Project 2: pb-lite: Boundary Detection

Tala Huhe (thuhe)

Overview

Edge detection is a classic problem in computer vision and a fundamental tool for many algorithms. The goal of such algorithms is to find all relevant discontinuities in an image. Because what constitutes a relevant discontinuity can vary highly among even humans, edge detection is considered to be a difficult problem.

The most naive method of solving this problem (Sobel) involves generating an intensity (i.e. grayscale) map from the original image, and computing differences in intensity across neighboring pixels. By cutting these values off at a predefined constant, this algorithm finds all edges sharper than that value. A slightly smarter version (Canny) attempts to decide which pixels comprise edges by checking whether the gradient at that pixel is similar to gradients at neighboring pixels. However, these two algorithms only consider changes in intensity, and fail to consider changes in texture.

In this project, we will attempt to beat these naive algorithms using *pb-lite*. In addition to considering intensity, we will also consider changes in texture and brightness in the local neighborhood. Using all of this information, we will assign to each pixel a probability that it lies on an edge.

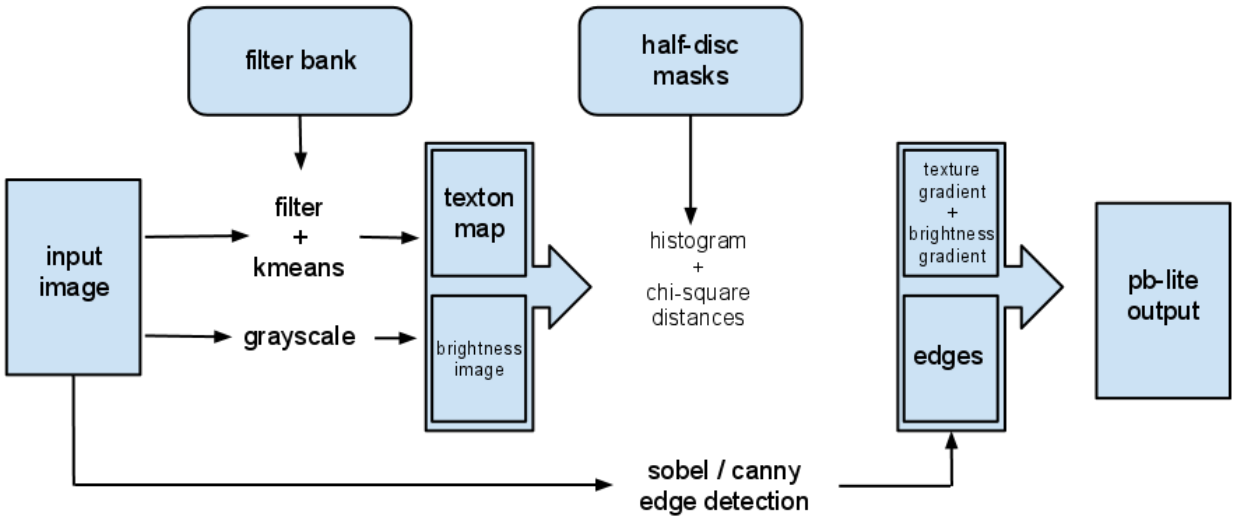
In clockwise order, from top-left: original image, Sobel edge detection, pb-lite, Canny edge detection.

Algorithm

This is a rough outline of our algorithm:

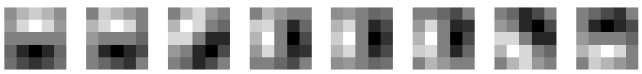
1. We begin by generating a bank of filters that we will use to detect textures. We also generate half-disc masks which we will later use to compute gradients in texture and brightness.
2. In order to simplify texture detection, we generate a grayscale version of our image.
3. We apply our filter bank to our input image in order to generate a series of responses. These correspond to parts of the image that look like our filters.
4. We use K-means to segment our image into multiple *textons*, where each texton is characterized by the pixels' responses to all the filters in our filter bank. This forms the texton map for our image.
5. For every pixel, we compute the change in both texture and intensity going in every direction. We average these together to compute the mean change in texture and intensity at every pixel.
6. We use a naive edge detector such as Sobel or Canny to compute a rough edge map for the image.
7. We combine the results of our naive edge detector by scaling the values by mean change in texture and intensity at that pixel.

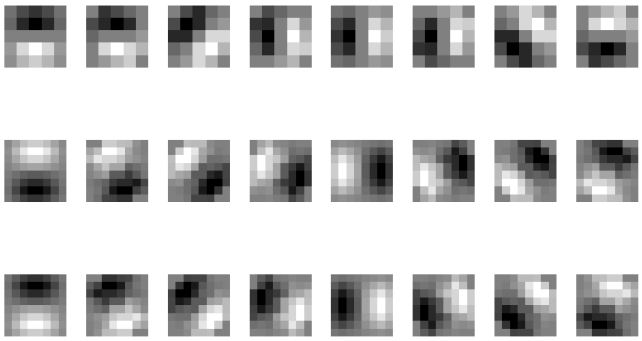
The pipeline is roughly the following:



Filter Bank

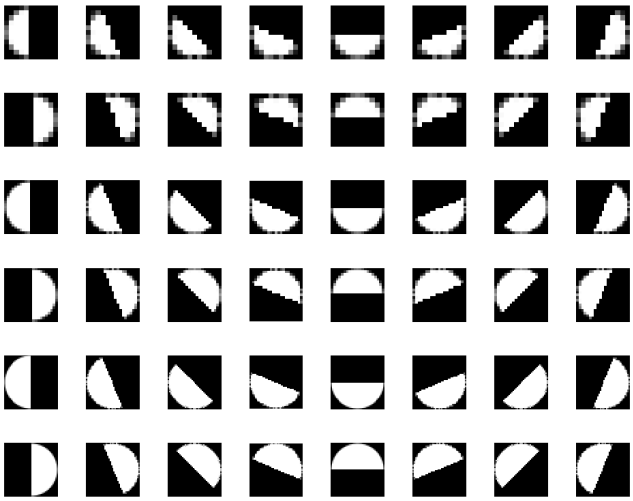
We will use a filter bank to detect texture properties in our input image. For this project, we use the derivative of Gaussian filters. We generate filters at two scales in order to detect scale change, and 16 orientations in order to detect textures in different orientations. Below is a visualization of our filter bank:





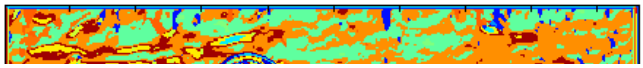
Half-Disc Mask

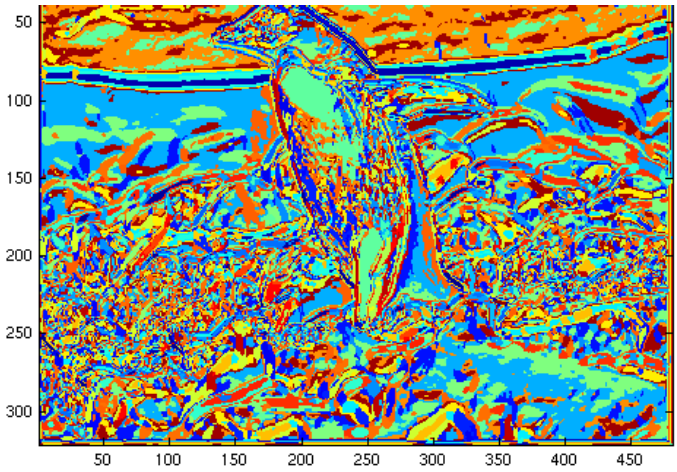
The half-disc masks are used to speed up computation of the texture and brightness gradients. We generate these at three scales in order to consider different-sized neighborhoods, and 8 orientations to consider neighborhoods in different orientations. For each scale and orientation, we compute a pair of masks. Each will be used for a different half of the neighborhood surrounding the pixel. Below is a visualization of the masks we use:



Texton Map

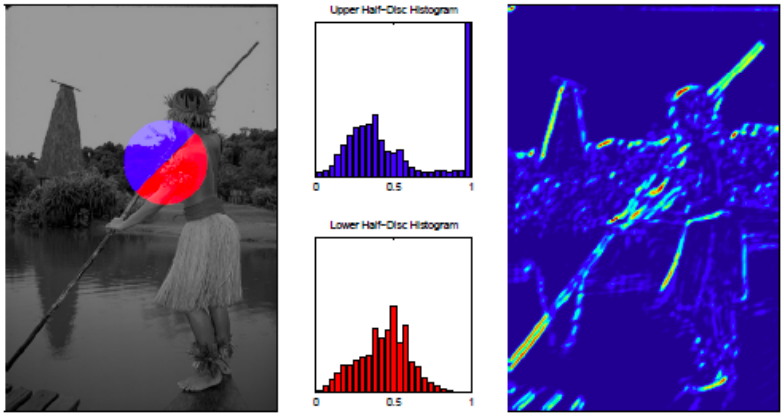
In order to generate the texton map, we categorize each pixel based on its responses to each filter in our filter bank. To segment the image into multiple textons, we use *K-means* with K = 64. Below is a sample texton map for one of our output images:





Computing Mean Texture and Brighness Differences

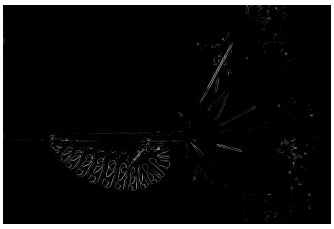
We begin by forming a histogram for each side of the neighborhood around a pixel. Once we compute these histograms, we use the chi-squared distance between these histograms to compute change in texture/brightness distributions at that pixel. In order to capture these changes for different scales and orientations, we repeat this process for all of our half-disc masks. We then average all of these values together to get the mean brightness change at a particular pixel. In order to compute the texture change, we simply repeat this process, replacing our intensity image with our texton map.

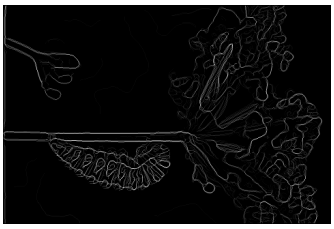


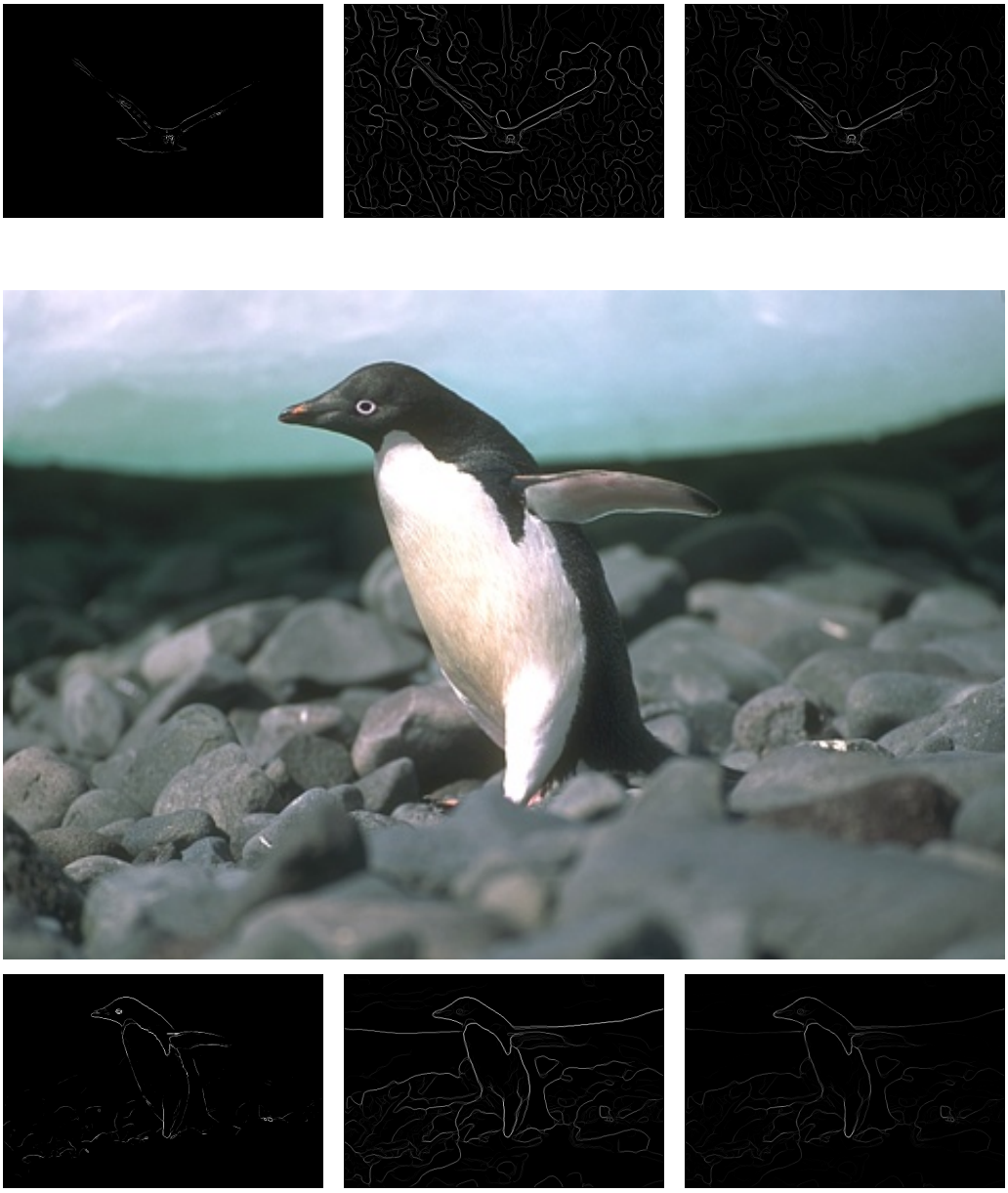
Results

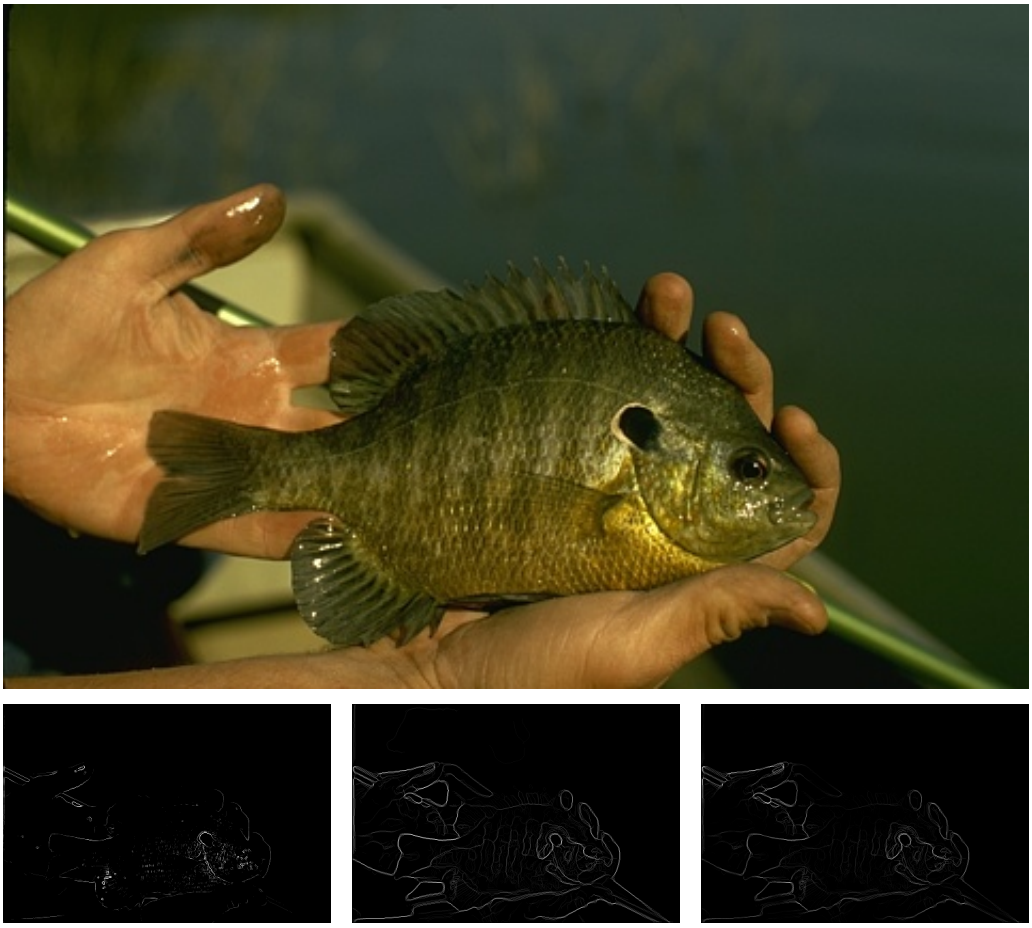
Qualitatively speaking, our algorithm produced better results than the baseline algorithms. However, pb-lite failed to pick up on edges that the naive edge detectors missed. This is because we are using pb-lite to filter out bad edges, not introduce more candidates.

The three results, in left-to-right order, are Sobel, Canny, and pb-lite. Hover over the results to see a full-size image.



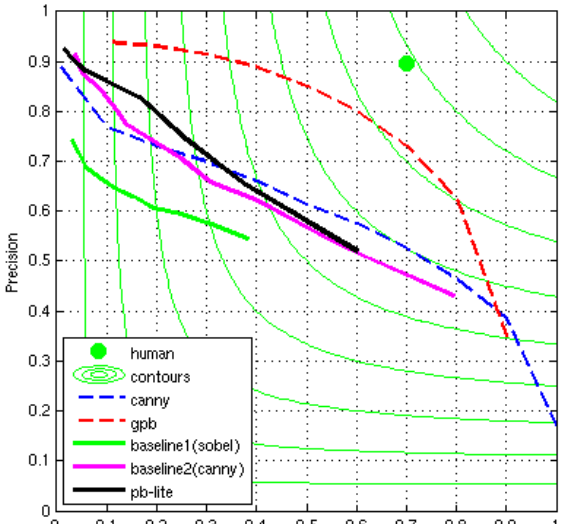


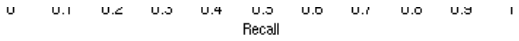




Failure case. The fish's fins were not picked up by a naive edge detector, so pb-lite didn't find it either.

We benchmark our results against the naive Sobel and Canny edge detectors. In addition, we compare our results against **gpb**, a more comprehensive version of our algorithm. Finally, we compare our results with that of **human annotators**.





Unfortunately, my implementation of pb-lite only received an F-score of 0.56, whereas the Canny edge detector received 0.57. For comparison, the Sobel baseline received a score of 0.45.