Homework 0 - Alohomora

Shiva Kumar Tekumatla
MS Robotics Engineering
Worcester Polytechnic Institute
stekumatla@wpi.edu

***Abstract*— In this work, I presented the simplified version of Probability of Boundary (pb lite) detection algorithm, and built Convectional Neural Network(CNN) algorithm for Image classification. Results of pb lite are compared with Canny, and Sobel edge detection algorithms. Overall proved that pb lite outperforms the Sobel and Canny for edge detection. To train and test CNNs , I used CIFAR-10 dataset , and explored different methods such as ResNet, ResNeXt and DenseNet to improve the classification accuracy.**

I. INTRODUCTION

In computer vision , detecting boundaries in a given image has been a well studied problem that can determine the various transitions from one a layer to an another in a given image. Detecting boundaries using one image is a difficult problem to solve, and generally detecting boundaries can require object-specific reasoning, and this may make this process difficult. A well presented solution to this problem is to study intensity discontinuities in the given image. These discontinuities are usually called edges.

Two classical methods to solve this problem,Canny and Sobel edge detection algorithms, look for these discontinuities. In this work, these two classical algorithms are compared against Probability of Boundary detection algorithm or simply called PB edge detection algorithm. In this work, I proved that PB algorithm outperforms both the aforementioned classical algorithms. Main advantage of this algorithm is that it considers texture and color discontinuities in the given image, and helps in suppressing false positives. These false positives produced in the textured regions are not usually handled by classical edge detection algorithms.

In phase 1 of this work,I developed a simplified version of PB algorithm that is called pb lite. pb lite uses brightness, color and texture information across different scales, and provides a per-pixel probability of boundary. Even thissimplified boundary detector will still significantly outperform the well regarded Canny and Sobel edge detectors. In this work, a evaluated the ground truth qualitatively from a subset of the Berkeley Segmentation Data Set 500 (BSDS500).

In phase 2 of this work, I implemented multiple neural network architectures and compared them using different criterion such as number of parameters, train and test accuracies. For this work, I used CIFAR-10 dataset. This 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.

II. PHASE 1 - SHAKE MY BOUNDARY

There are multiple steps in the design pipeline of pb lite algorithm. These steps are: 1) Creating Filter Banks 2) Texton Map 3) Brightness Map 3) Color Map 4) Texture, Brightness and Color Gradients. The general pipeline is given by Figure 1 .

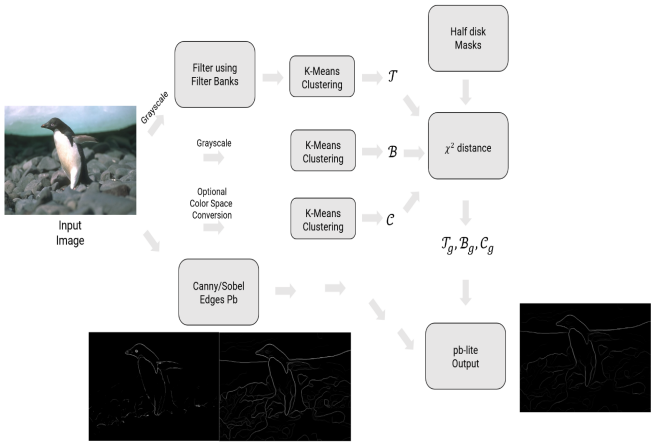

Fig. 1. Overview of the pb lite pipeline

A. Filter Banks

The first step of the pb lite boundary detection pipeline is to filter the image with a set of filter banks. I created three different types of filterbanks in this step. They are : 1) Oriented DoG Filters 2)Leung-Malik Filters 3) Gabor Filters. A given image is filtered using either with ine of these filters or with a combination of them. After this step, a texton map that can depict the texture in the image by clustering responses with respect to each filter is generated.

Since this is the first step in finding out the edges , this step is an important part of building low level features that we are interested in. These filter banks are used for two purposes, to measure texture properties and to aggregate regional texture and brightness distributions.

1) Oriented DoG Filters: Oriented Difference of Gaussian (DoG) filters are simply the Gaussian filters convolved with Sobel operators, and then rotating the results. Usually oriented DoG filters are generated with α orientations (from 0 to 360 degrees) and β scales, we should end up with a total of $\alpha \times \beta$ filters. In this work, I used three different scales with

16 orientations each. The scales used are :1,1.5,2 . Result of this filter bank is given by Figure 2 .

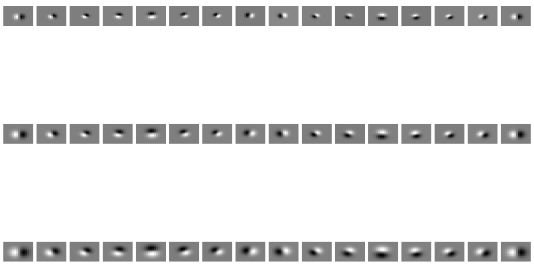

Fig. 2. Oriented DoG Filter Bank

2) Leung-Malik Filters: There are a total of 48 filters in the Leung-Malik(LM) filter bank. This filter bank consists of first and second order derivates of Gaussian filters at six different orientations, and three scales, eight Laplacian of Gaussian (LoG) filters , and four Gaussian filters. There are two different version of LM filters , namely LM Small (LMS) and LM Large (LML). In LMS, scales used are: 1, $\sqrt{2}$,2, and $2\sqrt{2}$ where as in LML , scales used are $\sqrt{2}$, 2, $2\sqrt{2}$ and 4. Result of LM filter bank is shown in Figure 3 .

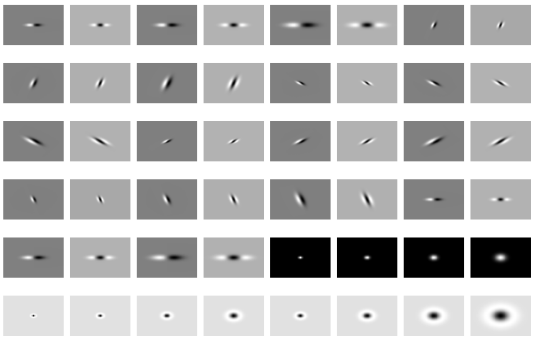

Fig. 3. Leung-Malik filter bank

3) Gabor Filters: Gabor filter is a type of linear system used for texture analysis . This filter works on basis of analysing specific frequency content in the image in specific directions in a localized region around the point or region of analysis. In this work I created a total of 25 Gabor filters. This result is shown by Figure 4 .

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

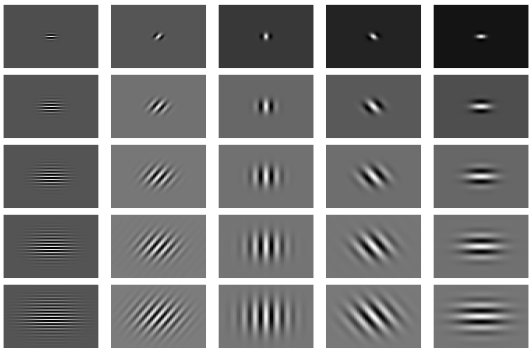

Fig. 4. Gabor filter bank

Fig. 5. Input Images

B. Texton Map

Filtering an input image with each element of your filter ban results in a vector of filter responses centered on each pixel. We can cluster each pixel by using kmeans clustering algorithm. Each pixel is then represented by a one dimensional, discrete cluster ID instead of a vector of high-dimensional, real-valued filter responses. For this previously generated Oriented DoG filter bank is used.Three different input images shown in Figure 5 are used for subsequent operations. Result of texton map on different images is shown in Figure 6 .

C. Brightness Map

The concept of the brightness map is as simple as capturing the brightness changes in the image. Here, I clustered the brightness values using kmeans clustering. I chose a total of 16 clusters. Results of brightness maps are shown in Figure 7 .

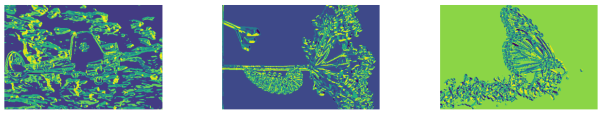

Fig. 6. Results of Texton Map

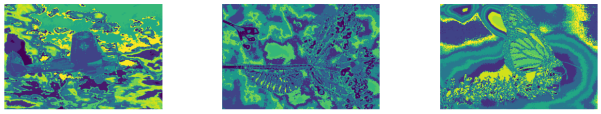

Fig. 7. Results of Brightness Map

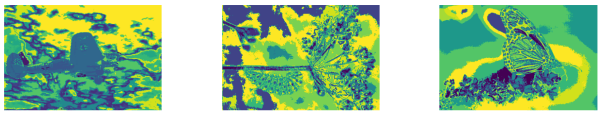

Fig. 8. Results of Color Map

D. Color Map

The concept of the color map is to capture the color changes or chrominance content in the image. Here, I again clustered the color values using kmeans clustering into 16 clusters. Results of color map are shown in Figure 8

E. Half Disk Masks

The half-disc masks are simply (pairs of) binary images of half-discs. These discs allow us to compute the χ^2 (chi-square) distances (finally obtain values of gradients) using a filtering operation, which is much faster than looping over each pixel neighborhood and aggregating counts for histograms. Half disc masks at different scales and sizes are shown in Figure 9 .

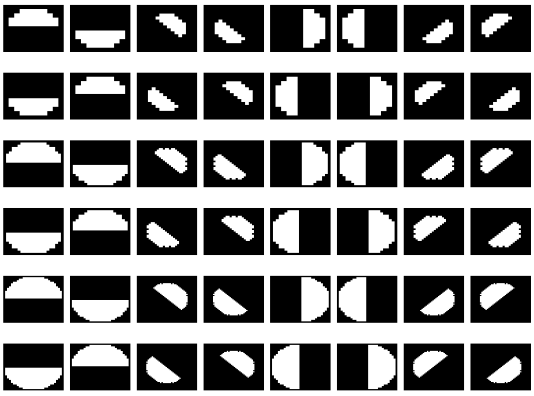

Fig. 9. Half Disk Masks

F. Texture, Brightness and Color Gradients

Texture , Brightness and Color gradients encode how much the texture, brightness and color distributions are changing at a pixel. We compute these by comparing the distributions in left/right half-disc pairs centered at a pixel. If the distributions are the similar, the gradient should be small. If the distributions are dissimilar, the gradient should be large. Because our half-discs span multiple scales and orientations, we will end up with a series of local gradient measurements encoding how quickly the texture or brightness distributions are changing at different scales and angles. The results of these gradients are shown by Figures 10, 11, and 12.

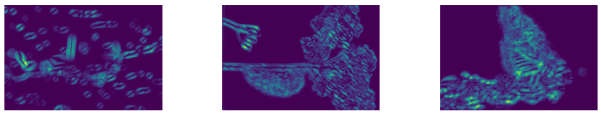

Fig. 10. Results of Texture Gradient

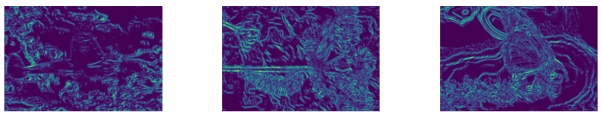

Fig. 11. Results of Brightness Gradient

G. Sobel and Canny baselines

Already provided Sobel and Canny Baselines are given by Figure 13, and 14 respectively.

H. Pb-lite Output

Output from each gradient is combined with Sobel and Canny baselines to get Pb-lite output. This output can be computed using equation 1 .

$$Pb = \frac{(T_g + B_g + C_g)}{3} \circ (w_1 \times canny + w_2 \times sobel) \quad (1)$$

The result of pb lite is given by Figure 15.

I. Observations

Based on the results provided here , we can say that pb lite performed slightly better than the other two baselines in certain aspects. For example, for images with more texture, both Sobel and Canny generated too many false positives while the pb lite suppressed these false positives. Overall , pblite provided the soft edges while the other two have comparatively rough edges. However, in some cases it seems that canny and Sobel outperformed the pb lite algorithm. This can be corrected by experimenting with the various varibles that are used in the algorithm design such as scale , size , number of clusters and radius of Half Disk Masks.

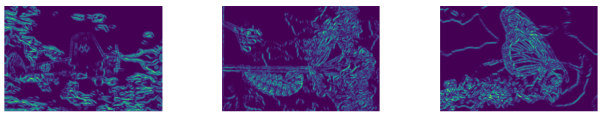

Fig. 12. Results of Color Gradient

Fig. 13. Sobel edge detection baseline

Fig. 14. Canny edge detection baseline

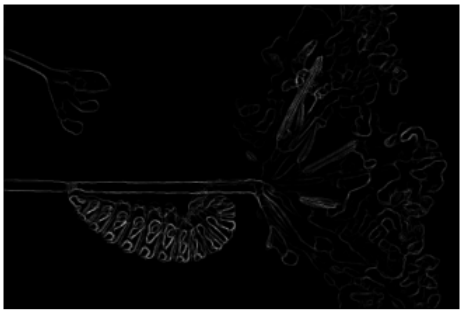

Fig. 15. Pb lite output

III. Phase 2 - Deep Dive on Deep Learning

In this section , different neural netwrok architecture are impemented. A randomized version of the CIFAR-10 dataset with 50000 training images and 10000 test images is used for this part.

A. Train your first neural network

For this section , I used the given dataset and trained it with the CNN that is of the structure *input - conv - pool - conv - pool - linear - linear -output* . The training loss, and accuracy are shown in Figure 16, and 17 respectively.

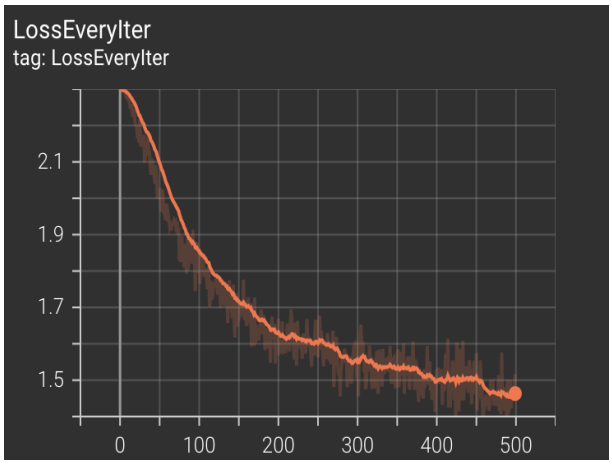

Fig. 16. Training error

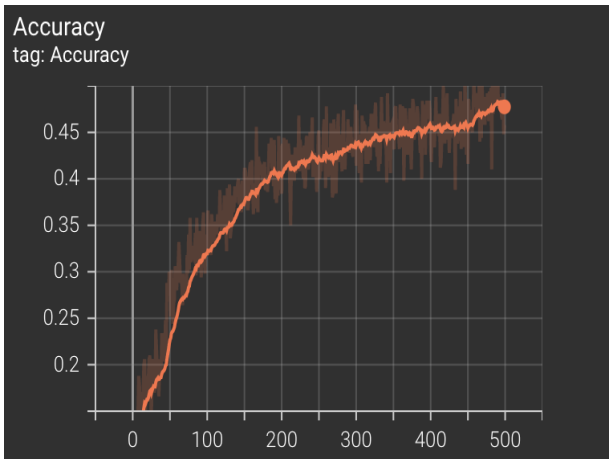

Fig. 17. Training accuracy

As observed here , the above defined neural network did not provide good results. The hyper parameters tried here are epoch size of 5 , and batch size of 500. This trained model did not perform well on test data either. This model gave an overall accuracy of 10.8 percentage. The confusion matrix is given by Figure 18.

By changing the output layers of first convolution network, there accuracy improved to 46.78 percentage. The results for this are shown by Figures 19 , 20 and **??**.

```
[    7    0  993    0    0    0    0    0    0    0]
[    1    0  999    0    0    0    0    0    0    0]
[    1    0  999    0    0    0    0    0    0    0]
[    0    0 1000    0    0    0    0    0    0    0]
[    1    0  999    0    0    0    0    0    0    0]
[    0    0 1000    0    0    0    0    0    0    0]
[    0    0 1000    0    0    0    0    0    0    0]
[    0    0 1000    0    0    0    0    0    0    0]
[    0    0 1000    0    0    0    0    0    0    0]
[    0    0 1000    0    0    0    0    0    0    0]
```

Fig. 18. Confusion Matrix

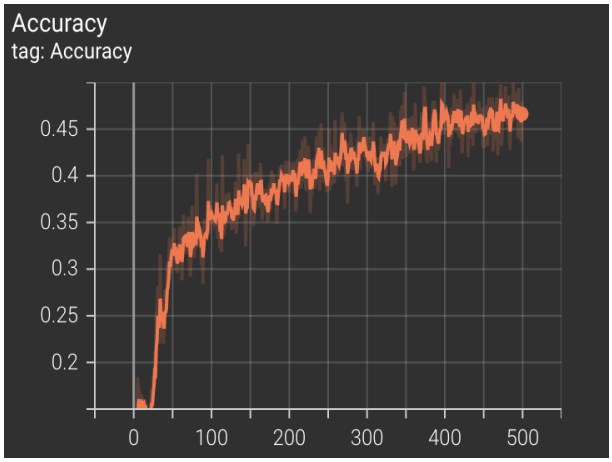

Fig. 19. Training accuracy

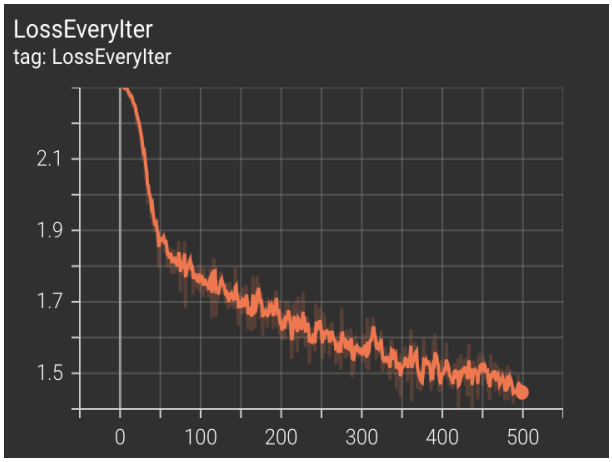

Fig. 20. Training error

```
[595  64  60  40  11  11  23  35 101  60]
[ 42 676   6  31   1   9  26  18  37 154]
[ 93  34 212 151 117 134 127  90  17  25]
[ 22  23  51 430  16 214 126  73   5  40]
[ 51  17 112 136 238 103 181 131  16  15]
[ 15  11  62 256  25 442  61 108   8  12]
[  7  18  33 179  62  34 612  33   3  19]
[ 29  19  23 125  32 131  55 545   5  36]
[286 101  18  32   1   5  16  21 423  97]
[ 61 220   7  49   2  14  47  45  50 505]
```

Fig. 21. Confusion Matrix