Homework 0: Alohomora!

Rohitkrishna Nambiar (115507944)
University of Maryland
College Park, Maryland 20740
rohit517@umd.edu

***Abstract*—In this paper, we implement the probability of boundary (pb) boundary detection algorithm in Phase 1. In Phase 2, we start with building a convolutional neural network (CNN) for image classification. CIFAR-10 dataset is used. Further, methods to improve classification accuracy are studied and implemented along with building the state of the art networks such as ResNet, ResNeXt and DenseNet.**

I. Introduction

The paper is divided as follows. We first cover sections of Phase 1 which includes the overview, filter banks, texton, brightness and color maps, along with their gradients and the final pb-lite output for a test image. In the last section, we cover Phase 2 which is a deep dive into deep learning. A convolutional neural network (CNN) is trained and analyzed for image classification.

II. Phase 1: Pb-lite Boundary Detection

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.

III. Filter Banks

The first step in the pb lite boundary detection pipeline is to filter the input image with a set of filter banks. Filtering is very important in low level image processing. In our pipeline, filtering with a set of filter banks is used to generate a texton map which depicts the texture in the image by clustering the filter responses. Three different sets of filter banks have been implemented below and are explained as follows.

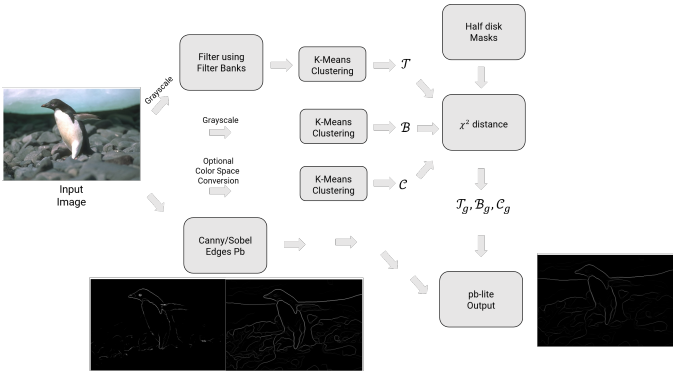


Fig. 1. Pb-lite Algorithm

A. Oriented DoG Filter Bank

The first set of filter banks is a Derivative of Gaussian (DoG) filter. To generate a DoG filter, we first create a gaussian kernel given the value of σ and the convolve it with a sobel mask. The sobel operator (S) acts as a derivative operator which is given by the following kernel

$$S = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{1}$$

$$DoG \;=\; S \otimes G \tag{2}$$

Now, we rotate the DoG filter obtained in the above step to obtain a filter bank at a single scale. This is done for another scale (σ) and we obtain a filter bank at different scale and orientations. During implementation we see that the size of the Gaussian kernel also plays an important role. If the kernel is smaller, then we have a gaussian that is not smooth. Further, smaller sized kernels can have the opposite effect(image is darkened) when sobel mask is convolved. We see that a good measure of choosing the kernel size is to have it six times the standard deviation or radius. We use a 13×13 gaussian kernel with sigma values $[1.2, 1.4]$ and number of orientation as 16.

B. Leung-Malik Filters

The Leung-Malik filters are a set of multi-scale multi-orientation filter bank with 48 filters. The first 36 filters consists of the first and second derivative of Gaussian filters at 3 scales and 6 orientations. We then have 8 Laplacian of Gaussian filters and 4 Gaussian filters. There are two versions of the LM filter namely LM Small (LMS) at scales

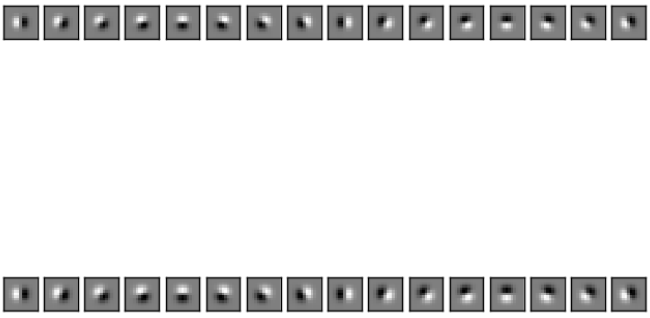


Fig. 2. Oriented DoG Filter Bank

$\sigma = \{1, \sqrt{2}, 2, 2\sqrt{2}\}$ and LM Large (LML) occuring at $\sigma = \{\sqrt{2}, 2, 2\sqrt{2}, 4\}$. The first and second derivative of Gaussian occur at first three scales with $\sigma_x = \sigma$ and $\sigma_y = 3\sigma_x$. The Gaussians occur at all the basic scales and LOG occur at σ and 3σ.

One of the observations during implementing the Leung Malik filter for LOG is that when we have a higher σ and relatively less value of kernel size, after convolution, we observe that the image is darkened. The solution to this issue is to increase the filter size as mentioned in the previous section. The LMS filter bank implemented is shown in Fig.3.

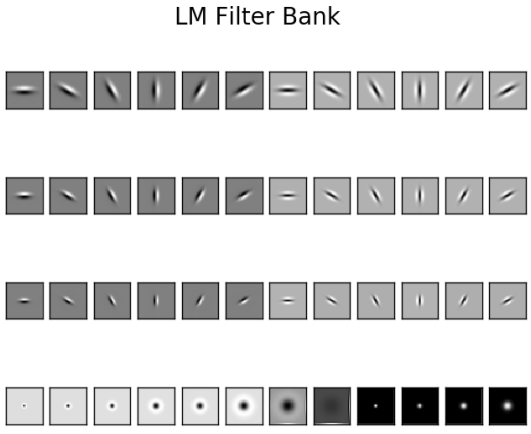


Fig. 3. Leung-Malik Filter Bank

C. 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. To implement the Gabor filter, the real component of the filter was used given by

$$g(x, y; \lambda, \theta, \psi, \sigma, \gamma) = exp(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}) \cos(\frac{2\pi x'}{\lambda} + \psi) \quad (3)$$

where λ is the wavelength of the sine wave, ψ is the phase offset, σ is the standard deviation of gaussian and γ is the spatial aspect ratio. A kernel size of 37, $\lambda = \{4, 6, 8, 10, 12\}$ and $\sigma = \{4, 4, 6, 8, 14\}$ was chosen. W observe that as the parameter γ changes the elongation of the gaussian changes. γ with value 1 was chosen to have a equal elongation along y and x. As mentioned above, the Gabor filter bank consisted of 5 scales and 8 orientations. The filter bank implemented can be seen in Fig.4.

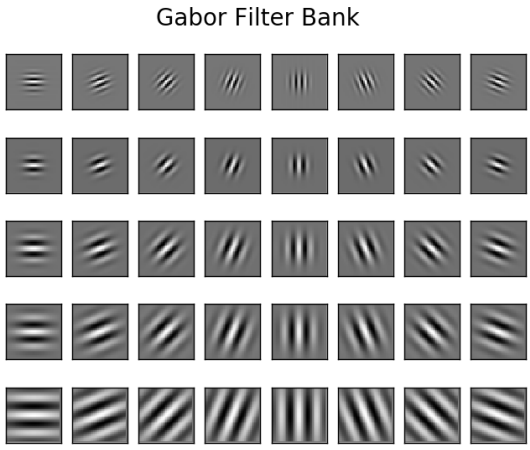


Fig. 4. Gabor Filter Bank

IV. Texton Map

We now create a texton map by using all the filters generated in the previous section. The texton map is a quantized image where the pixel values range from $(1 : K)$ where K is the number of clusters. Before we proceed to clustering, each filter is convolved with a gray scale input image. Thus N filters produce a 3D vector where the depth is equal to the number of filters. We use $K = 64$ in our case. A sample texton map for an input image is given in Fig.6.

Fig. 5. Input Image

V. Brightness Map

Similar to texton map, we compute the brightness map (B) by first converting the input image to gray-scale and then clustering the pixels. Here we use 16 clusters. The output is shown in Fig.7

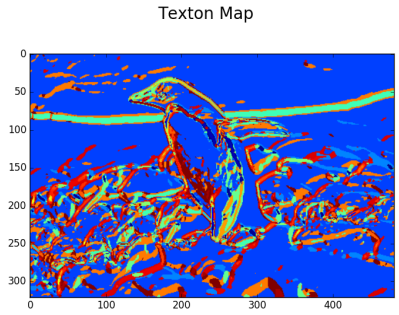


Fig. 6. Texton Map

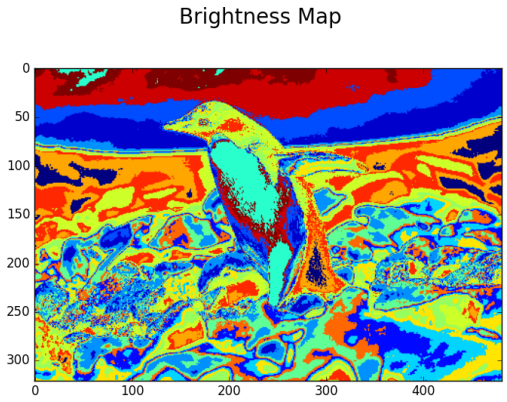


Fig. 7. Brightness Map

VI. Color Map

Similar to texton and brightness map, we compute the color map (C) of the input image by clustering the RGB pixels. Here we use 16 clusters. The output is shown in Fig.8

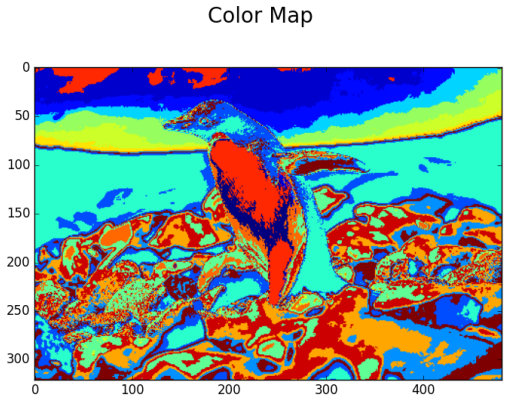


Fig. 8. Color Map

VII. Texture, Brightness and Color Gradients

To obtain T_g, B_g, 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.

A. Half-Disk Masks

Half-disc masks are pairs of binary images of half-discs. They help us to compute the χ^2 (chi-square) distances using a filtering operation, which is much faster than looping over each pixel neighborhood and aggregating counts for histograms. The half masks were generated at different scales and orientation. We have used 3 scales at $[5, 10, 15]$ where each value corresponds to the radius of the disk and 8 orientations. The size of the kernel varies as the radius of the disk varies. The half-disk implementation output can be seen below in Fig.9.

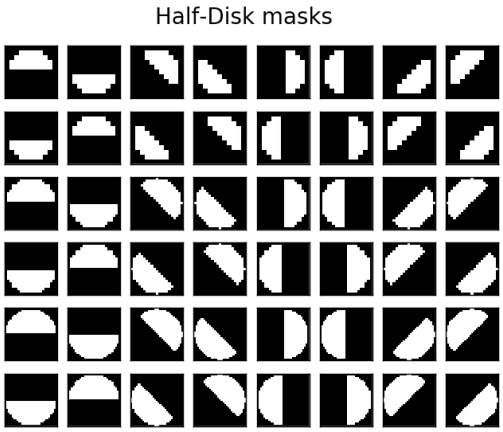


Fig. 9. Half-disk mask

B. Computing gradients

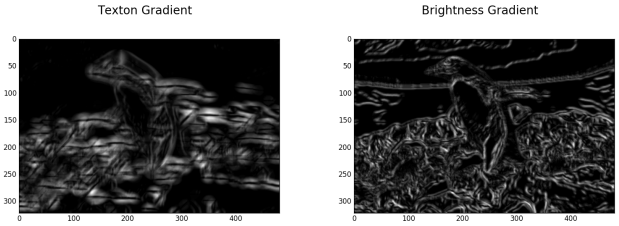
T_g, B_g, C_g encode how much the texture, brightness and color changes at every pixel. They are computed by comparing the distributions in left/right half-disc pairs. The gradient is small if distribution is similar and it is large otherwise. The distributions are compared using the χ^2 measure. The number of bins of the histogram is equal to the clusters formed in each map. The output matrix is of the dimension $M \times N \times K$ where K is the number of filters. We then take the mean of the matrix along the depth(N channel) where list of elements at each pixel position can be though of as the features. The output after computing gradients are given in Fig.10.

VIII. Pb-lite Output

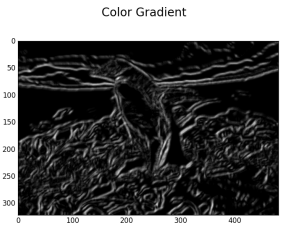
The Pb-lite output is obtained by using the sobel and canny baselines and combining with the gradients using the following function

$$PbEdges = \frac{(T_g + B_g + C_g)}{3} \odot (w_1 * cannyPb + w_2 * sobelPb) \tag{4}$$

The output of the Pb-lite is given below in Fig.11 along with the sobel, canny and ground truth images.



(a) (b)

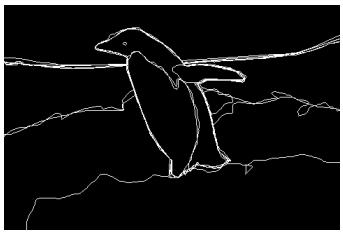


(c)

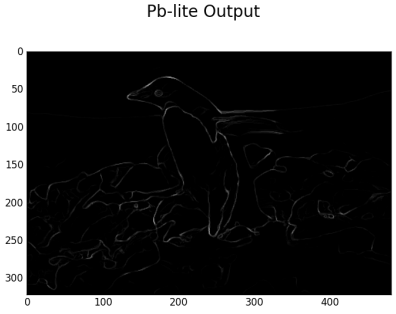
Fig. 10. (a) Texton Gradient (b) Brightness Gradient and (c) Color Gradient.

(a) Canny

(b) Sobel



(c) Ground Truth



(d) Pb-lite

Fig. 11. (a) Canny output (b) Sobel Output (c) Ground truth and (d) Pb-lite output.

IX. Phase 2: Deep Learning

A. Section 3.3

In this section, we implement a convolutional neural network for the task of image classification on the CIFAR10 dataset. The architecture of the network is as follows shown in Fig.12.

We see that this is a very shallow architecture implemented. This network consists of two $Conv -> Pool -> Norm$ layers followed by a fully connected layer and the final output layer. The total number of parameters trainable in the network is 4,225,162. Adam optimizer was chosen with a constant learning rate of 0.001. A batch size of 64 was chosen and the network trained for 50 epochs. An accuracy of 53.44 percent was obtained on the test set. The confusion matrix of the trained model on test data is given in Fig.13.

```
[685  30  40  32  45  21   5  38  22  82] (0)
[ 18 746   1  11   0   5   4  19  11 185] (1)
[147  19 234 160  21 164  26 161  11  57] (2)
[ 59  23  20 386  20 244  14 127   5 102] (3)
[ 60  20  47 179 166 135  24 330   8  31] (4)
[ 24  17  12 164   9 567  13 124   6  64] (5)
[ 25  18  16 161  13 100 461  93  13 100] (6)
[ 18  10   7  37  19  60   2 791   2  54] (7)
[160 102  15  23  20  22   0  33 490 135] (8)
[ 33  84   3  17   1   5   0  28  11 818] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
('Accuracy: 53.44', '%')
```

Fig. 13. Test confusion matrix

On observing the training accuracy and loss curve as shown in Fig.14 and Fig.15 respectively, we see that although the network does a good job on the training set, it fails to achieve a good accuracy on the test set. This maybe because the network is not able to generalize well and hence performs poorly on the test set.

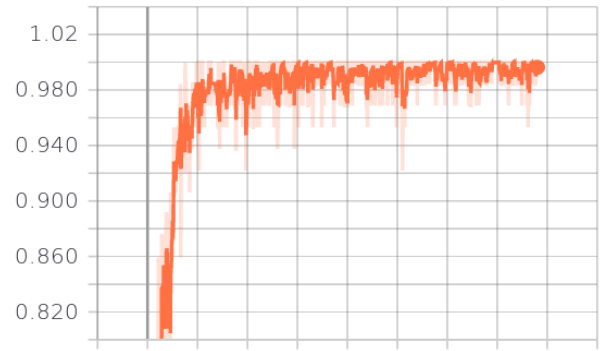


Fig. 14. Training accuracy

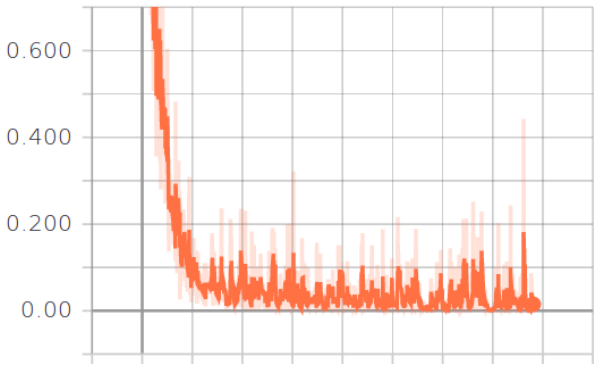


Fig. 15. Training error

X. Conclusion

Thus we have successfully implemented the pb-lite boundary detection algorithm and trained a shallow convolutional neural network for image classification on the CIFAR-10 dataset.

References

[1] CMSC733 HW-0, https://cmsc733.github.io/2019/hw/hw0/, 01/29/2018.
[2] Tensorflow, https://www.tensorflow.org/, 01/29/2018.

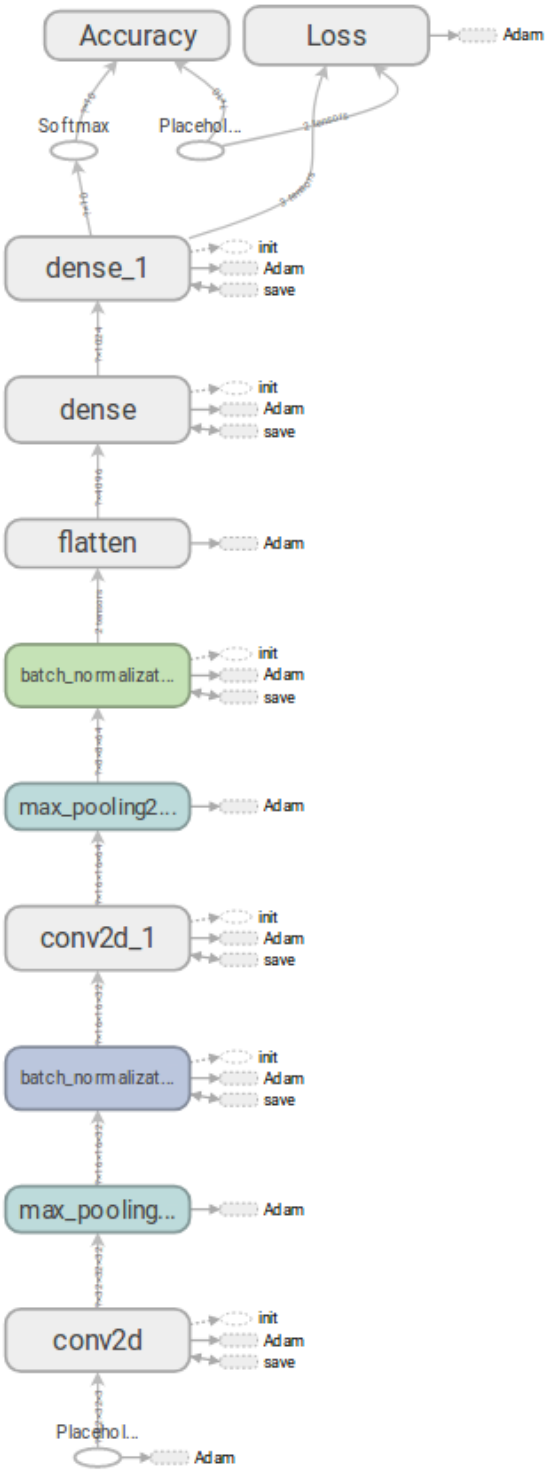


Fig. 12. Shallow architecture for image classification