RBE549: Homework 0 - Alohomora

Radha Saraf
WPI Robotics Engineering
rrsaraf@wpi.edu
Using 3 late days

***Abstract*—We develop a simplified version of pb(probability of boundary) boundary detection algorithm, which finds boundaries by examining brightness, color, and texture information across multiple scales. The output is a per-pixel probability of boundary. We also train a simple convolutional neural network on PyTorch for the task of classification. The input is a single CIFAR-10 image and the output is the probabilities of the 10 classes of objects that CIFAR-10 is composed of.**

***Index Terms*—Probability-based edge detection, Convolutional Neural Networks.**

I. Phase 1: Shake My Boundary

This section sees the implementation of a lighter version of the pb boundary detection algorithm introduced in [1]. It is different from classical computer vision techniques like Sobel and Canny in that it employes the texture and color information present in the image in addition to the intensity discontinuities. This is done in 4 steps in the sections to follow:

1. Filter Banks
2. Texture, Brightness and Color Maps T, B, C
3. Texture, Brightness and Color Gradients Tg, Bg, Cg
4. Pb-lite output combined with baselines

We will go through brief descriptions of each of these steps.

A. *Filter Banks*

The first step of the pb lite boundary detection pipeline is to filter the image with a set of filter banks. We will create three different sets of filter banks for this purpose. Filtering the images with these banks helps build the low level features that represent texture. We will use filtering both to measure these texture properties and to aggregate regional brightness and texture distributions.

*1) **Oriented DoG Filter Bank**:* These are a set of oriented DoG or Derivative of gaussian filters created by convolving a simple Sobel filter and a Gaussian kernel and then rotating the result. For the purpose of this paper, the filter bank sees filters of size 7x7 at 16 different orientations and 2 scales (1, $\sqrt{2}$). Fig. 1 shows these filters.

Fig. 1: DoG filters

*2) **Leung-Malik Filter Bank**:* The Leung-Malik filters or LM filters are again, a set of multi scale, multi orientation filter bank with 48 filters. It consists of first and second order derivatives of Gaussians at 6 orientations and 3 scales making a total of 36; 8 Laplacian of Gaussian (LOG) filters; and 4 Gaussians. We consider two versions of the LM filter bank. In LM Small (LMS), the filters occur at basic sigma scales (1, $\sqrt{2}$, 2, $2\sqrt{2}$). The first and second derivative filters occur at the first three scales with an elongation factor of 3. The Gaussians occur at the four basic scales while the 8 LOG filters occur at sigma and 3sigma. For LM Large (LML), the filters occur at the basic sigma scales ($\sqrt{2}$, 2, $2\sqrt{2}$, 4). Fig. 2 shows these filters.

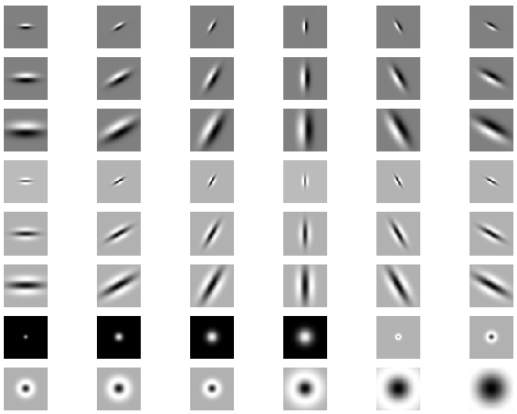

Fig. 2: LM filters

*3) **Gabor Filter Bank**:* Gabor filter [**?**] is a linear filter which analyzes whether there is any specific frequency content in the image in certain directions in a localized region around the point or region of analysis. For this work, the scales considered were [18, 24, 30, 36, 42] at 8 different orientations. Fig. 3 shows these filters.

B. *Texton, Brightness, Color maps- T, B, C*

*1) **Texton map**:* Filtering an input image with each element of the filter bank results in a vector of filter responses centered on each pixel. A distribution of these N-dimensional filter responses could be thought of as encoding texture properties. This representation is simplified by replacing each N-dimensional vector with a discrete texton ID. We do this by clustering the filter responses at all pixels in the image into K textons using K-means clustering. Each pixel is then

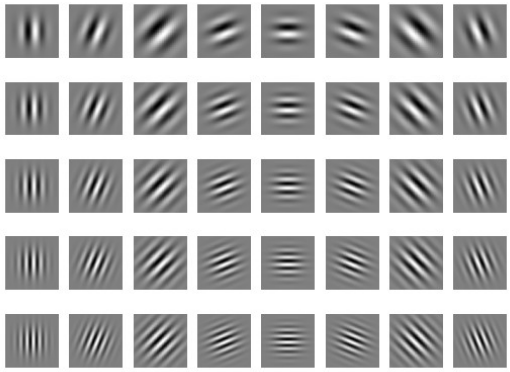

Fig. 3: Gabor filters

represented by a one dimensional, discrete cluster ID instead of a vector of high-dimensional, real-valued filter responses. This is then represented in the form of a single channel image with values in the range of [1,2,3,...,K]. Here, the value for K has been chosen as 64. It was observed that the filtered output of the images had very low intensity values which lead to poor clustering. So, after filtering, the output was normalized in the 0-255 range which significantly improved the clustering.

*2) **Brightness Map**:* The concept of the brightness map is as simple as capturing the brightness changes in the image. Here, again we cluster the brightness values(grayscale equivalent of the color image) using kmeans clustering into a chosen number of clusters (K=16). We call the clustered output as the brightness map B.

*3) **Color Map**:* The concept of the color map is to capture the color changes or chrominance content in the image. Here, again we cluster the RGB color values using kmeans clustering into a chosen number of clusters (K=16). We call the clustered output as the color map C.

Figures 4 through 13 show these texton, brightness and color maps for the ten different test images.

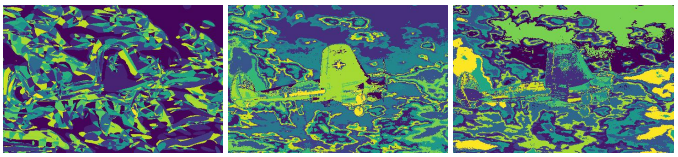

Fig. 4: T, B, C maps for image 1

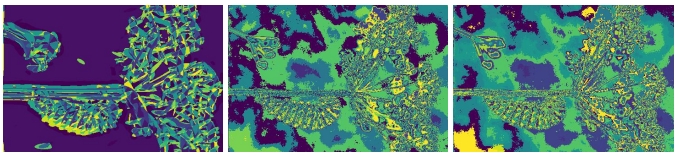

Fig. 5: T, B, C maps for image 2

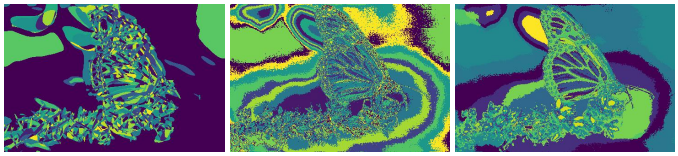

Fig. 6: T, B, C maps for image 3

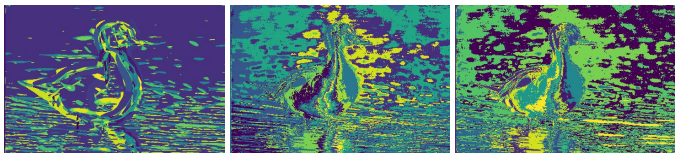

Fig. 7: T, B, C maps for image 4

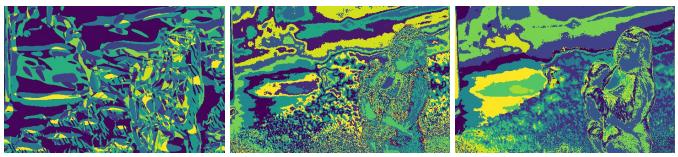

Fig. 8: T, B, C maps for image 5

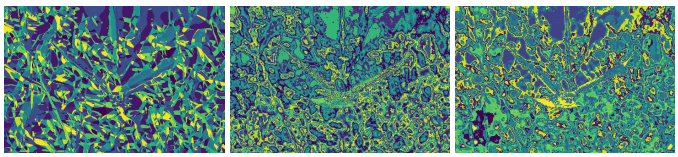

Fig. 9: T, B, C maps for image 6

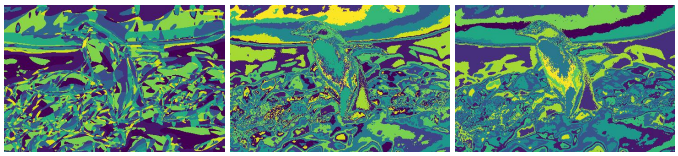

Fig. 10: T, B, C maps for image 7

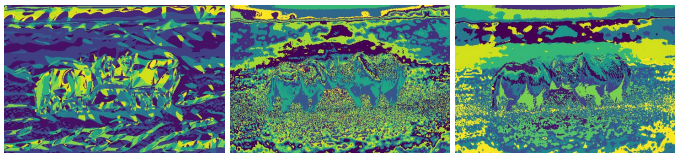

Fig. 11: T, B, C maps for image 8

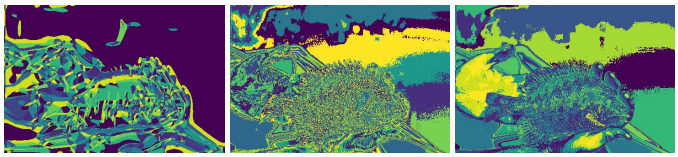

Fig. 12: T, B, C maps for image 9

C. Texture, Brightness and Color Gradients- Tg, Bg, Cg

Texture, Brightness and Color gradients encode how much the texture, brightness and color distributions are changing at a pixel. To obtain Tg, Bg, and Cg, we need to compute

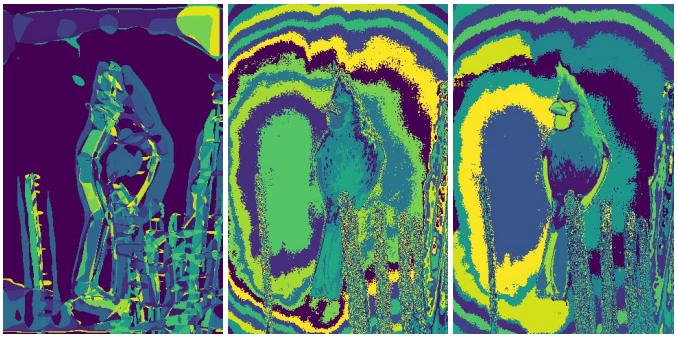

Fig. 13: T, B, C maps for image 10

differences of values across different shapes and sizes. This can be achieved very efficiently by the use of Half-disc masks.

1) ***Half Disk Masks****:* The half-disc masks are simply (pairs of) binary images of half-discs. These allow us to compute the chi-square distances using a filtering operation, which is much faster than looping over each pixel neighborhood and aggregating counts for histograms. Forming these masks is quite trivial. The set of masks used for this work with 8 orientations and 3 scales is shown in Fig. 4

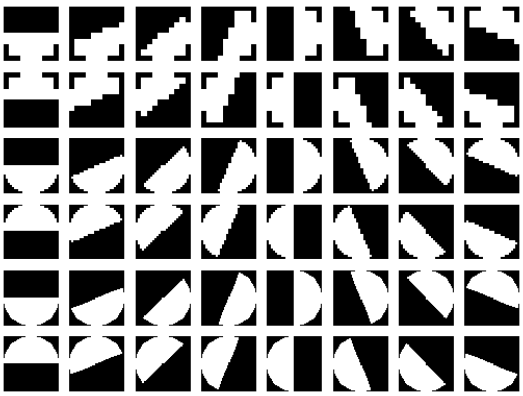

Fig. 14: Half-disc masks

We compute Tg, Bg, Cg by comparing the distributions in left/right half-disc pairs (opposing directions of filters at same scale) 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. We compare texton, brightness and color distributions with the chi-square measure. The chi-square distance is a frequently used metric for comparing two histograms. [**?**] came in handy for calculations of these gradient maps.

Figures 15 through 24 show the Tg, Bg, Cg gradients for the ten test images.

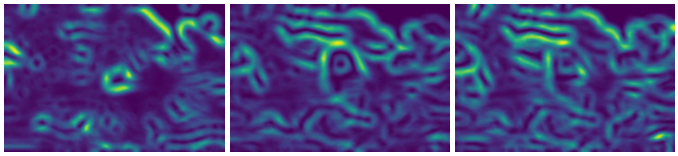

Fig. 15: Tg, Bg, Cg gradients for image 1

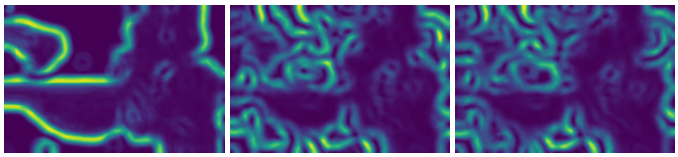

Fig. 16: Tg, Bg, Cg gradients for image 2

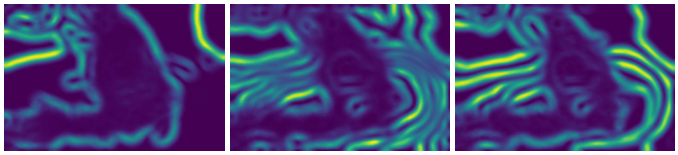

Fig. 17: Tg, Bg, Cg gradients for image 3

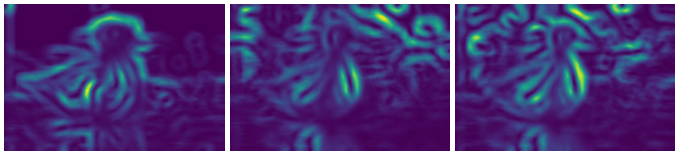

Fig. 18: Tg, Bg, Cg gradients for image 4

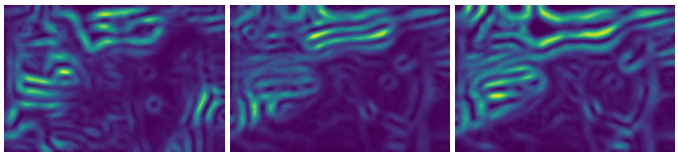

Fig. 19: Tg, Bg, Cg gradients for image 5

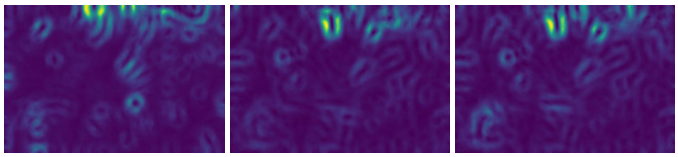

Fig. 20: Tg, Bg, Cg gradients for image 6

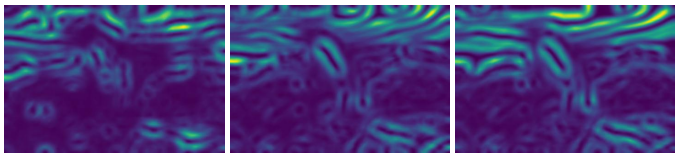

Fig. 21: Tg, Bg, Cg gradients for image 7

D. Pb-lite output combined with baselines

Figures 25 through 33 show the final pb-lite output combined with the canny and sobel baselines (both baselines given equal weightage), alongside the original canny and

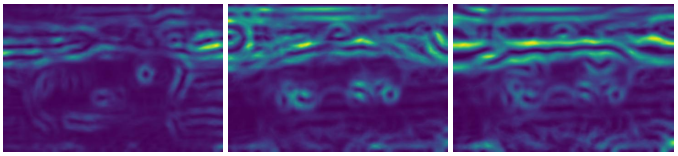

Fig. 22: Tg, Bg, Cg gradients for image 8

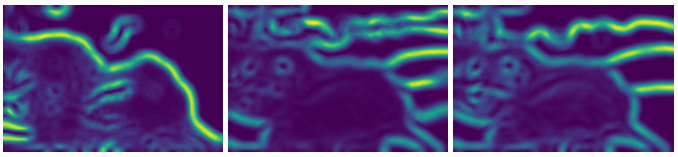

Fig. 23: Tg, Bg, Cg gradients for image 9

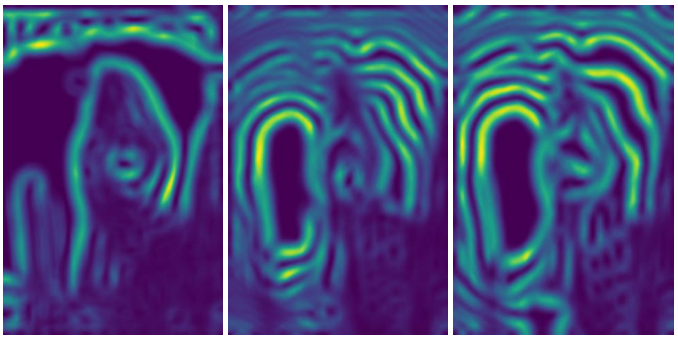

Fig. 24: Tg, Bg, Cg gradients for image 10

sobel results for comparison. It can be seen that the pb-lite edges are devoid of most of the noise that canny and sobel contain. This is because it is good at suppressing the false positives that show up as noise in sobel and canny. The final output is certainly not the best in this work since we see that alongside the noise the true edges are also getting suppressed. Probably, a different combination of filters might result in a better outcome. We tried extending the scales of LM filters for instance but it resulted in little improvement. There is scope for some trial and error here.

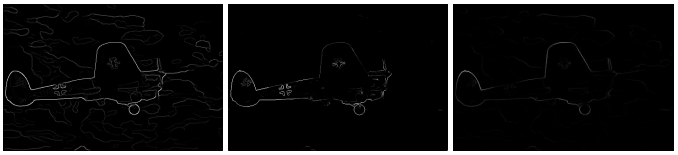

Fig. 25: Canny, Sobel, Pb-lite for image 1

Fig. 26: Canny, Sobel, Pb-lite for image 2

Fig. 27: Canny, Sobel, Pb-lite for image 3

Fig. 28: Canny, Sobel, Pb-lite for image 4

Fig. 29: Canny, Sobel, Pb-lite for image 5

Fig. 30: Canny, Sobel, Pb-lite for image 6

Fig. 31: Canny, Sobel, Pb-lite for image 7

Fig. 32: Canny, Sobel, Pb-lite for image 8

Fig. 33: Canny, Sobel, Pb-lite for image 9

REFERENCES

[1] https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/grouping-/papers/amfm_pami2010.pdf

[2] https://medium.com/@anuj_shah/through-the-eyes-of-gabor-filter-17d1fdb3ac97.

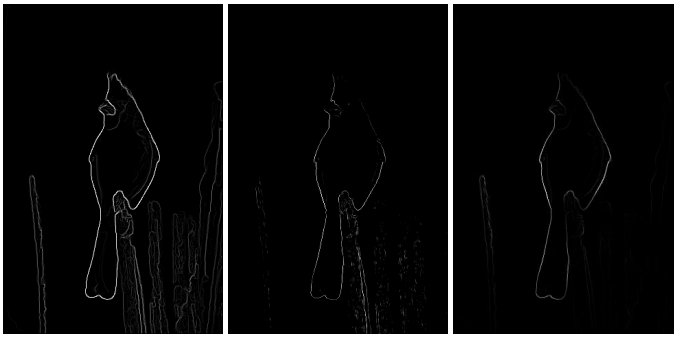

Fig. 34: Canny, Sobel, Pb-lite for image 10

[3] http://cs.brown.edu/courses/cs143/2011/results/proj2/sswarr/