Programming Assignment 2: Segmentation

CS 4670 Spring 2019

February 27, 2019

Key Information

Assigned:	Tuesday, February 26rd (Code is on CMS)
Due:	Friday, March 8th
Files to Submit:	`texture_gradient.py`

This project can be done **individually or in pairs**.

Overview

The goal of this assignment is to write several functions that detect edges within images and segment images into perceptually distinct clusters of pixels. In Part A, you will implement a simple edge detection routine with signal processing methods. In Part B, you will implement a stochastic color-based segmentation algorithm using a k-Means clustering approximation. In Part C, you will use Part B to do contour detection based on textures.

Part A: Simple Edge Detection

Background

Recall from lecture that segmenting an image is roughly equivalent to finding the boundaries or edges of an image, and that one easy way to do edge detection is by finding the gradient of an image.

One commonly used approximation for the directional derivative of an image is convolution with the Sobel filter. The Sobel filter is a linearly separable filter which approximates the difference of Gaussians. The Sobel filter in the positive-x direction has the following form:

+1	0	-1
+2	0	-2
+1	0	-1

You can derive a similar kernel for the +y direction.

Tasks

For Part A, you will write five functions. To calculate and display gradients (multidimensional derivatives) of images in `texture_gradient.py`. The function headers are as follows:

- `normalizeImage(cvImage, minIn, maxIn, minOut, maxOut)`
- `getDisplayGradient(gradientImage)`
- `takeXGradient(cvImage)`
- `takeYGradient(cvImage)`
- `takeGradientMag(cvImage)`

The `normalizeImage` and `getDisplayGradient` are visualization functions that will help you to make the results of the other three functions human-understandable. The other three functions should use a custom Sobel implementation effectively and efficiently. Make sure to separate any separable kernels, and call Numpy/Scipy/OpenCV functions where applicable.

Implementation notes

You may use most methods from any of the libraries mentioned in class, including convolution methods, but you may *not* use any third-party functions which construct a Sobel filter for you, filter the image with a Sobel kernel you did not create, or take a derivative through some other method. You will also lose points if you do not use the fact that the Sobel operator is linearly separable.

Part B: k-Means Clustering Segmentation

Background

Recall the k-means clustering problem from lecture:

k-Means Clustering Problem

Given some collection of points $(p_1, p_2, \cdots, p_n)$, partition the points into k disjoint segments $(S_1, S_2, \cdots, S_k)$ to minimize the variance of each segment. That is, find

$$\arg\min_{S_1,\cdots,S_k} \sum_{i=1}^{k} \sum_{x \in S_i} \|x - \mu_i\|^2 .$$

where μ_i is the mean of all the points in segment S_i.

Recall also that by rewriting our image as a set of (R, G, B, x, y) tuples we can use a k-means solver for segmentation.

Tasks

For Part B, you will fill in pieces of a k-means segmentation solver in `texture_gradient.py`. You will work on two functions:

- `computeDistance(features, centroids)`
- `getFeats(img)`

The `computeDistance` function should compute the distances between the features and the centroids. The `getFeats` function will return a feature array for each pixel of the image such that a feature is $< R, G, B, x, y >$.

Implementation notes

Your code should weigh pixel R,G,B,x,y values so that the range of values that can be taken on by each channel is the same as the range of values that can be taken on by the x- and y-coordinates.

As in Part A, feel free to use Numpy/Scipy/OpenCV functions as long as they do not trivialize the problem.

Part C: Texture Gradient Contour Detection

Background

One way to detect contours is to measure the difference in local image brightness, color, and texture channels. This technique is inspired by this *paper* that introduced the idea of textons. Recall from lecture that textons find canonical local features in a texture by filtering an image with linear filters from a filter bank.

These filters detect orientation of edges, spots, etc which form repeated patterns in a given patch. These patterns are clustered together and their centers are called textons. Each pixel is labeled to a texton i (1 to K) that is most similar in appearance. To identify boundaries, a disc around each pixel is considered as its neighborhood. The disc is split into two halves, where the cut lies on the boundary of the greatest change in pixel intensity. In **texture_gradient.py**, the halves are referred to as *masks* and are used to compute G, a 4D image by subtracting the orientation histograms of each disk. For part C, you will be implementing the aforementioned algorithm.

Tasks

This is the technique you will implement in Part C of this programming assignment. You will fill in the following five functions in `texture_gradient.py`:

- `getOrientedFilter(f, theta, size)`
- `getDoG(x, y, sigma_1, sigma_2)`

- `getTextons(img, vocab_size)`
- `getMasks(theta, size)`
- `computeTextureGradient(img, vocab_size, r)`

In `getOrientedFilter`, you will generate an oriented Gaussian filter. In `getDoG`, you will generate a difference of Gaussians by subtracting two Gaussians, commonly known as the Mexican hat. In `getTextons`, you will get textons and assign each pixel to a texton. In `getMasks`, you will divide a square neighborhood into two halves depending on the orientation. Finally, in `computeTextureGradient`, you will compute the texture gradient by taking the difference of histogarms in each of the two halves from `getMasks`.

Implementation notes

As in Part A, feel free to use Numpy/Scipy/OpenCV functions as long as they do not trivialize the problem.

Deliverables

You should submit two files, `texture_gradient.py` with all of your work, and a **.txt file** explaining your submission. You should fill in every function marked with `# TODO:PA2` in `texture_gradient.py`.

Testing and Debugging

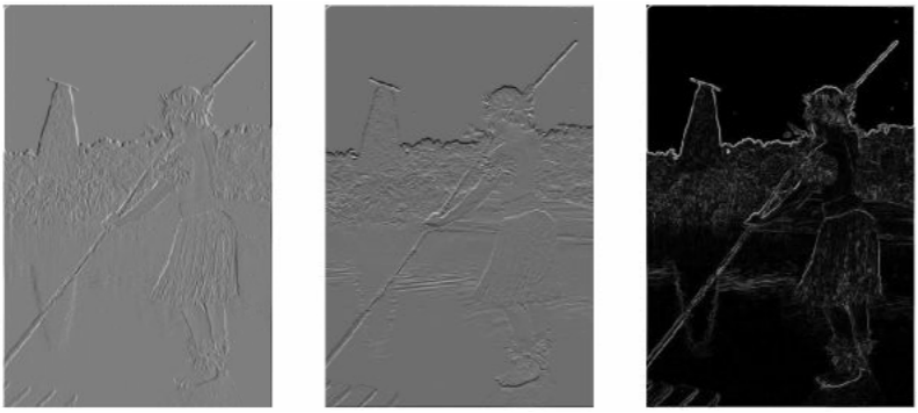
Jupyter Notebook

You are provided a file named **visualizations.ipynb** that will be helpful in debugging your code. It contains visualizations of any filters/images necessary for parts A, B and C. Before you can run this, make sure you have jupyter notebook *installed*. If you are using Anaconda, you should already have jupyter notebook installed. If not, you can install it through the Anaconda GUI.

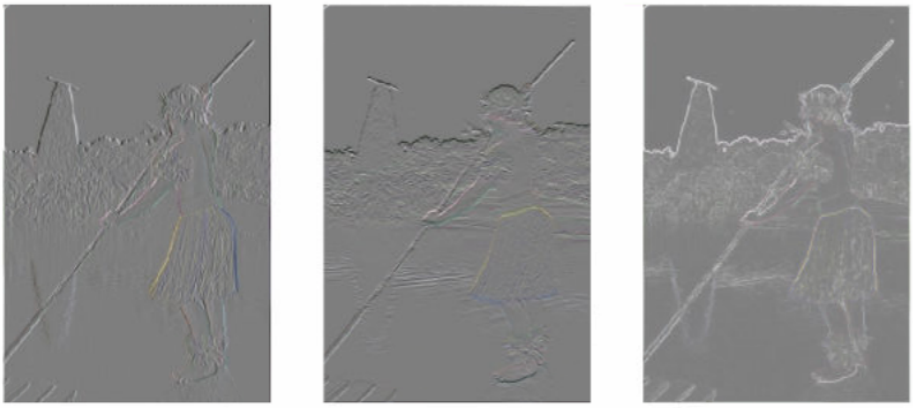
We have compiled all correct visualizations below. Your visualizations should closely resemble these:

Visualizations

Part A: X,Y gradients and gradient magnitude of grayscale image

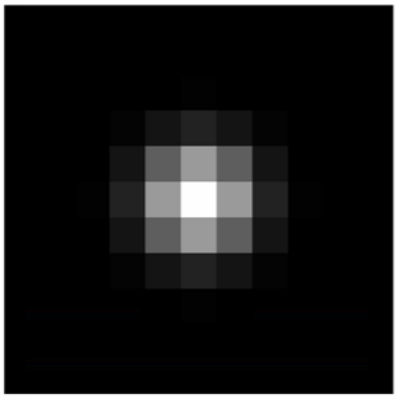


Part A: X,Y gradients and gradient magnitude of rgb image



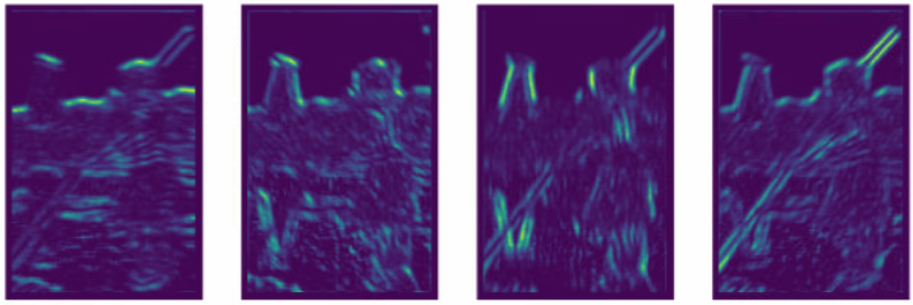
Part B: k-Means Segmentation

Part C: Gaussian Oriented Filter

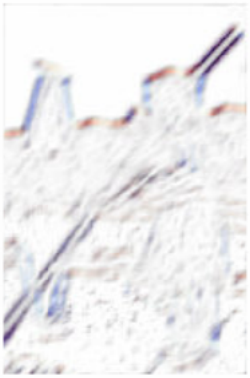


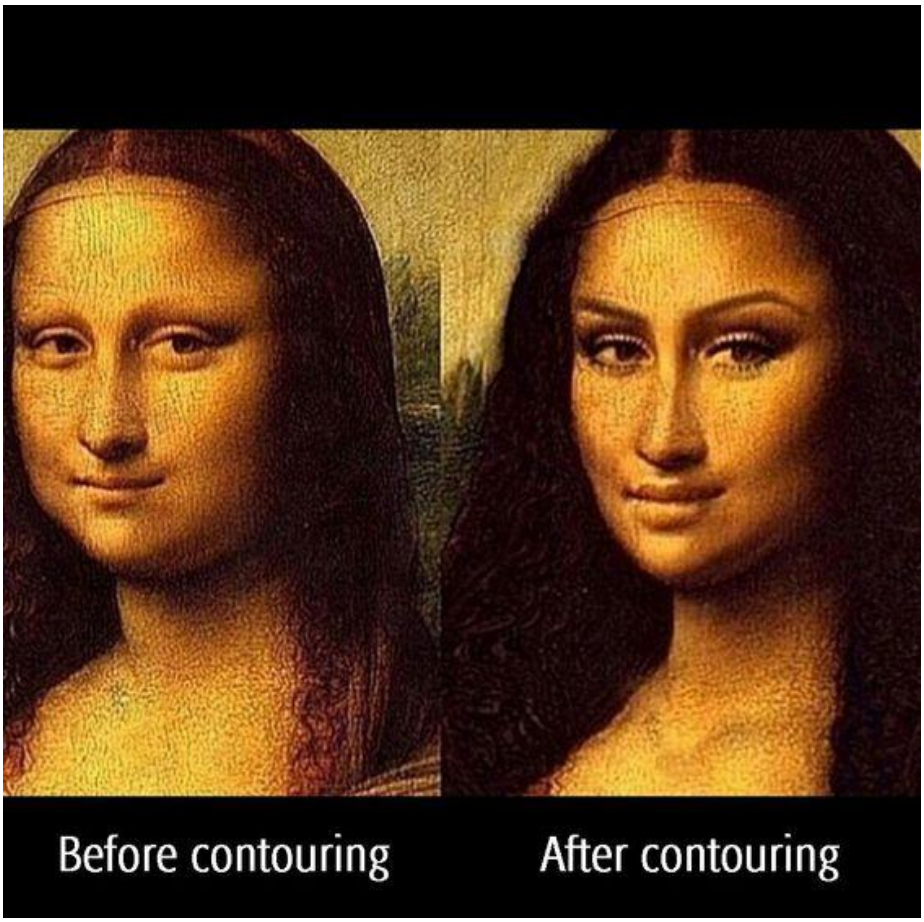
Part C: Filter Bank

Part C: 4-channel image G



Part C: Contours




Before contouring
After contouring