have algorithms which know where one object transitions to another. But boundary detection from a single image is fundamentally diffcult. Determining boundaries could require object-specific reasoning, arguably making the task hard. A simple method to find boundaries is to look for intensity discontinuities in the image, also known of edges.

Classical edge detection algorithms, including the Canny and Sobel baselines we will compare against, look for these intensity discontinuities. The more recent pb (probability of boundary) boundary detection algorithm significantly outperforms these classical methods by considering texture and color discontinuities in addition to intensity discontinuities. Qualitatively, much of this performance jump comes from the ability of the pb algorithm to suppress false positives that the classical methods produce in textured regions.

In this homework, you developed a simplified version of pb, which finds boundaries by examining brightness, color, and texture information across multiple scales (different sizes of objects/image). The output of your algorithm is a per-pixel probability of boundary.Our simplified boundary detector significantly outperforms the well regarded Canny and Sobel edge detectors.

A. Overview

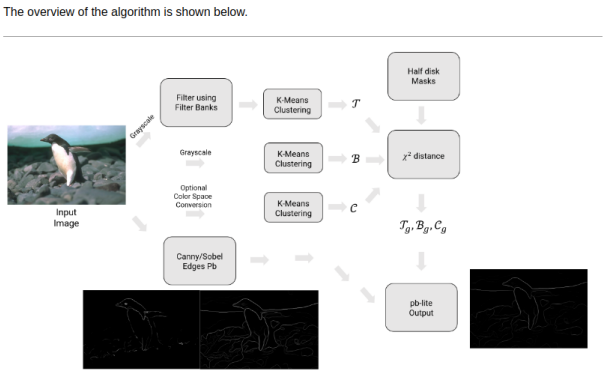


Fig. 1: Overview of the pb lite pipeline.

B. Filters

The first step of the pb lite boundary detection pipeline is to filter the image with a set of filter banks. We created three different sets of filter banks for this purpose. Once we filter the image with these filters, we generated a texton map which depicts the texture in the image by clustering the filter responses. We denoted each filter as $\mathcal{F}_\rangle$ and texton map as $\mathcal{T}$.

1) Oriented DoG filters: The Oriented Difference of Gaussian Filter is generated by taking the difference of two normal Gaussian Filters with same variance but the centers are of each is shifted by an amount equal to the standard deviation. This filter can also be created by convolving a simple Sobel Filter and a Gaussian kernel. The figure below shows the gaussian filter bank used for generating the results shown in the future sections.

The filter bank generated here is similar to the example given in the question paper i.e.,filter bank of size 2×16 with 2 scales and 16 orientations.

Fig. 2: Oriented DoG filter bank.

2) Leung-Malik Filters: The Leung-Malik filter bank is a collection 48 filters with multiple scales and orientations. It consists of first and second order derivatives of Gaussians, Laplacian of Gaussian(LOG) filters and 4 Gaussian filters. All these filters account for different types of features in the image. The filter bank is shown in the following figure.

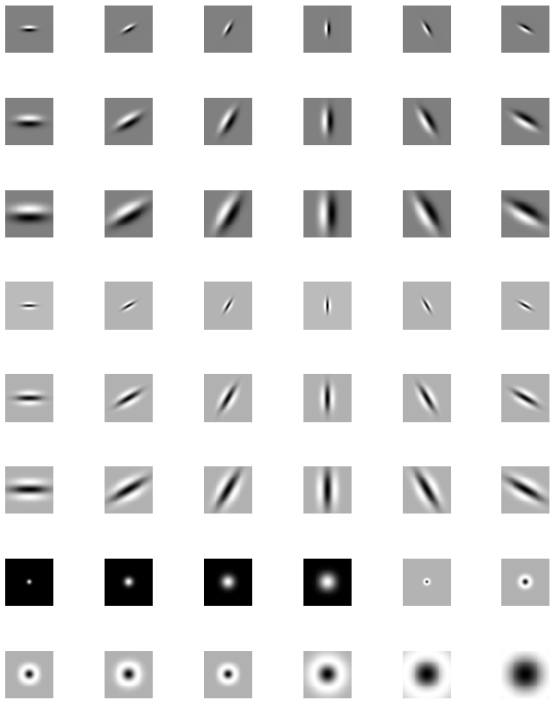


Fig. 3: Leung-Malik filter bank.

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 formation done in this code is slightly different from the original equation, this was done to reduce the number of parameters. Although an alternative filter set is also provided. So the code used for the implementation produces filter uniformly spaced from 0 to 360 in 4 equal intervals, shown in the following figure 4.

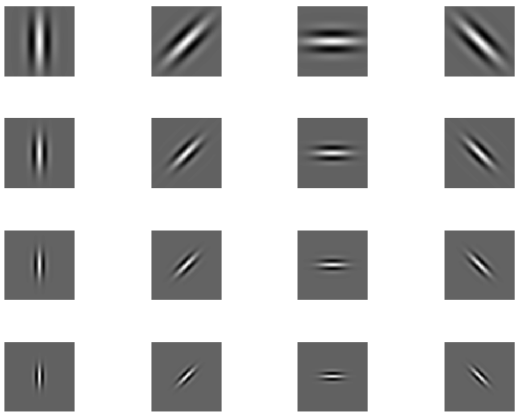


Fig. 4: Gabor filter bank used for final implementation.

With some variations in parameters in the original equation like lamda- λ = 30, sigma- σ = varying, gamma- γ = 0.25, in our case the psi- ψ is set to 0. Since it is created just for understanding the significance of parameters and not used in the final implementation, so the variation is done across 0 to 90 in 8 equal variations.

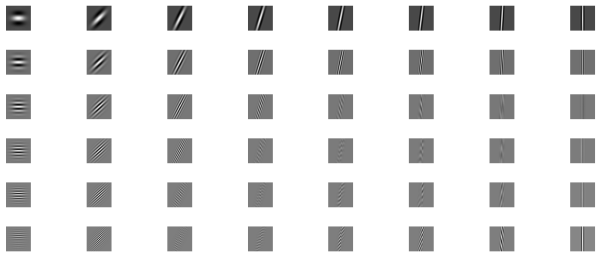


Fig. 5: Gabor filter bank Variation made but not used for final implementation.

Before applying the Maps, here are all the original images:

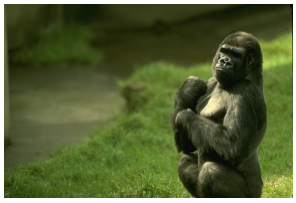
(a) image1

(b) image2

(a) image3

(b) image4

(a) image5



(b) image6

(a) image7

(b) image8

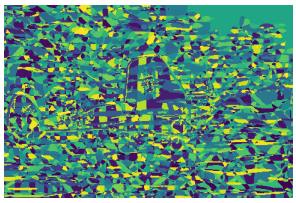
(a) image9

(b) image10

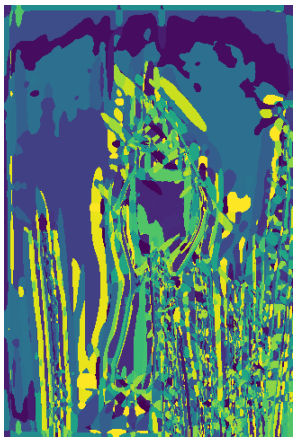
C. *Texton Map$\mathcal{T}$*

Filtering an input image with each element of out filter bank (we can have a lot of them from all the three filter banks we implemented) results in a vector of fillter responses centered on each pixel. For instance, if our filter bank has N filters, we will have N filter responses at each pixel. A distribution of these N-dimensional filter responses could be thought of as encoding texture properties. We will simplify this representation by replacing each N-dimensional vector with a discrete texton ID. We will do this by clustering the filter responses at all pixels in the image in to K textons using kmeans . Each pixel is then represented by a one dimensional, discrete cluster ID instead of a vector of high-dimensional, real-valued filter responses (this process of dimensionality reduction from N to 1 is called "Vector Quantization"). This can be represented with a single channel image with values in the range of [1,2,3,$\cdots$,K]. K = 64 seems to work well but feel free to experiment. To visualize the texton map, we used matplotlib.pyplot.imshow command with proper scaling arguments.

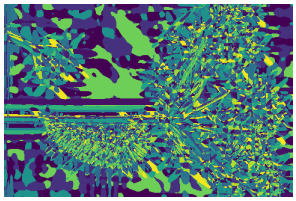
Here are the tmaps of all the images given in the dataset:



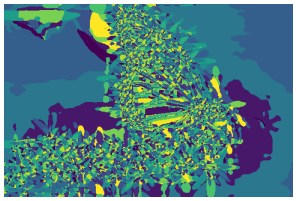
(a) image1 tmap



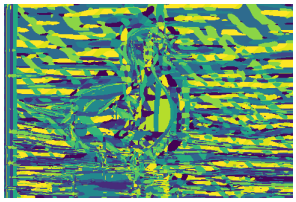
(b) image2 tmap



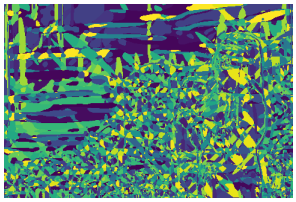
(a) image3 tmap



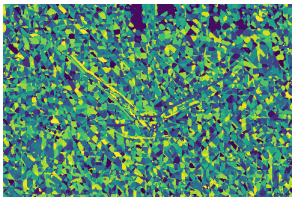
(b) image4 tmap



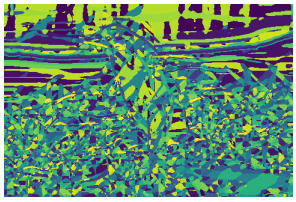
(a) image5 tmap



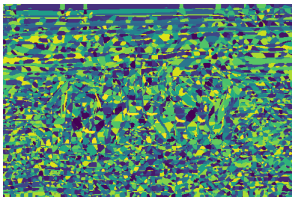
(b) image6 tmap



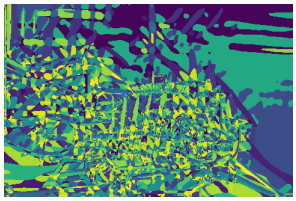
(a) image7 tmap



(b) image8 tmap



(a) image9 tmap

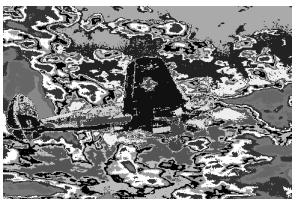


(b) image10 tmap

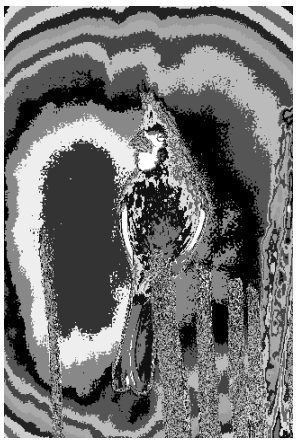
Fig. 6: Tmap images of all the images in the dataset

D. Brightness Map$\mathcal{B}$

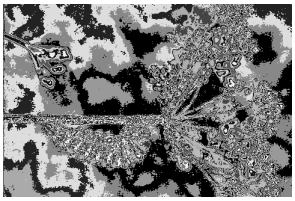
The concept of the brightness map is as simple as capturing the brightness changes in the image. Here, again we cluster the brightness values using kmeans clustering (grayscale equivalent of the color image) into a chosen number of clusters (16 clusters seems to work well). We call the clustered output as the brightness map $\mathcal{B}$. Here are the bmaps of all the images given in the dataset:



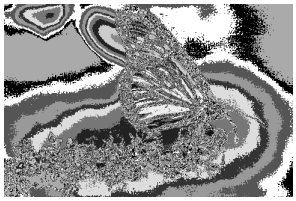
(a) image1 bmap



(b) image2 bmap



(a) image3 bmap

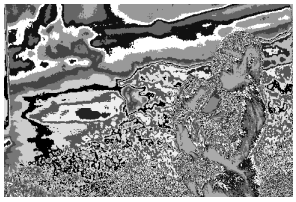


(b) image4 bmap

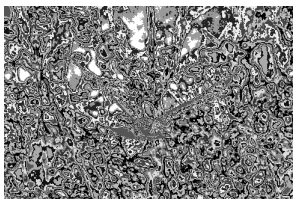
E. Color Map$\mathcal{C}$

The concept of the color map is to capture the color changes or chrominance content in the image. Here, again we cluster

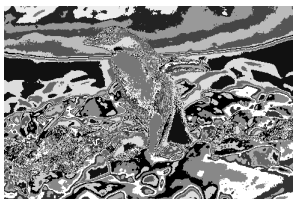
(a) image5 bmap



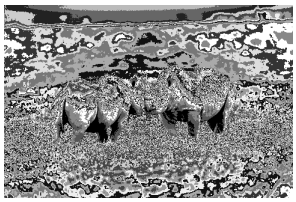
(b) image6 bmap



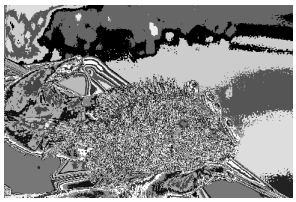
(a) image7 bmap



(b) image8 bmap



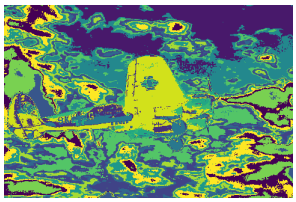
(a) image9 bmap



(b) image10 bmap

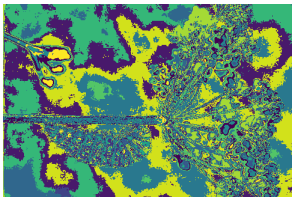
Fig. 7: Bmap images of all the images in the dataset

the color values (we have 3 values per pixel if we have RGB color channels) using kmeans clustering into a chosen number of clusters (16 clusters seems to work well). We call the clustered output as the color map $\mathcal{C}$. It should be noted that we can also cluster each color channel seprarately here.

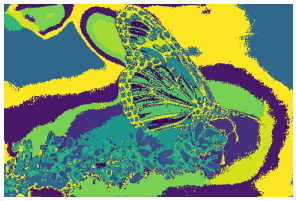


(a) image1 cmap

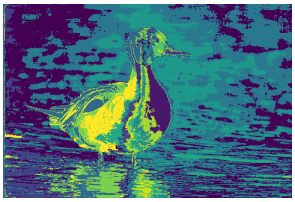
(b) image2 cmap



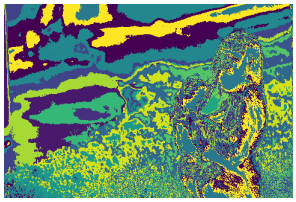
(a) image3 cmap



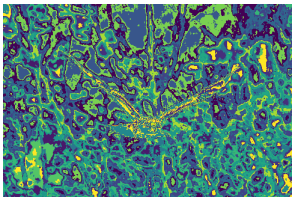
(b) image4 cmap



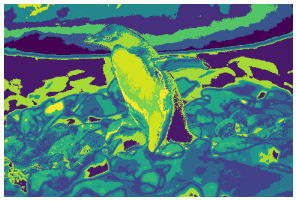
(a) image5 cmap



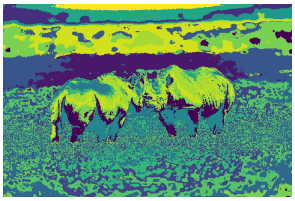
(b) image6 cmap



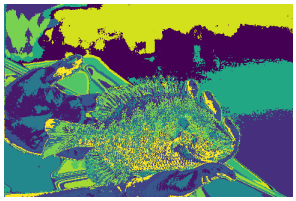
(a) image7 cmap



(b) image8 cmap



(a) image9 cmap



(b) image10 cmap

Fig. 8: Cmap of all the images in the dataset

F. Texture, Brightness and Color Gradients $\mathcal{T}_g$,$\mathcal{B}_g$,$\mathcal{C}_g$

To obtain $\mathcal{T}_g$,$\mathcal{B}_g$,$\mathcal{C}_g$ 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-disks are generated by multiplying an array of size equal to the radius/scale of the circular disk with all values which lie inside this radius equal to 1 and rest 0, with an array of equal size but where one half of the array is 0s and the other half consists of 1s. This multiplication results in a half-disk which can be rotated to produce the desired half-disk mask. It should be noted that if we rotate the disk after multiplying the two arrays, it will result in pixel voids. This can be avoided by rotating the rectangular block matrix of 0s and 1s and then by applying a "logical OR" operator on them. This gives the required half-disk masks.

Using the above generated Half-Disk masks we compute the $\mathcal{T}_g$,$\mathcal{B}_g$ and $\mathcal{C}_g$ maps by comparing the distributions generated using each half-disk pair with a χ^2 measure. The binning scheme is defined for K indexes which is equal to the number of K-Means clusters for each $\mathcal{T}_g$,$\mathcal{B}_g$ and $\mathcal{C}_g$. This

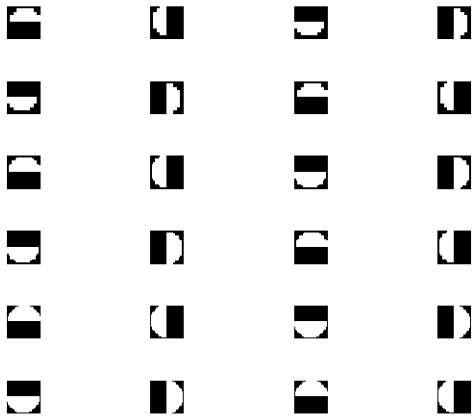
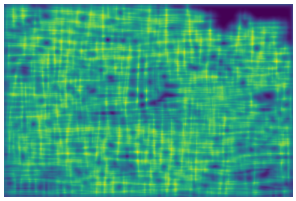


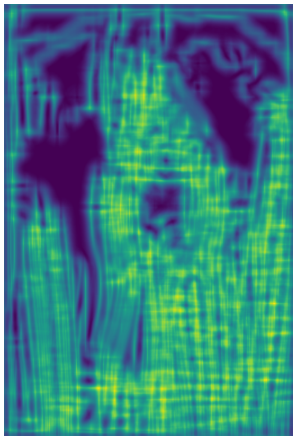
Fig. 9: Half disc masks at different scales and sizes.

procedure is repeated for all the half-disk pairs to generate a 3D matrix of size m×n×N where m, n are the dimensions of the image and N is the number of filters. The values of K for texton, brightness and color are 64, 16 and 16 respectively.

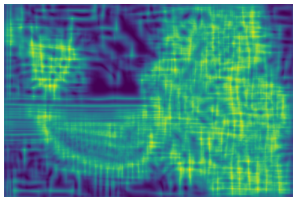
Following are the $\mathcal{T}_g$ of all the images:



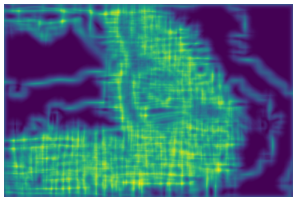
(a) image1 $\mathcal{T}_g$



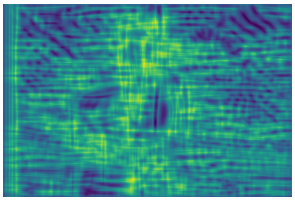
(b) image2 $\mathcal{T}_g$



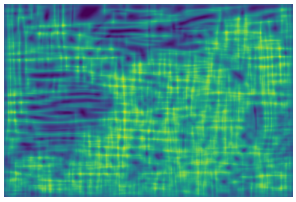
(a) image3 $\mathcal{T}_g$



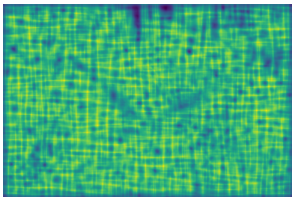
(b) image4 $\mathcal{T}_g$



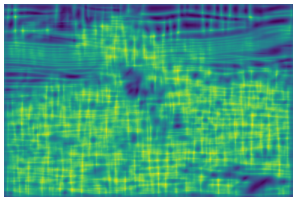
(a) image5 $\mathcal{T}_g$



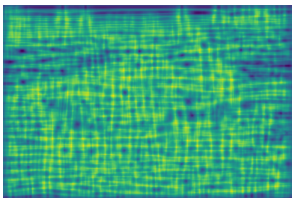
(b) image6 $\mathcal{T}_g$



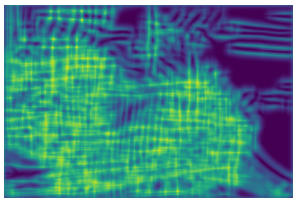
(a) image7 $\mathcal{T}_g$



(b) image8 $\mathcal{T}_g$

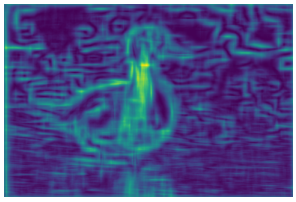


(a) image9 $\mathcal{T}_g$

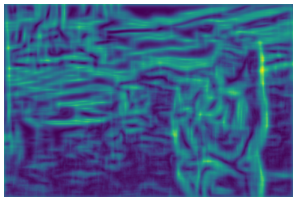


(b) image10 $\mathcal{T}_g$

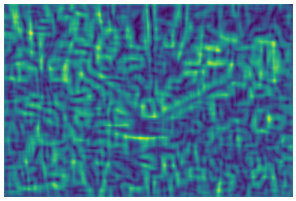
Fig. 10: $\mathcal{T}_g$ of all the images in the dataset



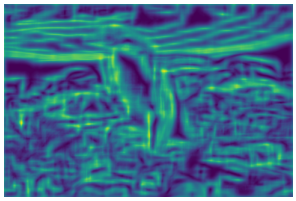
(a) image5 $\mathcal{B}_g$



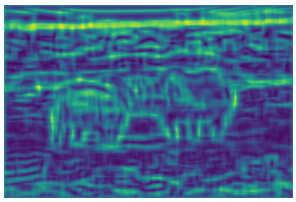
(b) image6 $\mathcal{B}_g$



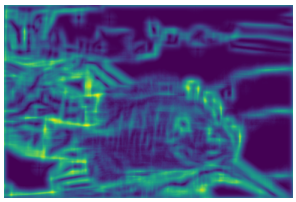
(a) image7 $\mathcal{B}_g$



(b) image8 $\mathcal{B}_g$



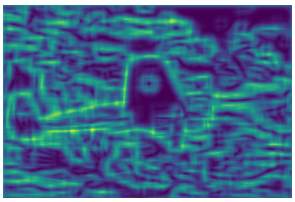
(a) image9 $\mathcal{B}_g$



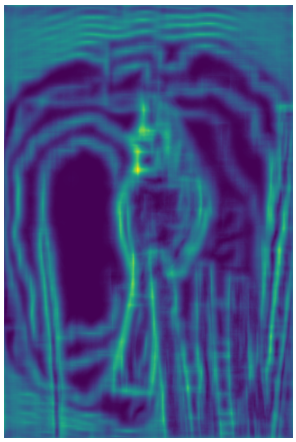
(b) image10 $\mathcal{B}_g$

Fig. 11: $\mathcal{B}_g$ of all the images in the dataset

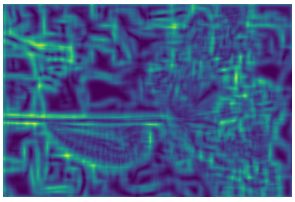
Following are the $\mathcal{B}_g$ of all the images:



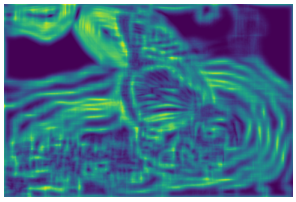
(a) image1 $\mathcal{B}_g$



(b) image2 $\mathcal{B}_g$

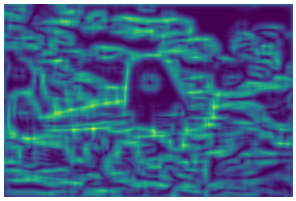


(a) image3 $\mathcal{B}_g$

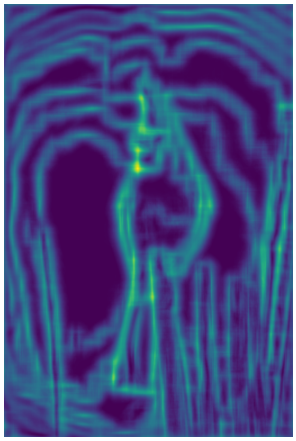


(b) image4 $\mathcal{B}_g$

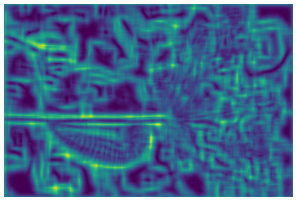
Following are the $\mathcal{C}_g$ of all the images:



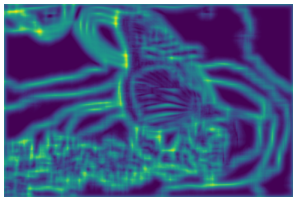
(a) image1 $\mathcal{C}_g$



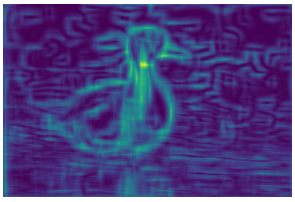
(b) image2 $\mathcal{C}_g$



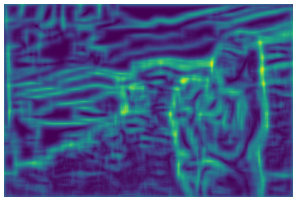
(a) image3 $\mathcal{C}_g$



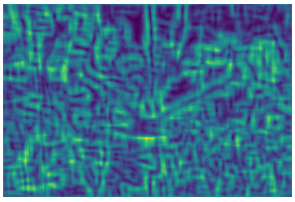
(b) image4 $\mathcal{C}_g$



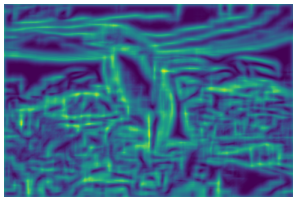
(a) image5 $\mathcal{C}_g$



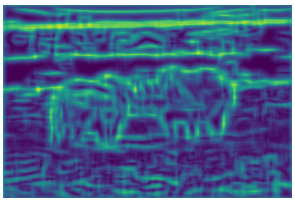
(b) image6 $\mathcal{C}_g$



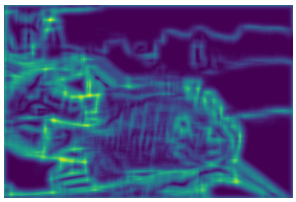
(a) image7 $\mathcal{C}_g$



(b) image8 $\mathcal{C}_g$



(a) image9 $\mathcal{C}_g$



(b) image10 $\mathcal{C}_g$

Fig. 12: $\mathcal{C}_g$ of all the images in the dataset

G. Pb-lite Output

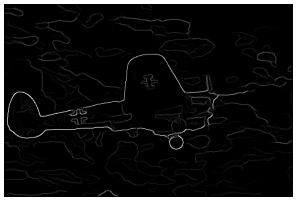
Finally, the gradient maps generated are combined with Canny and Sobel baselines using the equation:

$$PbEdges = \frac{\mathcal{T}_g + \mathcal{B}_g + \mathcal{C}_g)}{3} \odot (W_1 * cannyPb + W_2 * sobelPb) \tag{1}$$

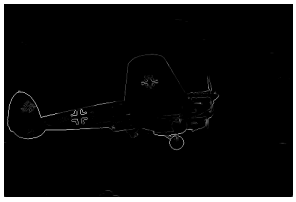
where $W_1 = W_2 = 0.5$

These values can be altered to get different results, we may attain better results or worse results based on the weights we choose.

Here are the canny, sobel and Pb-Lite outputs for all the images.



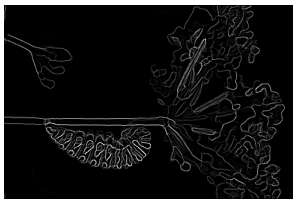
(a) canny



(b) sobel

(c) Pb-lite

Fig. 13: Image1



(a) canny

(b) sobel

(c) Pb-lite

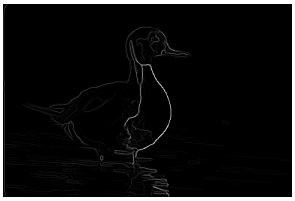
Fig. 14: Image2

(a) canny

(b) sobel

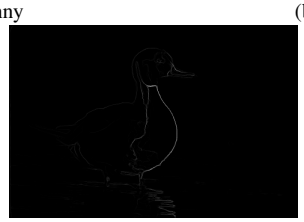
(c) Pb-lite

Fig. 15: Image3



(a) canny

(b) sobel



(c) Pb-lite

Fig. 16: Image4

(a) canny

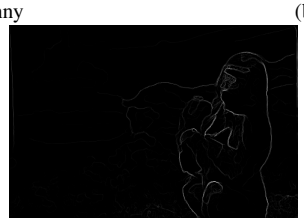
(b) sobel

(c) Pb-lite

Fig. 19: Image7

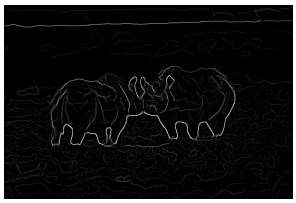
(a) canny

(b) sobel



(c) Pb-lite

Fig. 17: Image5



(a) canny

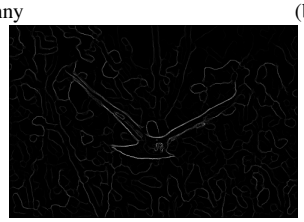
(b) sobel

(c) Pb-lite

Fig. 20: Image8

(a) canny

(b) sobel



(c) Pb-lite

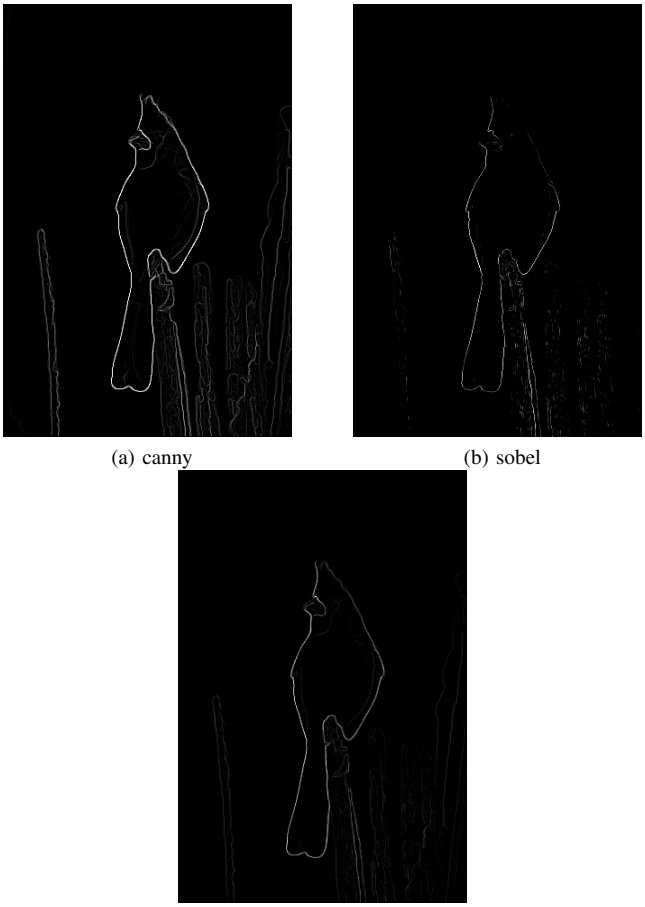
Fig. 18: Image6

(a) canny

(b) sobel

(c) Pb-lite

Fig. 21: Image9



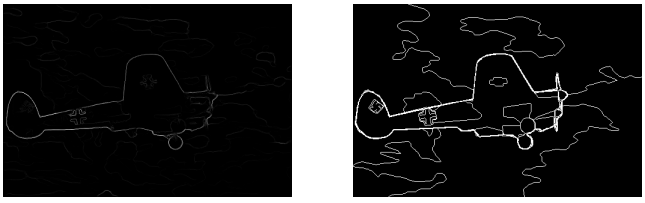
(a) canny (b) sobel

(c) Pb-lite

Fig. 22: Image10

H. Discussion/Conclusion for Phase1

While computing the PB-Lite output it was observed that changing just the weights of the canny and sobel baselines produced significant changes in the output image which can be seen in the two images.



(a) Pb-Lite of image 1 (b) Pb-Lite of image 1

Fig. 23: Effect of weights on images

This happens due to the fact that Canny baseline also includes weaker unwanted edges due to the averaging operation performed while calculating the canny baseline.So basically what we are doing is eliminating the false positive pixels. The weights are arbitrarily taken as advised in the question paper for this homework implementation i have used w1 = w2 = 0.5 as default.

III. Phase 2: Deep Dive on Deeplearning

This secton is all about implementing different neural networks on the given dataset CIFAR-10. The aim is to sucessfully implement the architecures and attempt to improve the accuracies. It should be noted that due to some hardware constraints i am not able to provide the datas for more epocs since it takes a lot of time to runm, so a base of 5 epoc is considered and minibatch size of 5. There are many more scopes of improvement in the given architectures but i was not able to explore them since i do not have RTX gpu driver in ubuntu 16.04. Therefore my prime focus was to implement all these networks and bring out some sort of acceptable accuracy rate to pull out some sort of explanation for the dataset.

CIFAR-10 is a dataset consisting of 60000, 32×32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

Fig. 24: Sample images from CIFAR-10 dataset.

A. Convolutional Neural Network (CNN)

This is the first neural network - Convolutional Neural Network. This CNN consists of four convolution layer followed by a pooling layer and a fully connected layer. Softmax layer is added at the end after the fully-connected layer. Given is the hyper parameter list.

Batch Size	5
Learning rate	0.001
Epochs	20
Filter size	5x5, 3x3
Number of filters	20, 40, 80
Activation layers	Relu
Stride length	1

Filter size of 5x5 is used in the first convolution layer and then 3x3 is used in the subsequent layers. Stride length is kept 1 in each convolution. ReLU activation and batch norm is done after each and every convolution layer.Softmax layer takes input the output of the fully connected layer. Filter size of 5x5 is used in the first convolution and filter is size of 3x3 is used in the subsequent convolution layers. Stride length is kept 1 in the entire network.

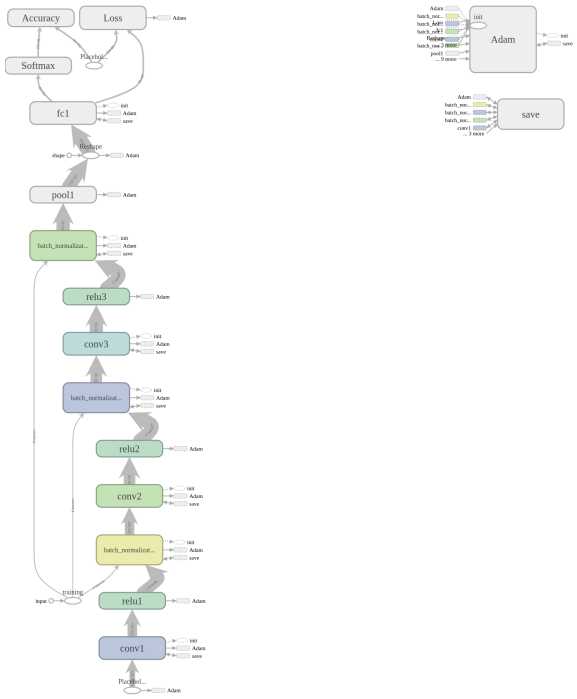
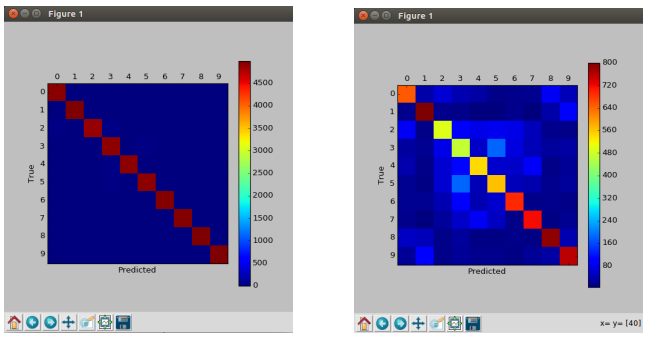


Fig. 25: CNN - Architecture.

Here is the confusuin matrix after applying the trained model on test and train images.When applied on the test images it provided an accuracy of 65% and when applied on the training images it gave an accuracy of 99.16%.



(a) CNN - Train (b) CNN - Test

Fig. 26: Confusion Matrix for CNN

The following figure shows the accuracy and loss graph per epoch for the CNN.

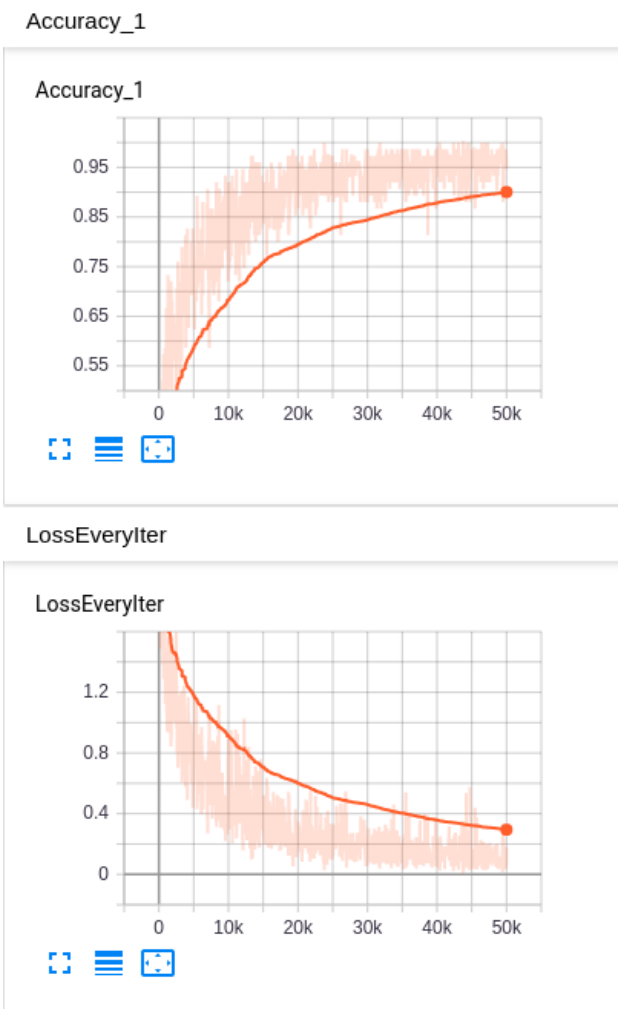


Fig. 27: CNN - Loss and Accuracy Graph.

The accuracy can be increased by increasing the number of epochs, changing the filter size and number of filters as well. But there will be a time when the accuracy will not increase and to the contrary the accuracy might decrease for the test images due to over fit.

B. *Residual Neural Network (ResNet)*

With the same standardisation and data augmentation in CNN we now implement ResNet architecture. A Residual Network, or ResNet is a neural network architecture which solves the problem of vanishing gradients in the simplest way possible, i.e., by applying skip connections in a general residual block. This allows the network to accommodate deep layers without having the vanishing gradient problem.

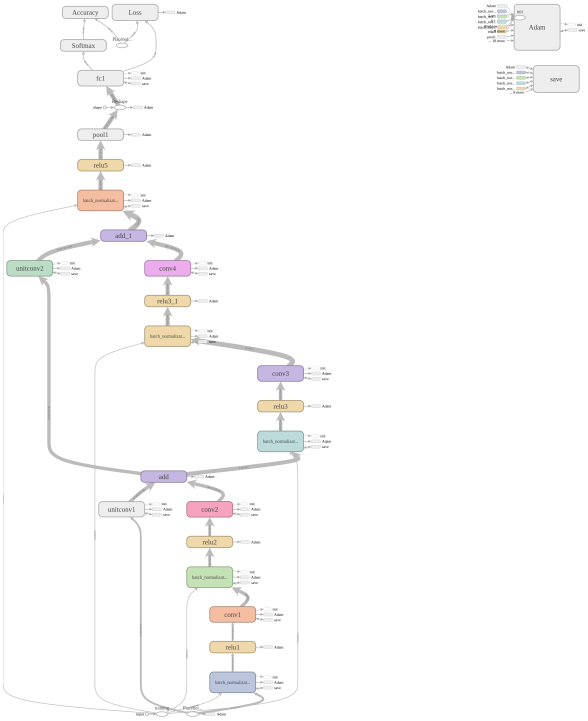


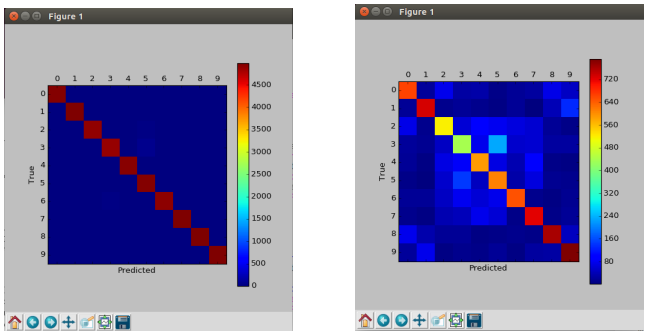
Fig. 28: RNN - Architecture.

This ResNet consists of four convolution layers and two skip-connections. Each skip connection contains two convolution networks. Each convolution layer is done after performing batch-norm and RELU activation layer respectively. After four convolution there is a max pooling layer and a fully connected layer. Softmax layer takes input the output of the fully connected layer. Filter size of 5x5 is used in the first convolution and filter is size of 3x3 is used in the subsequent convolution layers. Stride length is kept 1 in the entire network.

The Hyper parameters are the following:

Batch Size	5
Learning rate	0.001
Epochs	20
Filter size	5x5, 3x3
Number of filters	20, 40, 80
Activation layers	Relu
Stride length	1

After training the model for 5 epochs the model is applied on the training as well as test images. In the test images the accuracy was around 85% and on train images it was around 99%. Thus we can see in CNN i.e., after 20 epochs the accuracy was almost identical while in ResNet it only took 5 epochs to get to this accuracy.



(a) RNN - Train (b) RNN - Test

Fig. 29: Confusion Matrix for RNN

The following figure shows the accuracy and loss for each epochs for ResNet architecture.

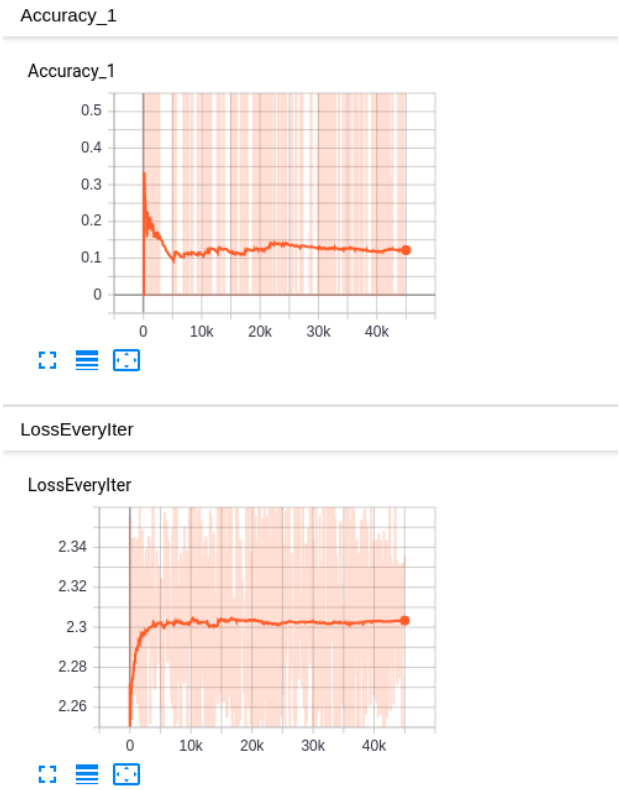


Fig. 30: RNN - Loss and Accuracy Graph.

The accuracy can be improved with increasing the number of epocs, changing the number of augumented images, changing the number of filters and their sizes.

C. ResNext

ResNext is an improvement on ResNet architecture which can be seen. Resnet with 20 epocs have around 65% accuracy where as this network with only 5 epocs produced an accuracy of 59% on the test images. In addition to utilizing the concept of residual learning framework from ResNet, the concept of ”Cardinality” is also introduced in this network. Cardinality is the size of the set of split transformations that an input goes through before being passed to the fully connected layers.In ResNext architecture, the input is split into different paths (i.e.,number of split paths is equal to the cardinality) and convolutions are performed in each of these split paths. The outputs of these split layers are then concatenated with the input followed by application of non-linearity. It is empirically shown in the paper that even under the restricted condition of maintaining complexity, increasing cardinality is able to improve classification accuracy. Moreover, increasing cardinality is more effective than going deeper or wider when we increase the complexity of the network.

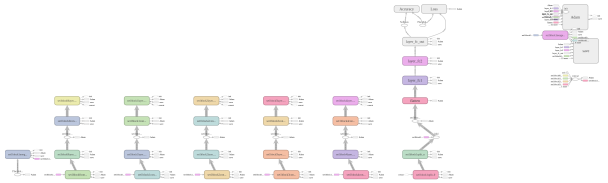
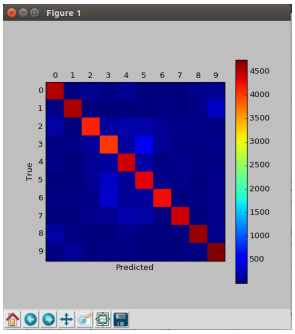


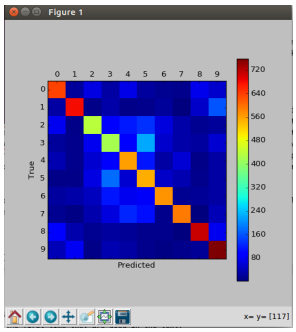
Fig. 31: ResNext - Architecture.

For this Homework ResNext architecture implemented is a single residual layer with a cardinality of 5. Each cardinal(split)layer has 2 convolution layers. The outputs from these layers are added instead of being concatenated and then passed through another convolution layer. The output is then added to the original input and fed through a Relu activation layer to the fully connected layers.

The confusion matrices are provided below. Once the model is trained, the model is applied to the training as well as testing images to test the accuracy. On the training images the accuracy is 88.124% and on the testing images it is 59.2%.



(a) ResNext - Train



(b) ResNext - Test

Fig. 32: Confusion Matrix for ResNext

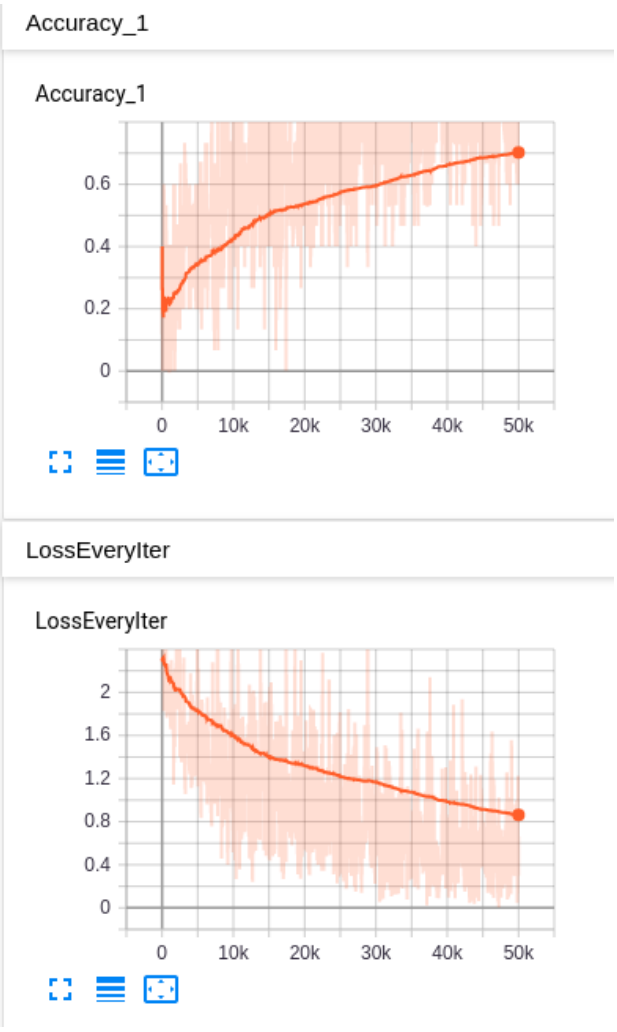


Fig. 33: ResNext - Loss and Accuracy Graph.

The accuracy can be improved with increasing the number of epocs, changing the number of augumented images, changing the number of filters and their sizes.

D. DenseNet

As from the previous sections we can seen that the accuracy and efficiency of the CNNs can be substantially improved by making shorter connections between layers close to the input and those close to the output. In DenseNet each layer connects to every other layer in a feed-forward fashion.This solves the vanishing gradient problem and allows stronger feature propagation. This leads to a decreased number of features without compromising the training and testing accuracy

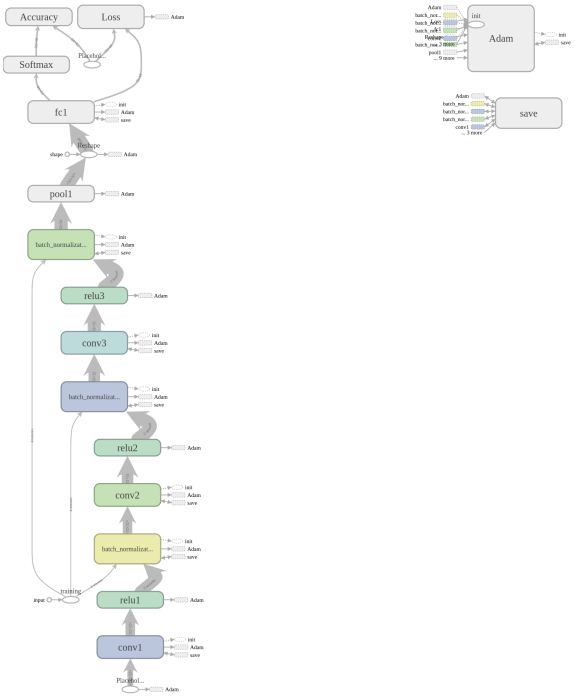
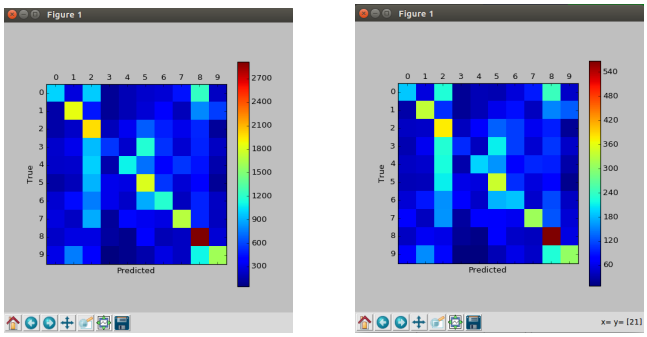


Fig. 34: DenseNet - Architecture.

For this Homework, the DenseNet architecture implemented contains 4 convolution layers in the dense block. The input image is first convoluted and the output is feeded to all the convolution layers in succession. This process is repeated for all the convolution layers. The output is then finally concatenated and passed through an activation layer after which it is sent on to the fully connected layers. The loss over epochs, training and test accuracy over epochs can be seen in the upcomming figures.

The confusion matrices are provided below. Once the model is trained, the model is applied to the training as well as testing images to test the accuracy. On the training images the accuracy is 39.43% and on the testing images it is 29.15%. It can be seen that the accuracies are very low, which i think van be improved with increasing the number of epocs.



(a) DenseNet - Train (b) DenseNet - Test

Fig. 35: Confusion Matrix for DenseNet

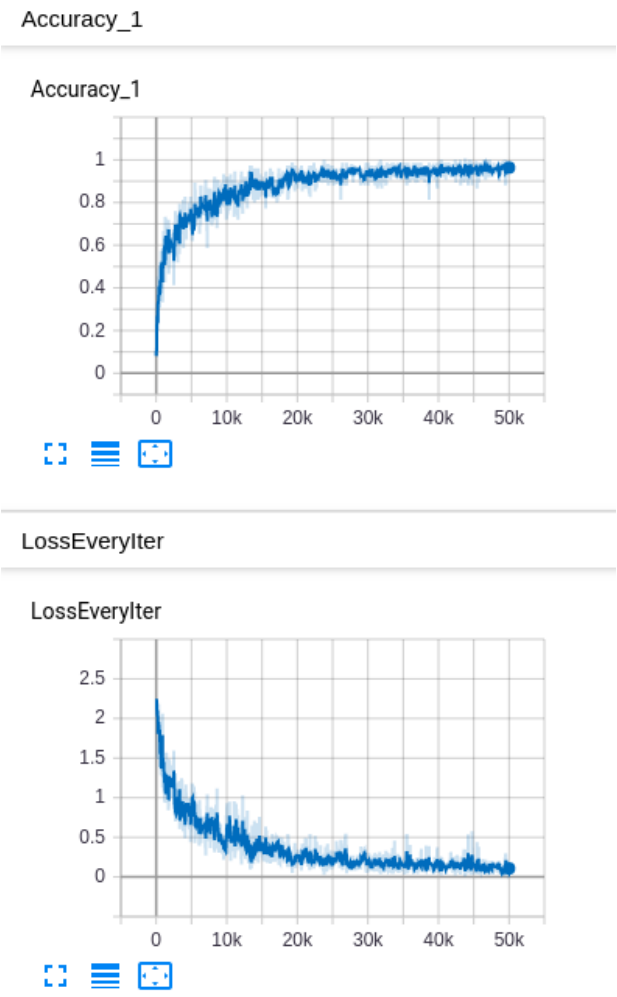


Fig. 36: DenseNet - Loss and Accuracy Graph.

The accuracy can be improved with increasing the number of epocs, changing the number of augumented images, changing the number of filters and their sizes.

E. Discussion/Conclusion for Phase1

In this Homework we implemented different neural network architectures on CIFAR-10 dataset to develop an image classifier models. Architecture flowscharts, confusion matrices and accuracy/loss per epoc for every network is provided in the respective sections. Thess datas help to draw conclusion for each networks.

Every network has its own advantages and disadvantages. A simple CNN takes moderate time (smaller than all others) to train but the test results are not very accurate. Theoretically it can be said that by making the CNN deeper the accuracy of the network can be improved, but it is not the case. As we increase the number of convolution layers the complexity of the network increases and after a certain limit, the accuracy of this network decreases due to "Overfitting". To counter this problem these different types of architectures were formed. These new networks avoid the overfitting problem in their own ways: by adding skip connections(ResNet), by introducing Cardinality(ResNext) or by feed-forwarding the input directly to every other layer in succession(DenseNet).

The accuracies attained are still very limited. All the architectures that are discussed above rely on a supervised learning approach,which implicitly assums that there is enough data to train these models. The dataset given to us was a trimmed down version of the CIFAR-10 dataset. So, we applied techniques for data augmentation like geometrically transforming the image, cropping etc to generate more data so that the models could be trained better. But there can't ever be enough data. But adhering to the positives of such models, they still produce better results in terms of image classification than traditional image processing approaches.

REFERENCES

[1] Homework question paper.

[2] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik, Contour Detection and Hierarchical Image SegmentationIEEE Trans. Pattern Anal. Macch. Intell. 33, 5 (May 2011), 898-916 Knuth: Computers and Typesetting, http://dx.doi.org/10.1109/TPAMI.2010.161

[3] Github:*Open source code help*.

[4] Tensorflow Tutorials: https://www.tensorflow.org/tutorials.

[5] Gabor filter: https://medium.com/@anuj$_s hah/through-the-eyes-of-gabor-filter-17d1fdb3ac97$