# My Alohomora

Aadhya Puttur

## I. Introduction

To detect edges in an image, you have to observe the intensity. Where the intensity function in an image changes rapidly are where edges are located. If you gray-scale an image with an object on it, throughout the image the unpredictable or changes in intensity are on edges. The Canny edge detector involves 1) calculating the gradient of the image, 2) gradient magnitude values of pixel not at the local maxima is going to be zero, 3) edge-linking is then performed. Canny edge detector resulting image displays edges with pixel value of 1 and everywhere else to be zero.The Sobel operator uses two 3x3 kernels that are convolved with the original image to calculate the horizontal and vertical changes in an image

## II. Phase 1: Shake My Boundary

In phase 1, I will be discussing my analysis on the implementation of the pb-lite boundary detection algorithm. This algorithm is compared to the sobel and canny algorithm. Sobel and canny both observe changes in intensity in images, although in pb-lite it considers texture information. To do this I created 1) a filter bank 2) a texton, brightness color map 3) texture, brightness, color gradient map computation 4) boundary detection.

### A. FilterBanks

First, we want to create a filter bank to capture texture information from images. The three filters that are used are oriented division of Gaussian (DoG) filter, Leung-Malik filter, and Gabor filter. The result are shown in Fig. 1 to Fig. 3.

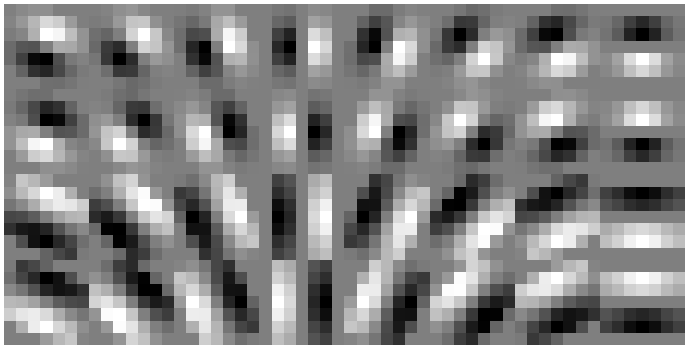


Fig. 1. Oriented Derivative of Gaussian Filters, generated with $\sigma_{=1,2}$

A. Puttur is with the Department of Computer Science, Worcester Polytechnic Institute, Worcester, MA 01609, USA (e-mail: aputtur@wpi.edu)

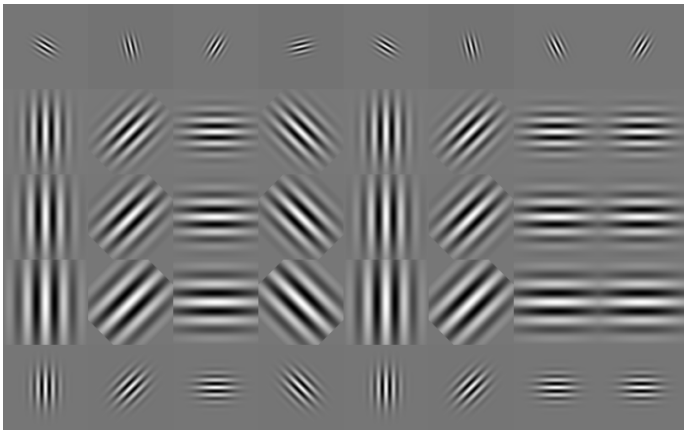


Fig. 2. Oriented Gabor Filters

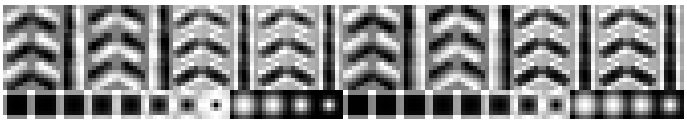


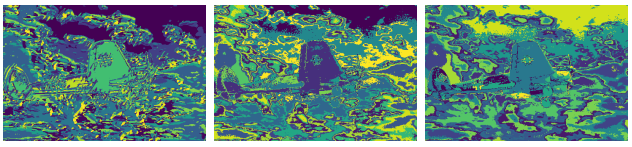
Fig. 3. Leung-Malik large and small filter banks. $\sigma = 1, \sqrt{2}, 2, 2\sqrt{2}$ and $\sigma = \sqrt{2}, 2, 2\sqrt{2}, 4$ and with elongation factor 3

The Gaussian filter is used as it has the optimal trade off between being localized in space and frequency. Laplacian of Gaussian graphically is known as the "inverted Mexican hat" and is used to reduce the effects of noise through smoothing the image with Gaussian. The shape of the function tells us that Laplacian of Gaussian is rationally symmetric as it has a ring of negative values in the middle and positive values outside of this ring Fig. 3. The Gabor Filters are designed based on the filters of the human visual system. The Gabor filters is a Gaussian kernel function modulated by a sinusoidal plane wave, which is why you see the changes in the image from black and white Fig. 2.

### B. Generating a texton, brightness, and colormap

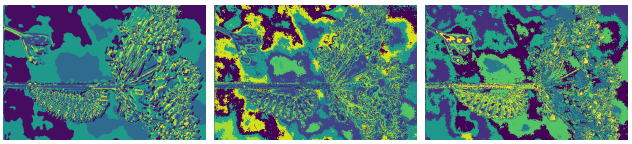
Once the necessary filter banks have been created, we will every single one of the filter banks on the images. A vector of filter responses will be created at each pixel of the image where if we have N filters in the filter bank. We will do this by clustering the filter responses at all pixels in the image in to K textons using sklearn's KMeans. We set K = 64 for the texton map, and 16 for the brightness, and color map. The results of the texton, brightness, and color map shown from Fig. 4 - 13.

The purpose of a brightness map is to then capture the



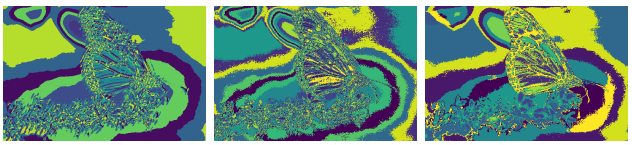
(a) (b) (c)

Fig. 4. $\mathcal{T}, \mathcal{B}, \mathcal{C}$



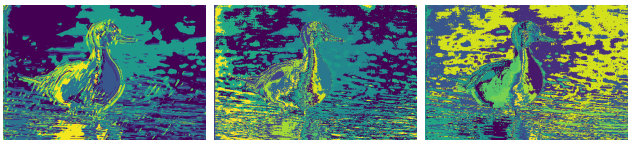
(a) (b) (c)

Fig. 5. $\mathcal{T}, \mathcal{B}, \mathcal{C}$



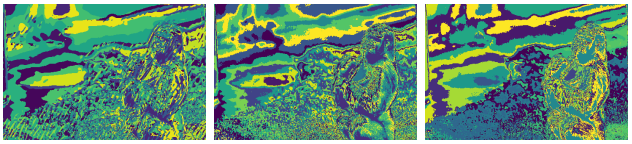
(a) (b) (c)

Fig. 6. $\mathcal{T}, \mathcal{B}, \mathcal{C}$



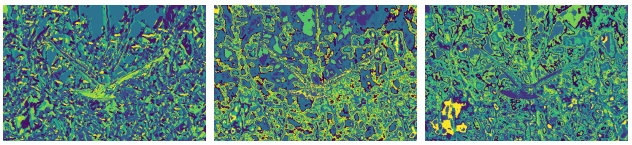
(a) (b) (c)

Fig. 7. $\mathcal{T}, \mathcal{B}, \mathcal{C}$



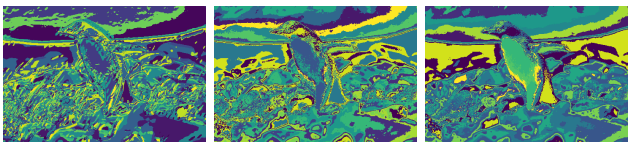
(a) (b) (c)

Fig. 8. $\mathcal{T}, \mathcal{B}, \mathcal{C}$



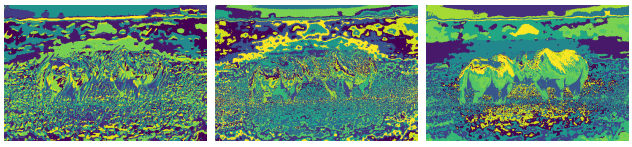
(a) (b) (c)

Fig. 9. $\mathcal{T}, \mathcal{B}, \mathcal{C}$



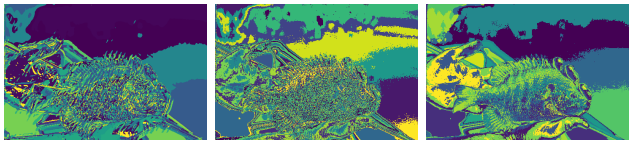
(a) (b) (c)

Fig. 10. $\mathcal{T}, \mathcal{B}, \mathcal{C}$



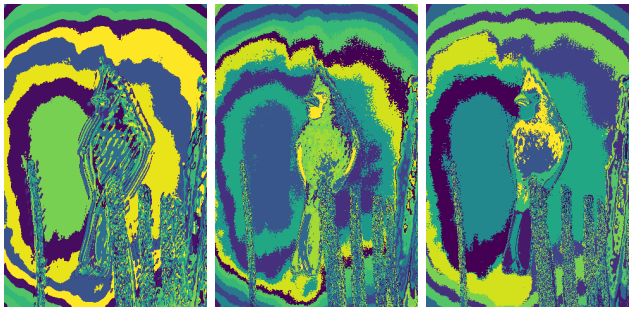
(a) (b) (c)

Fig. 11. $\mathcal{T}, \mathcal{B}, \mathcal{C}$



(a) (b) (c)

Fig. 12. $\mathcal{T}, \mathcal{B}, \mathcal{C}$



(a) (b) (c)

Fig. 13. $\mathcal{T}, \mathcal{B}, \mathcal{C}$

(a)

(b)

(c)

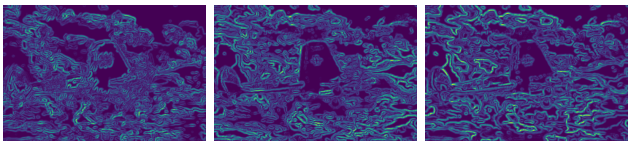
(d)

Fig. 14. Half Disk Mask Pairs

brightness changes in the image. We cluster the brightness values using kmeans and use 16 number of clusters. For the color map, unlike the other maps, we use the rgb values to capture the color changes in the image.

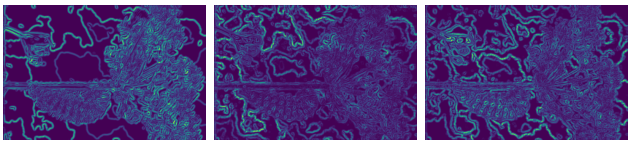
### *C. Finding Texture, Brightness, and Color Gradients*

Texture, Brightness and Color gradients are represented as $\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g$ respectively. They tell us how much texture, brightness, and color distributions are changing at each pixel. To get the gradients we use Half-disc masks to compute differences of values across different shapes and sizes Fig. 14. Half-disc masks are pairs of binary images of half-discs.



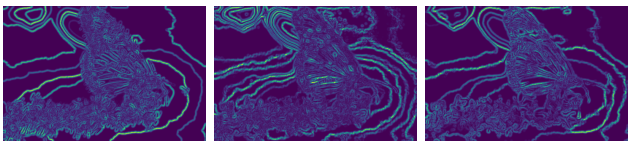
(a) (b) (c)

Fig. 15. $\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g$



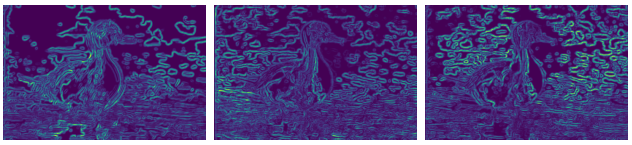
(a) (b) (c)

Fig. 16. $\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g$



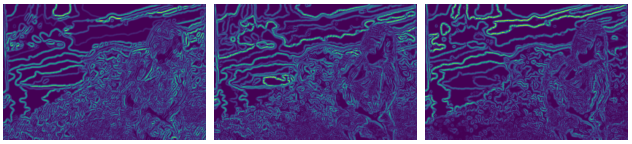
(a) (b) (c)

Fig. 17. $\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g$



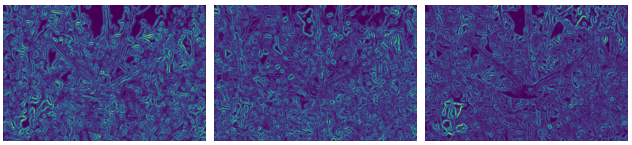
(a) (b) (c)

Fig. 18. $\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g$



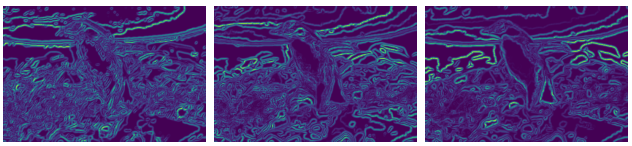
(a) (b) (c)

Fig. 19. $\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g$



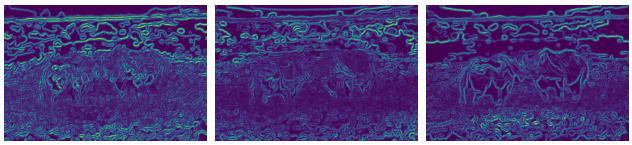
(a) (b) (c)

Fig. 20. $\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g$



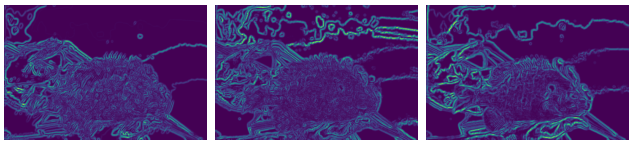
(a) (b) (c)

Fig. 21. $\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g$



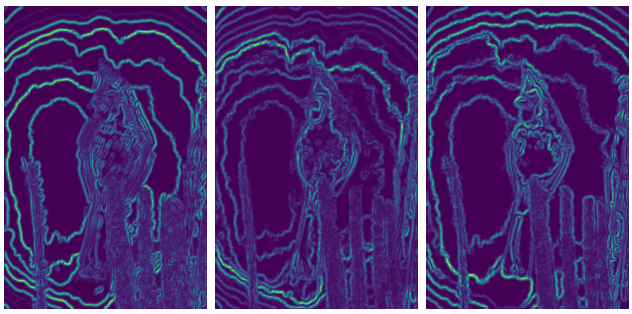
(a) (b) (c)

Fig. 22. $\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g$



(a) (b) (c)

Fig. 23. $\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g$



(a) (b) (c)

Fig. 24. $\mathcal{T}_g, \mathcal{B}_g, \mathcal{C}_g$

(a) (b) (c)

Fig. 25. sobel, canny, and pb-lite

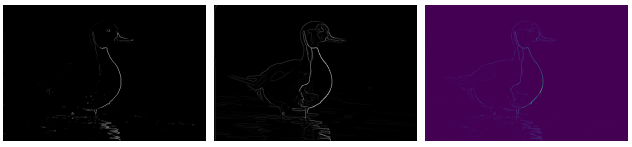
(a) (b) (c)

Fig. 26. sobel, canny, and pb-lite

(a) (b) (c)

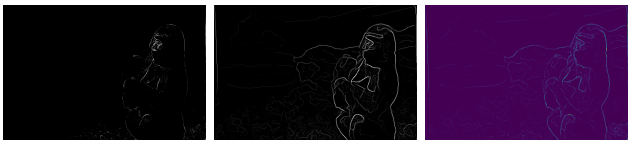
Fig. 27. sobel, canny, and pb-lite

We use this to calculate $\mathcal{X}^2$ or the chi-square distance. The pairs are used in order to make comparisons to see if the gradients of the opposing sides are dissimilar or similar.



(a) (b) (c)

Fig. 28. sobel, canny, and pb-lite



(a) (b) (c)

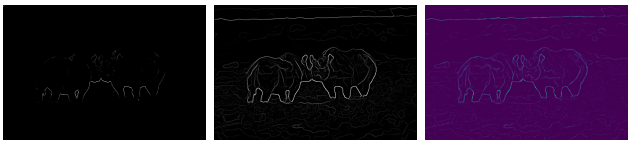
Fig. 29. sobel, canny, and pb-lite

(a) (b) (c)

Fig. 30. sobel, canny, and pb-lite

(a) (b) (c)

Fig. 31. sobel, canny, and pb-lite



(a) (b) (c)

Fig. 32. sobel, canny, and pb-lite

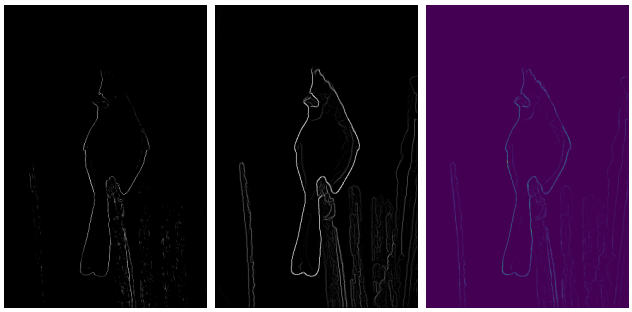
(a) (b) (c)

Fig. 33. sobel, canny, and pb-lite

From this, we then generate a 3D matrix of size m x n x N where N is number of filters.

### D. Pb-lite output

From these gradient changes, we then want to apply them on the sobel and canny images from the dataset. We combine these features to be able to see the strength of the boundaries on sobel and canny. From the output from Fig. 25 - Fig. 34 we can see that there is a lot of suppression on the filters I



(a) (b) (c)

Fig. 34. sobel, canny, and pb-lite

generated for pb-lite. Sometimes this is a good thing since sobel and canny although they are are good edge detectors, they sometimes overshoot their performance on detecting edges.

## III. Phase2

### A. My First Network

For my first neural network, I had a batch size of 32, with 50 epochs, and I used SGD with learning rate of 0.0001. After these 50 epochs my training accuracy was barely 0.50.

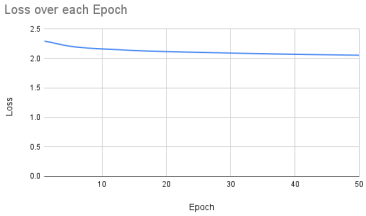



Fig. 35. Cross Entropy Loss during Training, batch = 32

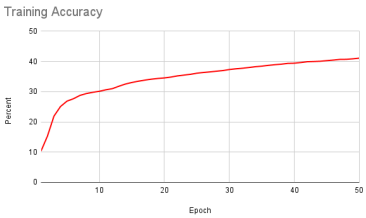



Fig. 36. Training Accuracy, batch = 32

The test accuracy was 40.17 which is very bad. As you can see in Fig. 36 the training accuracy was exponentially increasing at one point and then after about 10 epochs you can see the growth starting to decrease. It was converging near 50 percent.