CSCI 6651 Homework 1 Summary
Grant Matthews

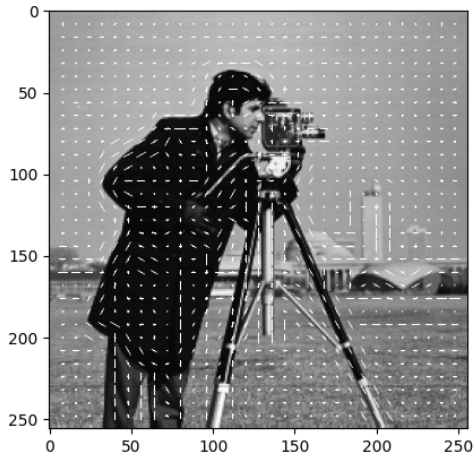
To build a HOG, I first need to calculate the change in intensity over a pixel in both the X and Y directions. I used two 3x3 kernels and moved them over the image to create two new images where each pixel value corresponded to the change in intensity in either the X or Y direction.

With the two new images, one representing gradient in the X direction and the other in the Y direction, I was able to create two new images. These images had pixel values corresponding to the angle and magnitude of the gradient at each pixel. Using simple Pythagorean math on the X and Y gradient intensities, I was able to calculate the angle at which the gradient was pointing (using arc tangent) and the intensity of the gradient in that direction (using Pythagoras theorem).

The core of the HOG algorithm lies in its ability to compress this information for easier processing. I split the image into blocks using a specified block size (in this instance 8x8 pixels) and I built a histogram for each block that contained 6 bins. Each bin corresponds to 30 degrees (out of 180). For each pixel in the block, I added the gradient magnitude to the bin that corresponded to the gradient's angle.

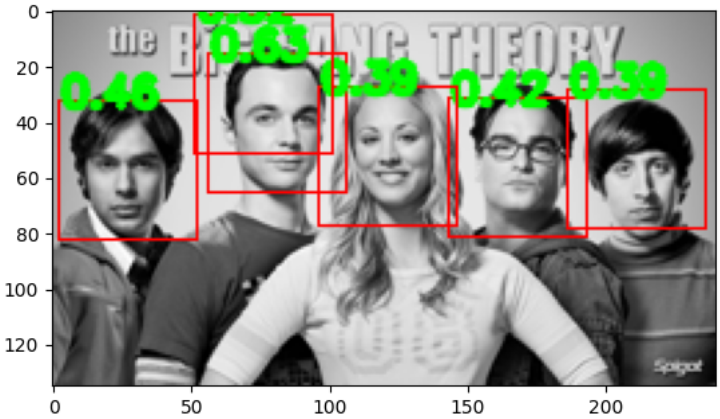
To complete the compression of the image, I created descriptors using every possible combination of adjacent 2x2 blocks. Each descriptor is just a concatenation of the 4 (2x2) 6 value arrays storing the histogram data for each block.

The visualization of this is below, where each white "star" is a descriptor and the size of the lines in each direction represents the total magnitude of the gradients in that direction across all the pixels in that descriptor.



The facial recognition algorithm builds off of my HOG algorithm. It first takes a template image and builds a HOG for it. Then, my algorithm takes a test or target image and iterates over every possible subimage that is the same size as my template image. For each of these sub images it also builds a HOG. It then compares each HOG using the formula for S scores to determine vector similarity. Each sub image that has a similarity score above a certain threshold (currently set at 0.3) has its position saved as a potential bounding box around a face in the image.

This generates a lot of potential bounding boxes so I used non max suppression to find the highest scoring bounding boxes and remove any boxes that share at least 50% of the same area. This left me with the detection visualized below.



As you can see there are two bounding boxes around Sheldon's head. This is because both were valid bounding boxes with S scores above 0.3 but they did not overlap by more than 50%. Thus the algorithm has detected Sheldon twice. This can be potentially avoided by modifying the threshold value and the area value.