Writeup1

David Ma

February 7, 2020

1 Summary

This project can be split into two components: generating the HOG, and then using the HOG for face detection.

1.1 Generating the HOG

I started off by differentiating the picture in the X and Y direction via a 3x3 matrix. The filters can be found in the function get_differential_filter, which is used in the filter_image function to produce X and Y filtered images.

Given X and Y filtered images, we can calculate the angle and magnitude of the gradient for each pixel by combining the two differentials. The angle is calculated as arctan(Y/X), and the magnitude is the norm of the two differentials, as seen in the function get_gradient.

Once we have a gradient value at each pixel, we can compile them into a Histogram of Oriented Gradients, as seen in the function build_histogram. For each cell, which is a collection of pixels, we count the number of gradients that point in each direction and weight them by their magnitudes. Finally, we normalize with get_block_descriptor and concatenate them into a single vector with extract_HOG.

1.2 Face Detection

Once we have the extract_HOG function, we can use it for face recognition. Given a template image, we find its HOG as a vector. We then slice the target image into subimages of the same size and dimensions of the template and find those HOGs. Next, for each subimage's HOG, we find the cross correlation score with the template's HOG, which is effectively a normalized dot product. If the correlation score is sufficiently high, we consider that to be a face detected. Finally, to clean up the detected faces, we run non-maximum suppression, which is where we ignore detected faces that are too close together. I found a IoU threshold of .3 worked the best, as seen by the results below. Using only the HOG and the IoU gave the image on the left, and just a little bit more cleanup gave the image on the right.

2 Results