HW5 Write-Up: Image Understanding

16-720: Computer Vision

Spring 2016

Cole Gulino

April 12, 2016

1. Image Detection Using Histogram of Gradients

1.1: Image Gradients

Write a function that takes a grayscale image as input and returns two arrays the same size as the image, the first of which contains the magnitude of the image gradient at each pixel and the second containing the orientation, with the function signature:

```
[mag, ori] = mygradient(I)
```

For this function, I calculated the gradient and then calculated the orientation and the magnitude. Figure 1 and 2 shows the gradient magnitude and orientation of the gradient for `image0.jpg` and `image1.jpg`.

I passed the orientations using `atan2(Gy, Gx)` which gave me values in the range of $(-\pi, \pi)$.

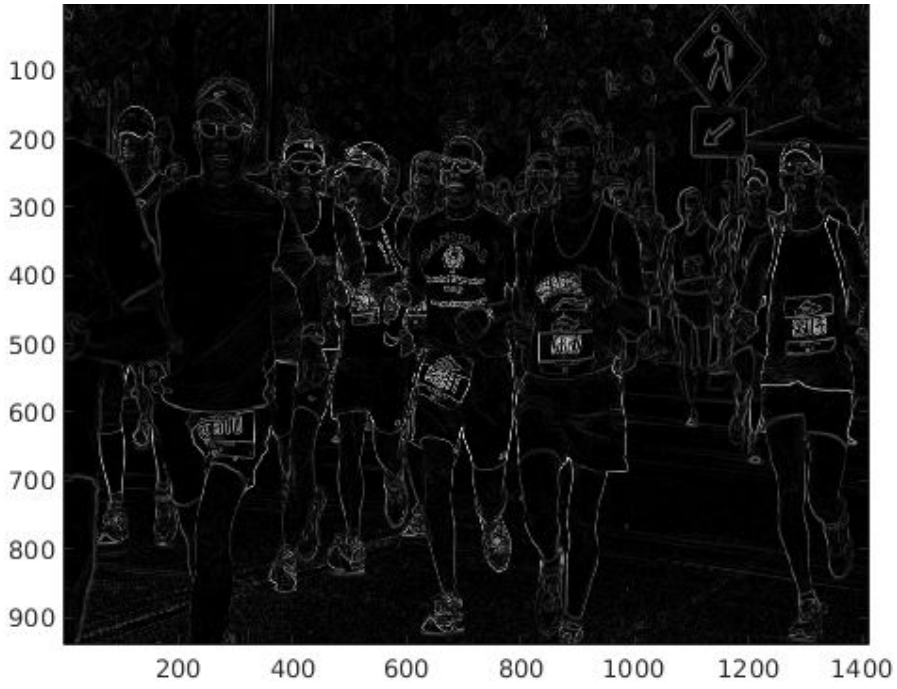

Figure 1a) Gradient Magnitude of `test0.jpg`

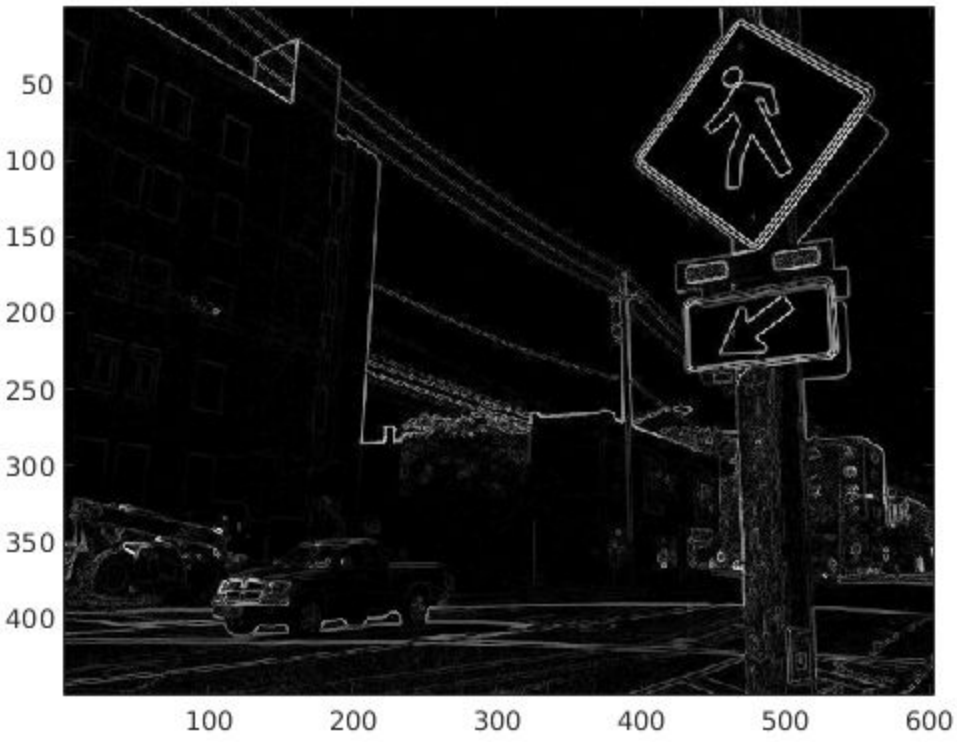

Figure 1b) Gradient Magnitude of `test1.jpg`

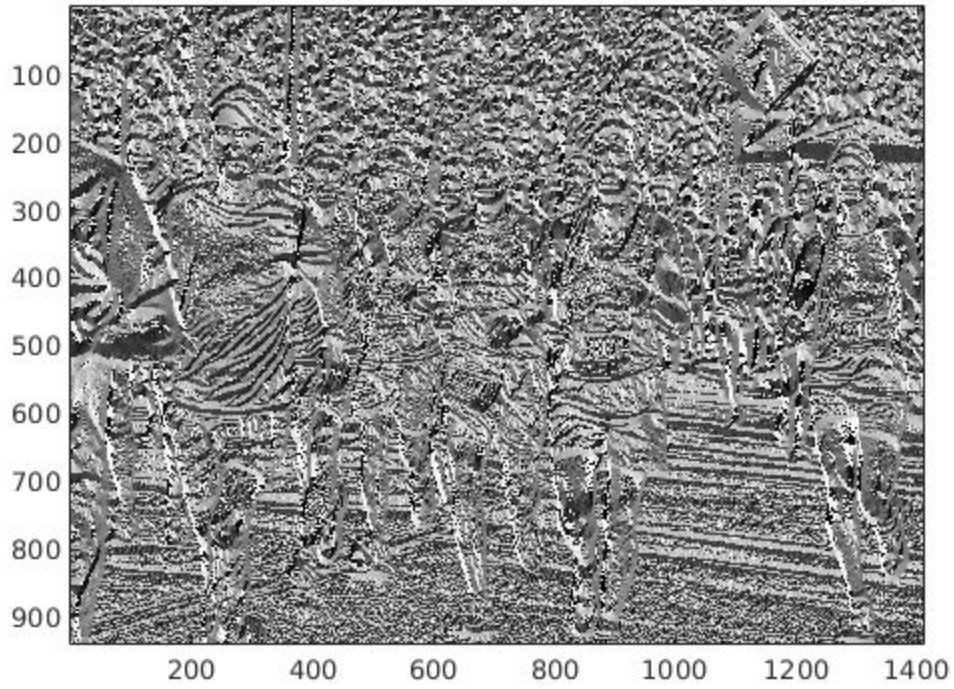

Figure 2a) Gradient Orientation of `test0.jpg`

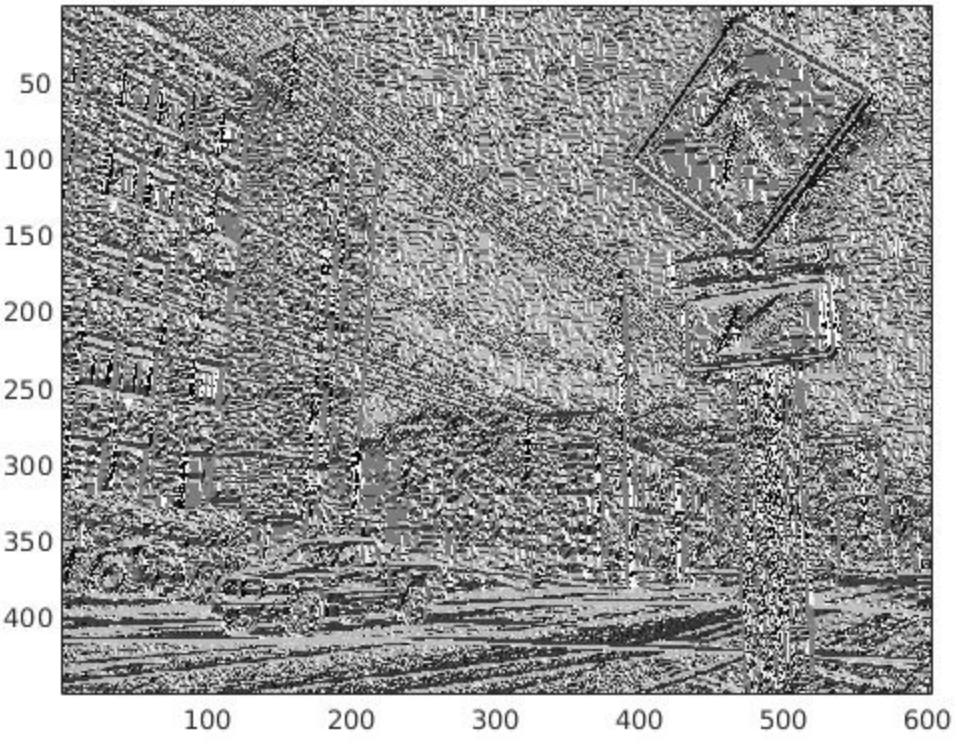

Figure 2b) Gradient Orientation of `test1.jpg`

1.2: Histograms of Gradient Orientations

Write a function that computes gradient orientation histograms over each 8x8 block of pixels. Your function should bin the orientation into 9 equal sized bins between -pi/2 and pi/2. The input of your function will be an image of size HxW. The output should be a three-dimensional array ohist whose size is (H/8)x(W/8)x9 where ohist(i,j,k) contains the count of how many edges of orientation k fell in block (i,j). With the function signature:

```
ohist = hog(I)
```

For section 1.2, I used these values as the thresholding values:

$$(\text{gradient magnitude})_i > 0.1 * \max(\text{gradient magnitude}(:))$$

$$\text{gradient orientation bucket min} \leq (\text{gradient orientation})_i < \text{gradient orientation bucket max}$$

I then took these binary values from logicals and broke them up into 8x8 blocks and summed up the values. Figure 3 shows an example of the output:

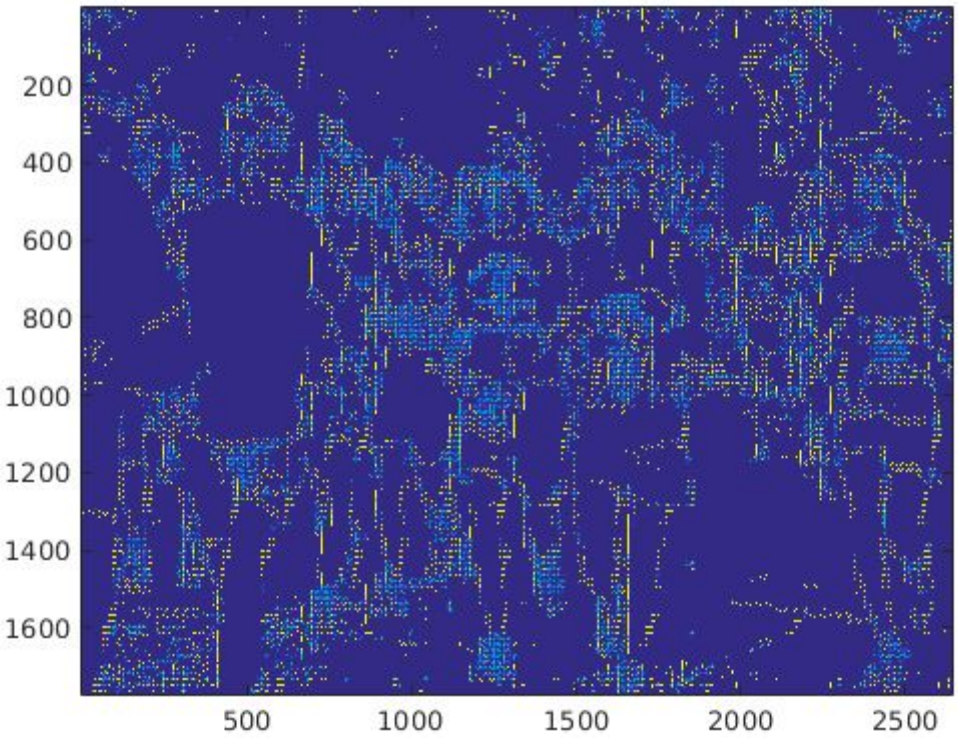

Figure 3a) HOG Feature Visualization for `test0.jpg`

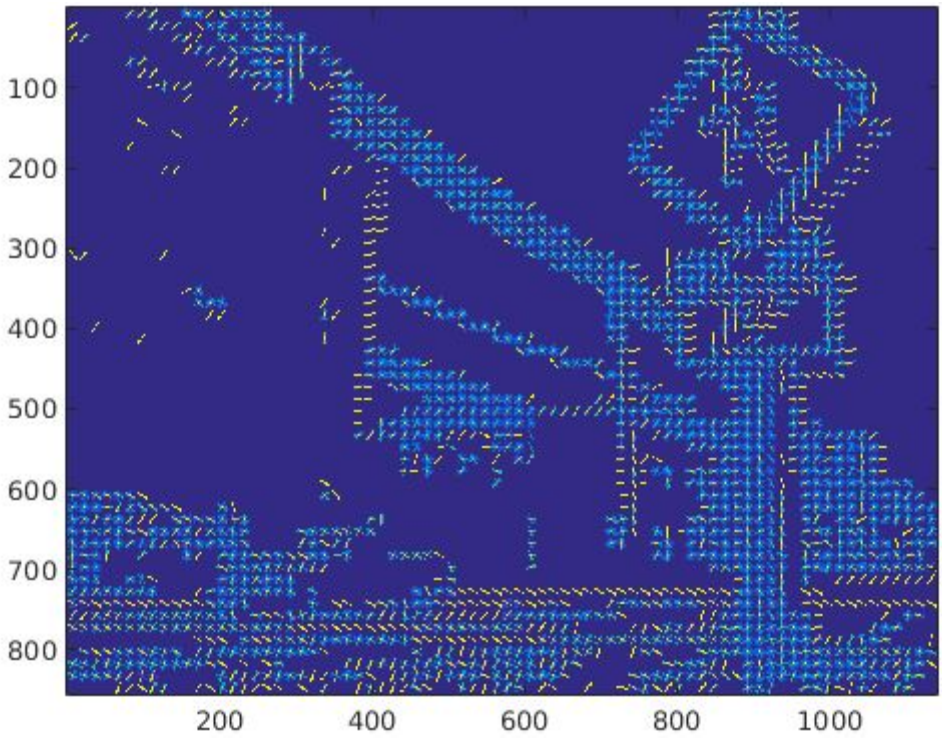

Figure 3b) HOG Feature Visualization for `test1.jpg`

1.3: Detection

Write a function that takes a template and an image and returns the top detections found in the image. Your function should have the prototype:

```
[x, y, score] = detect(I, template, ndet)
```

In this function, I first calculate the histogram of oriented gradients for the image. I then correlate the each orientation information for each image using the function:

```
filter2(template(:,:,1), ohist(:,:,1), 'same')
```

I then sum each of the correlations to get one score for each pixel.

I then sort the values in descending order and ensure that each new x and y value are not too close to another value already found in a function with the signature:

```
Function [bool] = xy_unique(x_new, y_new, x_vals, y_vals, template)
```

This function is included in the `detect` function.

Figure 4 shows the heatmap visualization of correlation, and Figure 5 shows the heatmap as a height field with a clear peak.

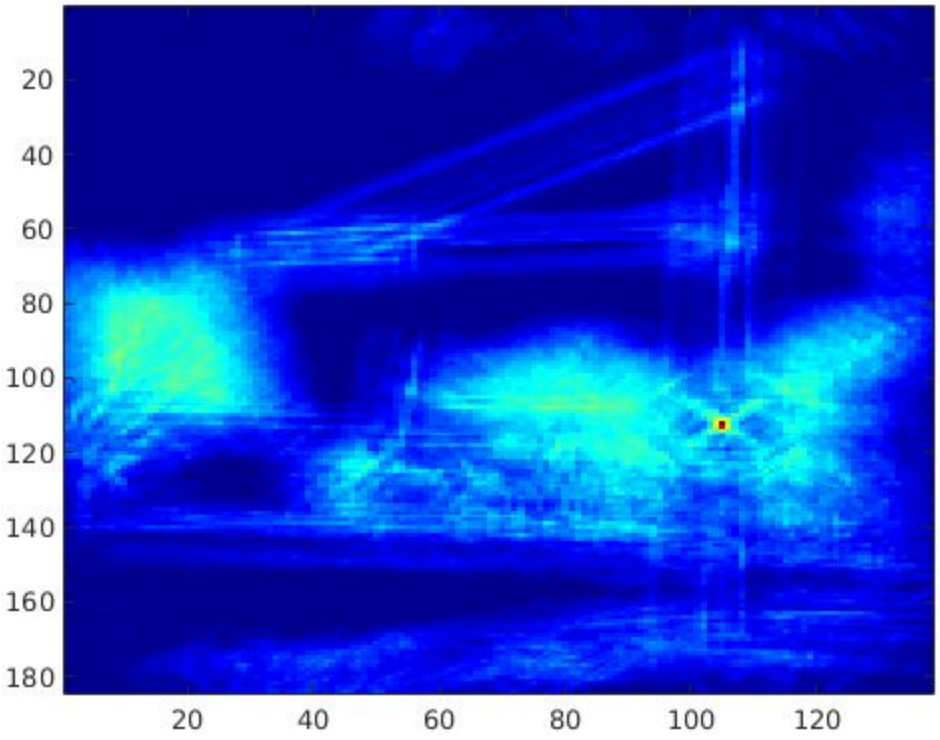

Figure 4) Heatmap of Correlation for `test3.jpg`

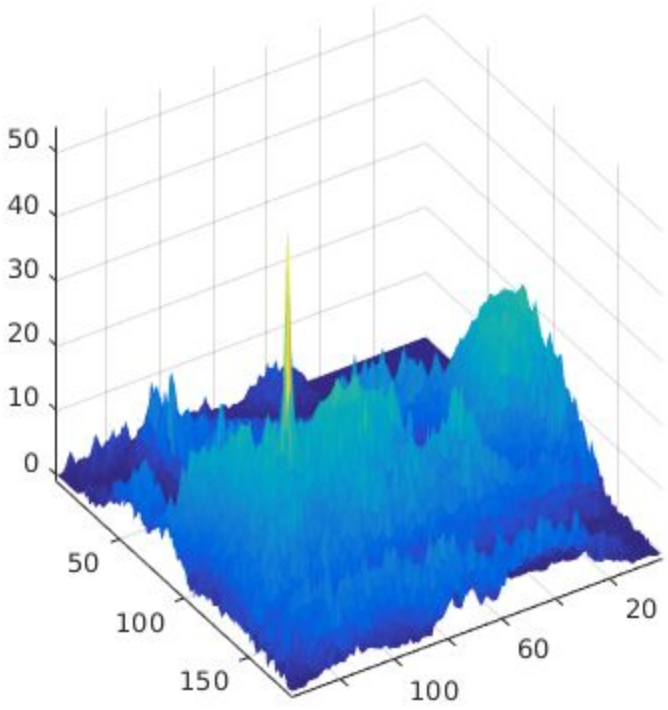

Figure 5) Heatmap Shown as a Height Field with Peak at Detection for `test3.jpg`

Figure 6 shows the window of detection for the image:

Figure 6) Window of Detection for `test3.jpg`

Figure 7 shows the Heatmap of correlation for the rest of the test images and the window of detection for the rest of the images.

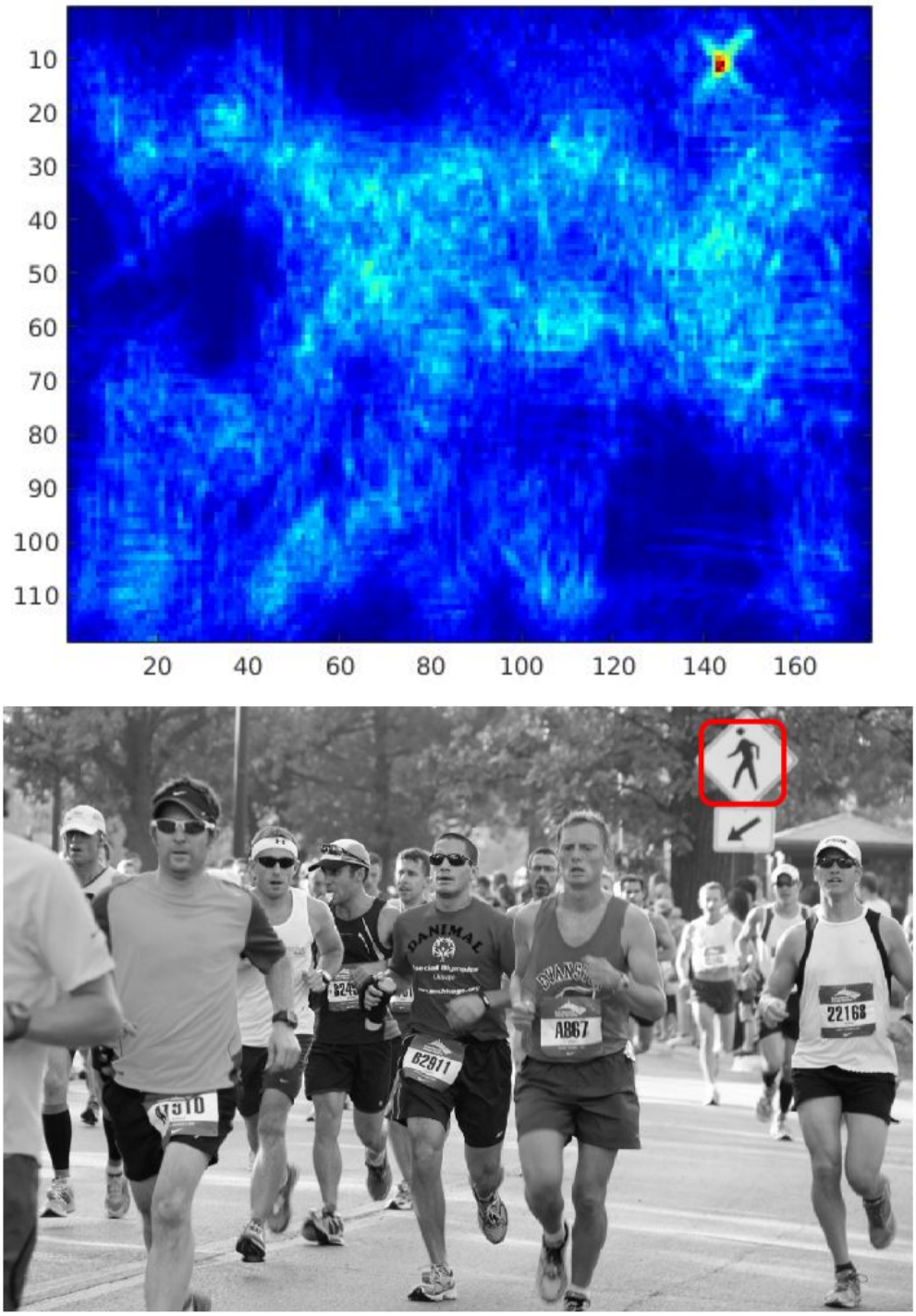

Figure 7a) Heatmap of Correlation and Window of Detection for `test0.jpg`

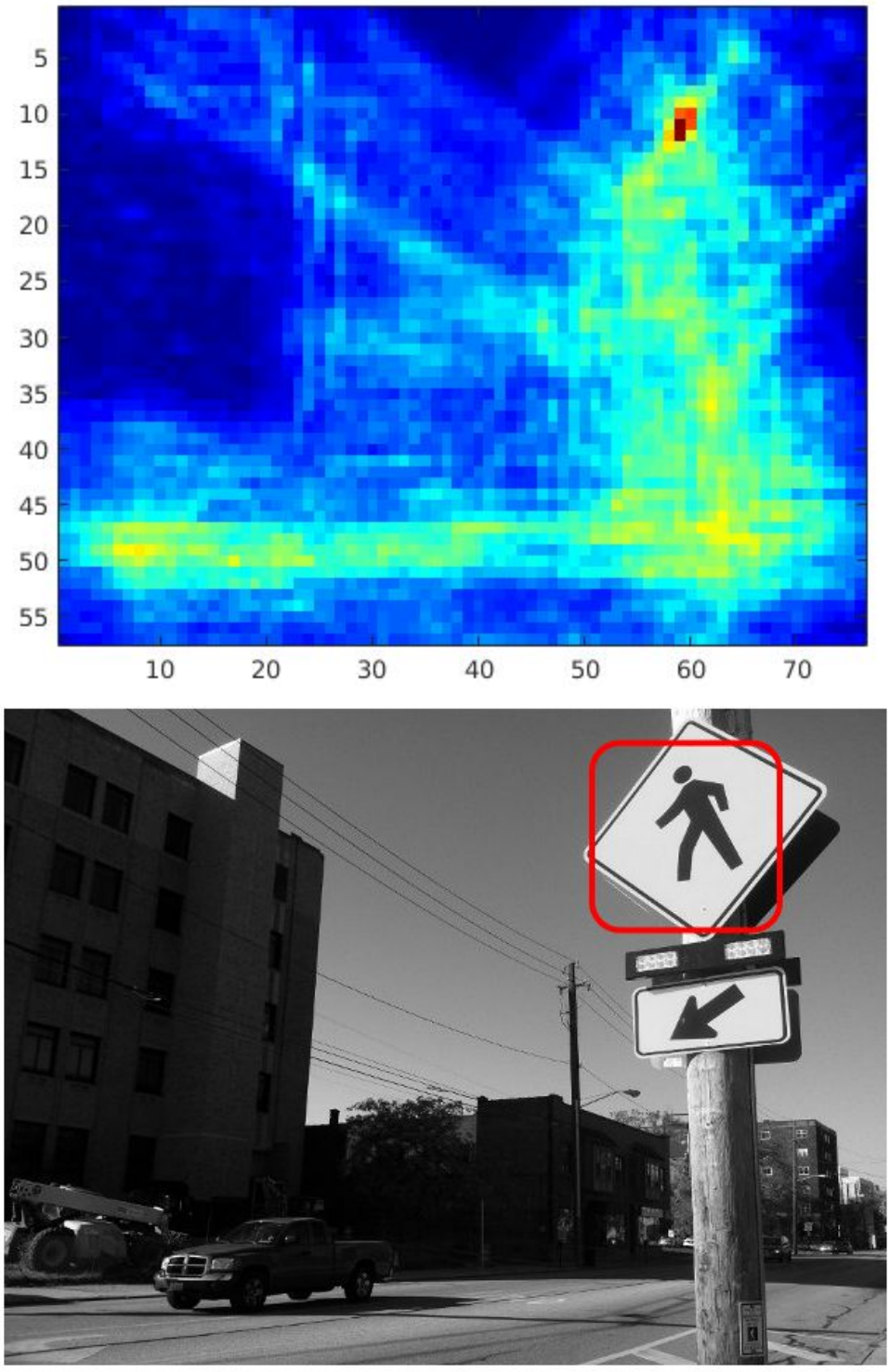

Figure 7b) Heatmap of Correlation and Window of Detection for `test1.jpg`

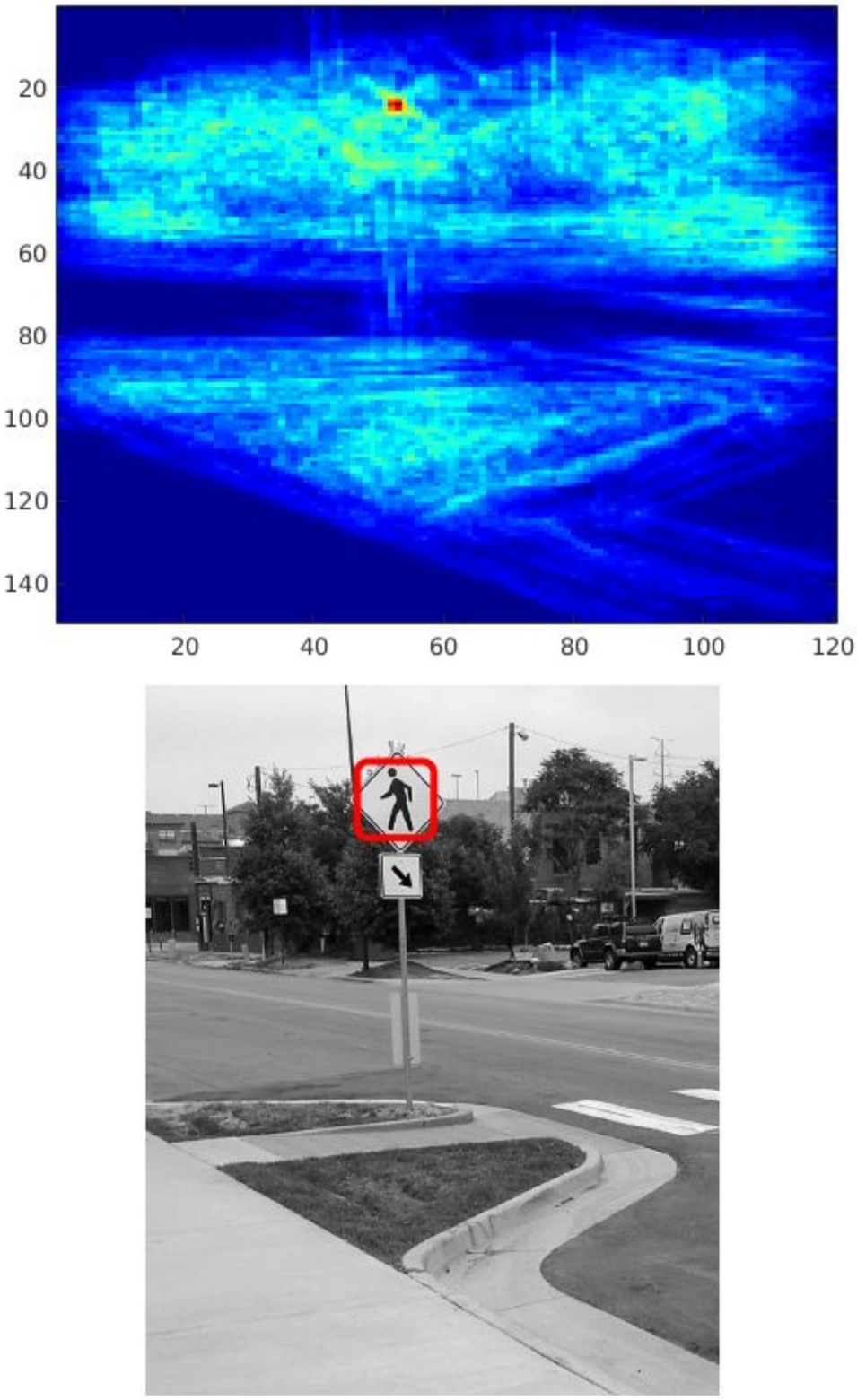

Figure 7c) Heatmap of Correlation and Window of Detection for `test4.jpg`

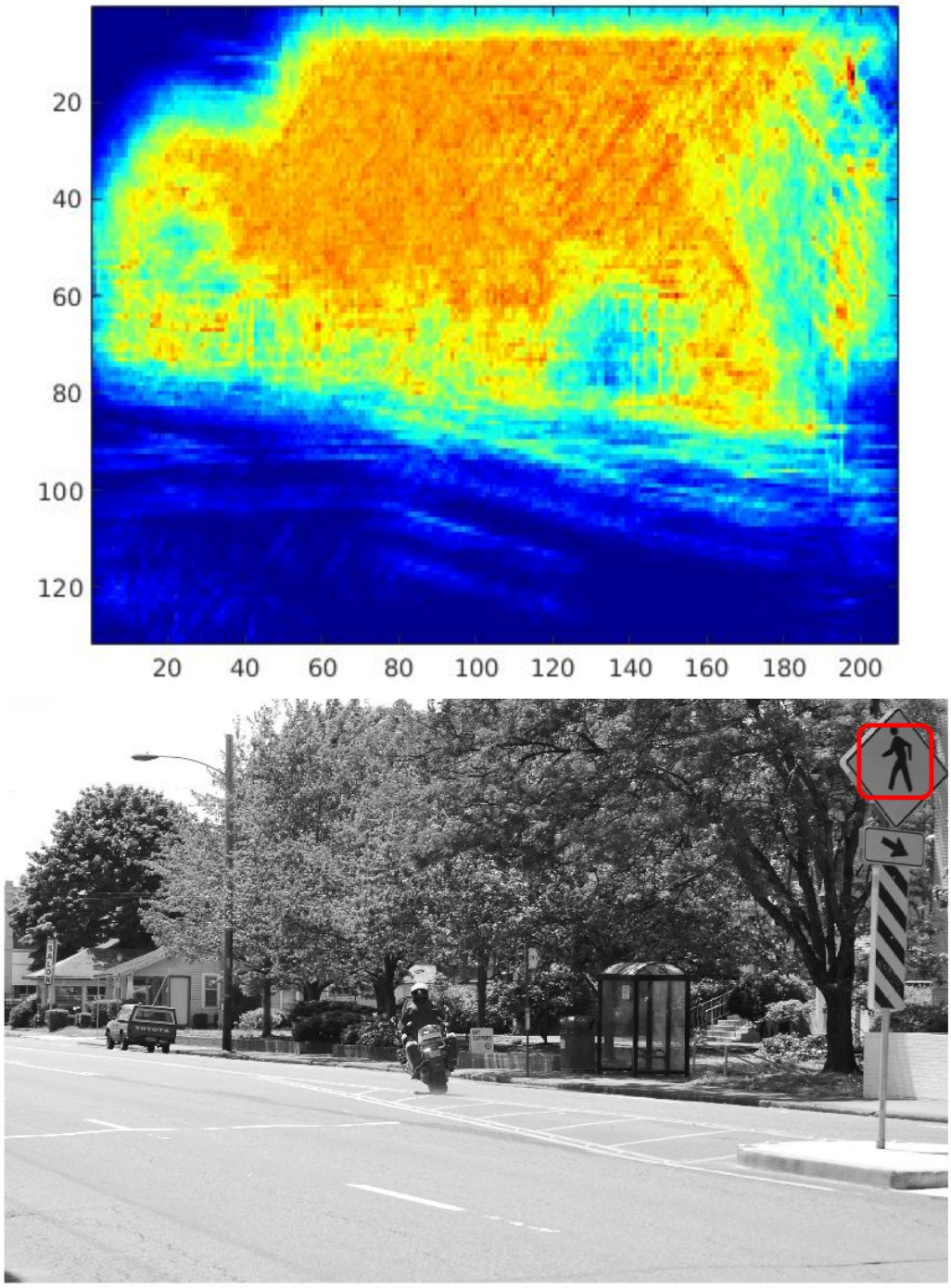

Figure 7d) Heatmap of Correlation and Window of Detection for `test5.jpg`

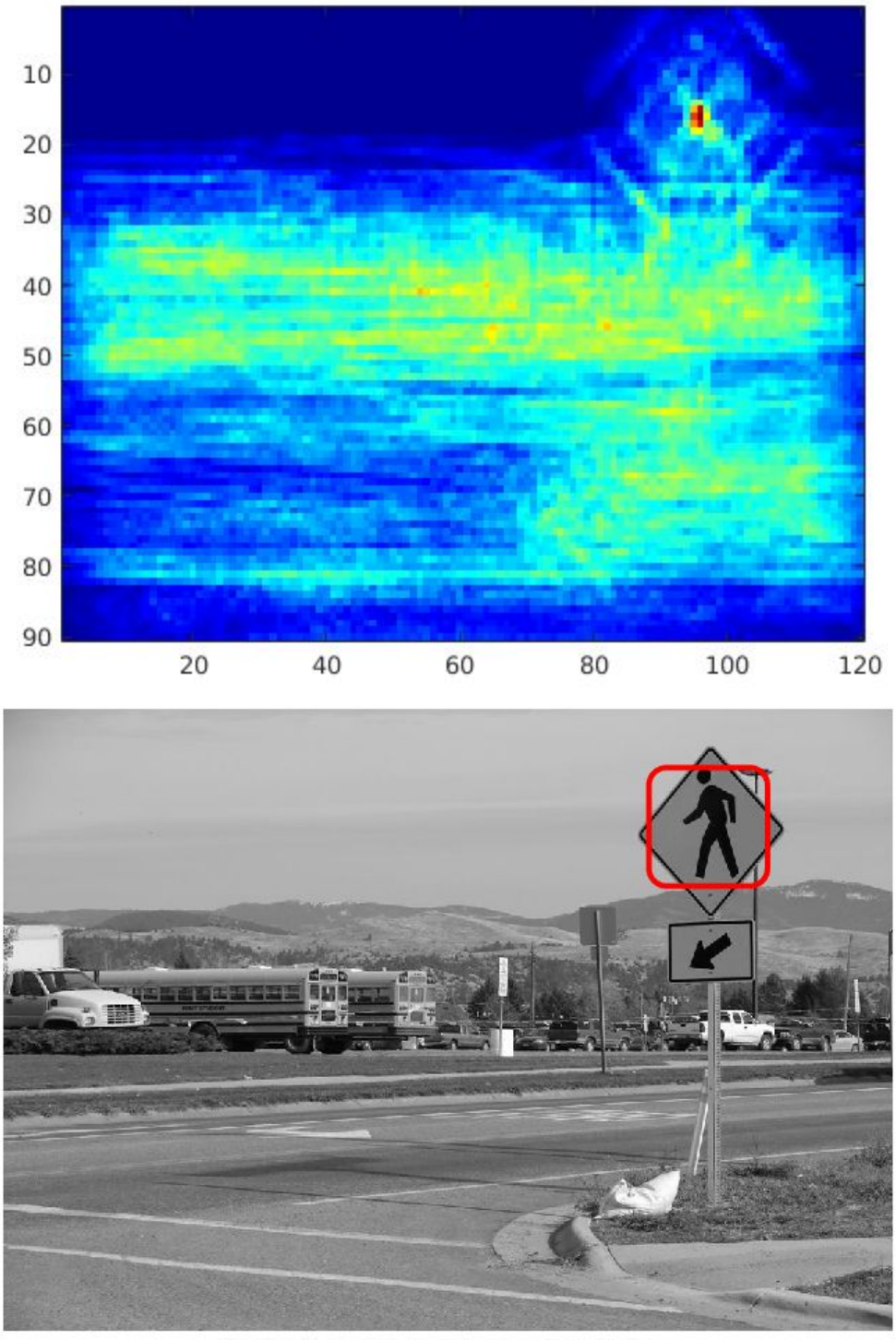

Figure 7e) Heatmap of Correlation and Window of Detection for `test6.jpg`

1.4: Extra Credit - Multiple Detections

For this section I used another image of runners to try and get a face detector. The image I used to train this with is shown in Figure 8:

Figure 8) Face Detector Training Image

The detection and correlation heatmap of the faces in the test image are shown in Figure 9:

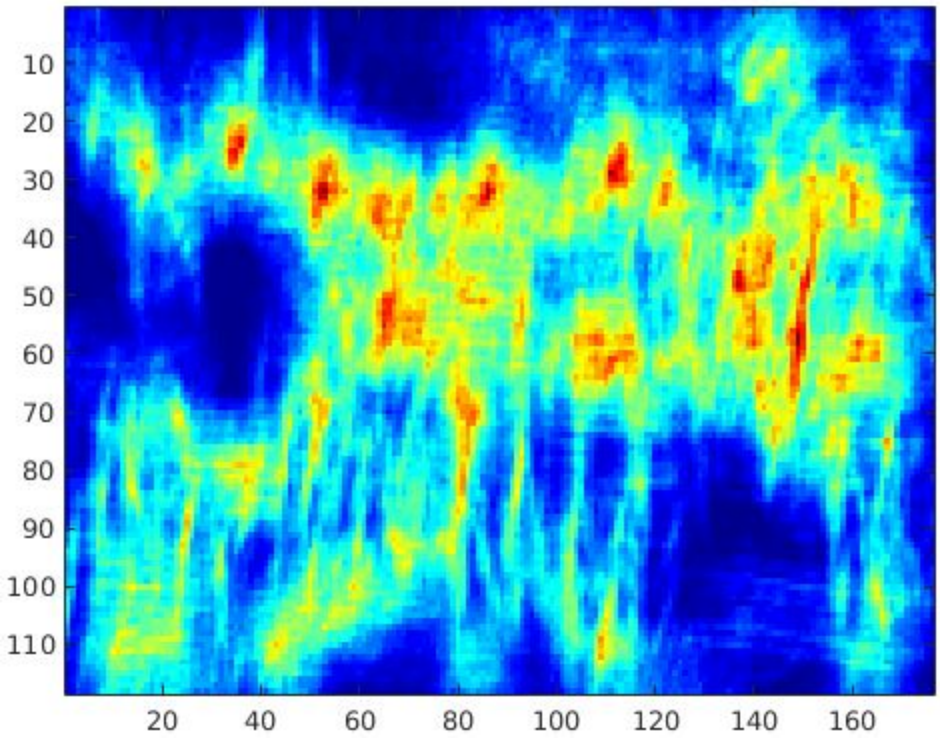

Figure 9a) Multiple Face Detections Correlation Heatmap

Figure 9b) Multiple Face Detections

This shows the 5 best detections. There was one mis-detection as seen on the left-side.

2. Learning Templates

2.1: Select Patches

Write a script or function that allows the user to select patches. In the end of the script, the patches should be saved for training purposes. This should let the user mark examples from several images. The script is called:

`select_patches.m`

I decided to implement a script instead of a function. The script is broken up into parts. The first part requests the user to specify the images that it wants to load. The user should then place all of the images they want to run in the cell called `Itrains`. Once the training data is placed into the cell, the user should place the indexes of the cells that they want to run in the array `images_to_run`.

The next section requires the user to specify the number of positive and negative training examples to identify in the image with the variables `nPositive` and `nNegative`.

Once these are specified, the images will be presented and the bounding boxes can be drawn by the user. Positive examples can be drawn first and will show up green after the user draws them. After all of the positive examples have been drawn, the negative examples can be drawn. These will show up as red to the user.

After this, the image boxes will be gathered and resized to be the average of the size of all the templates. The positive templates will be ensured to be a multiple of 8.

For the training set I used the images: `test0.jpg, test1.jpg, test2.jpg, test4.jpg, test5.jpg, test6.jpg`.

For the test image, I used: `test3.jpg`.

I wrote two functions for writing the detect script. One version called `draw_detection` that provides the signature specified on Piazza. The other function is called `draw_detection_upscale` that changes how it scales the bounding boxes if I have to upscale the image to get smaller detections.

2.2: Positive Template Learning

I implemented a function called `tl_pos.m` that takes in the positive template matches found in section 2.1.

It then averages the HOG features for all of the templates and provides one average template for the image, which it then returns.

For this function, I used the average of the templates that I learned:

$$T_{pos} = \frac{1}{|T_{pos}|} \sum_{i \in T_{pos}} T_i$$

Figure 10 shows the heatmap of correlation and window detection on the test image: `test3.jpg`.

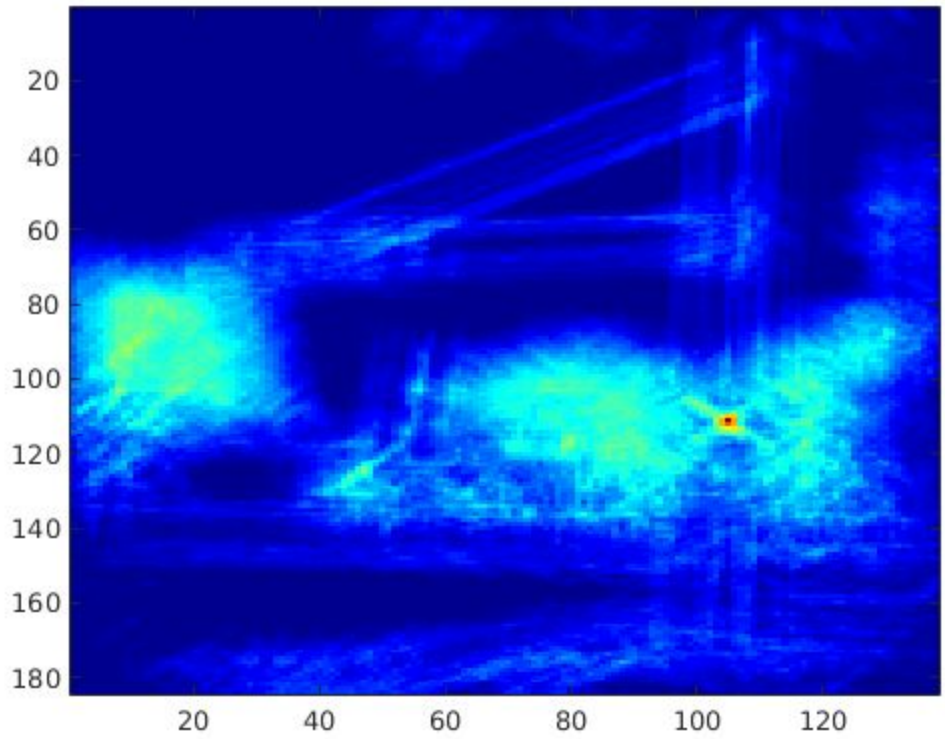

Figure 10a) Heatmap of Correlation for `test3.jpg` using Positive Template Learning

Figure 10b) Detection for `test3.jpg` using Positive Template Learning

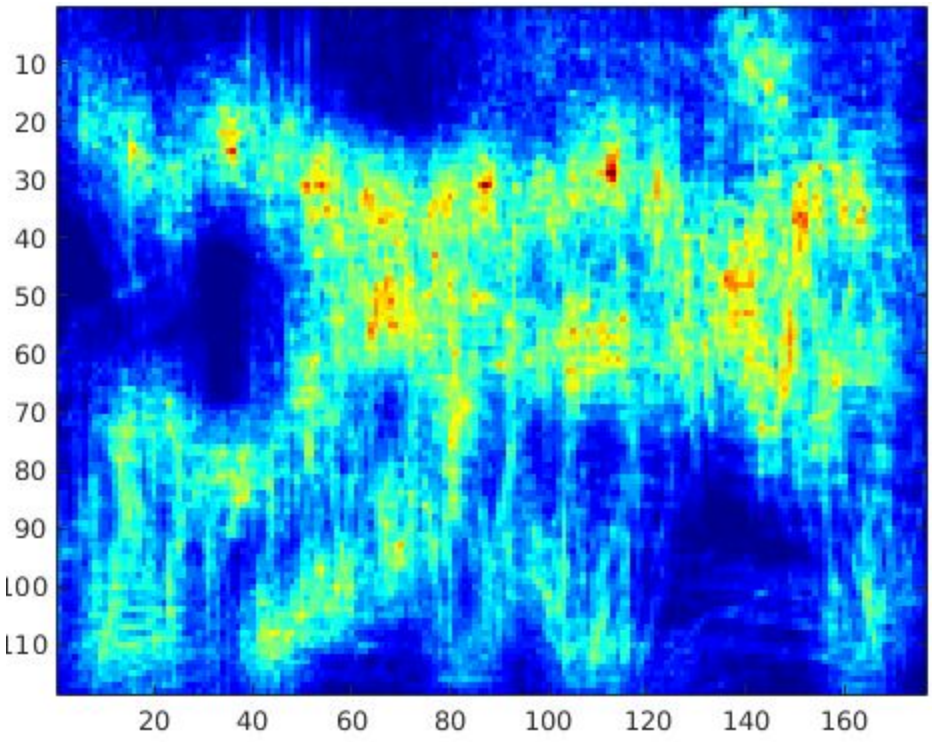

Figure 10c) Heatmap of Correlation for `test0.jpg` using Positive Template Learning

Figure 10d) Detection for `test0.jpg` using Positive Template Learning

Positive templates for the sign and the faces are shown in Figure 18 and 19.

I also used the positive template learning on stop signs. My positive templates are shown in Figure 11. These templates are saved in `template_images_stop_sign_pos.mat` with negatives in `template_images_stop_sign_neg.mat`

Figure 11) Positive Templates for the Stop Sign

Figure 12 shows two test examples using positive template learning for the stop sign.

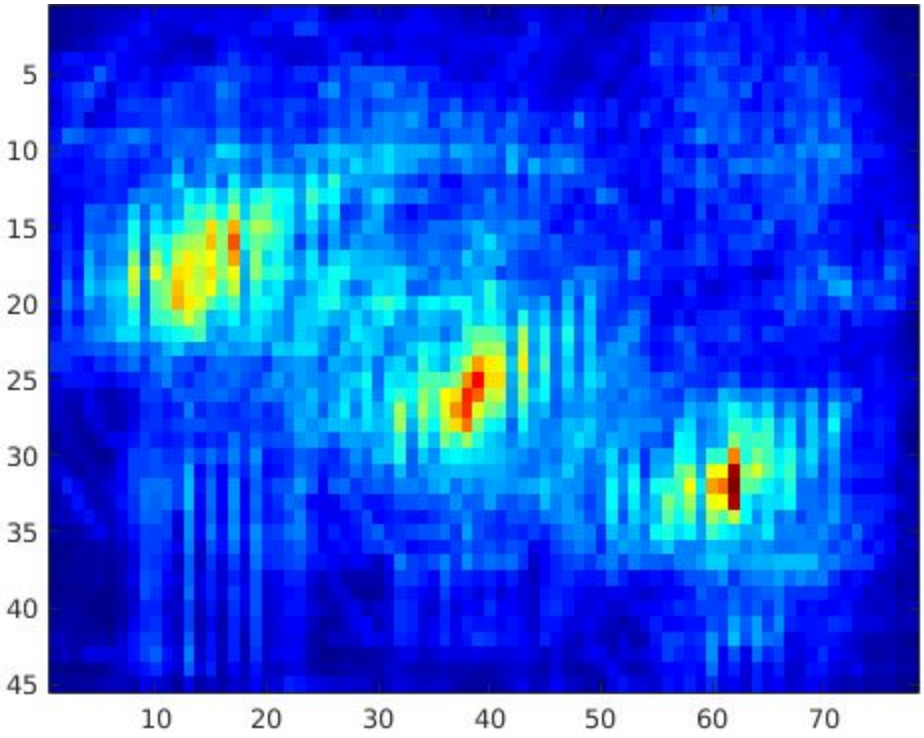

Figure 12a) Correlation Heatmap for `stop_sign_test2.jpg` using Positive Template Learning

Figure 12b) Top Five Detections for `stop_sign_test2.jpg` using Positive Template Learning

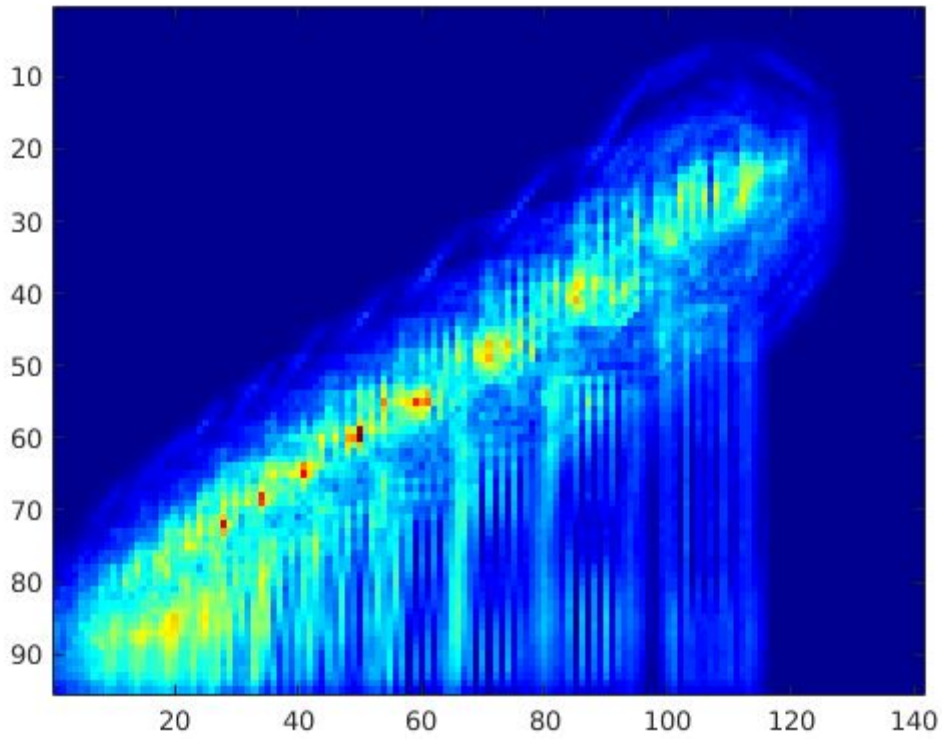

Figure 12c) Correlation Heatmap for `stop_sign_test3.jpg` using Positive Template Learning

Figure 12d) Top Five Detections for `stop_sign_test3.jpg` using Positive Template Learning

2.3: Positive Negative Template Learning

I implemented the functionality in `tl_pos_neg.m`. I used the equation shown below from the homework prompt:

$$T_{final} = \mu_{pos} - \mu_{neg} \quad \text{where} \quad \mu_{set} = \frac{1}{|set|} \sum_{i \in set} T_i$$

Figure 13 shows the heatmap of correlation and window detection on the test images for the crosswalk sign, the faces, and the stop sign.

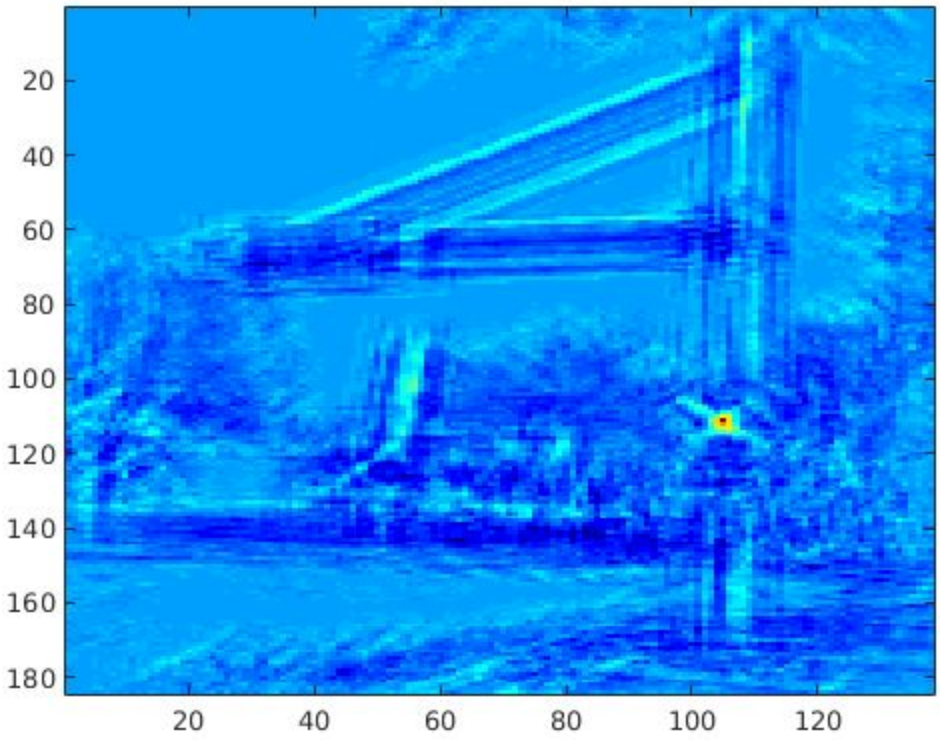

Figure 13a) Heatmap of Correlation for test3.jpg using Positive Negative Template Learning

Figure 13b) Detection for `test3.jpg` using Positive Negative Template Learning

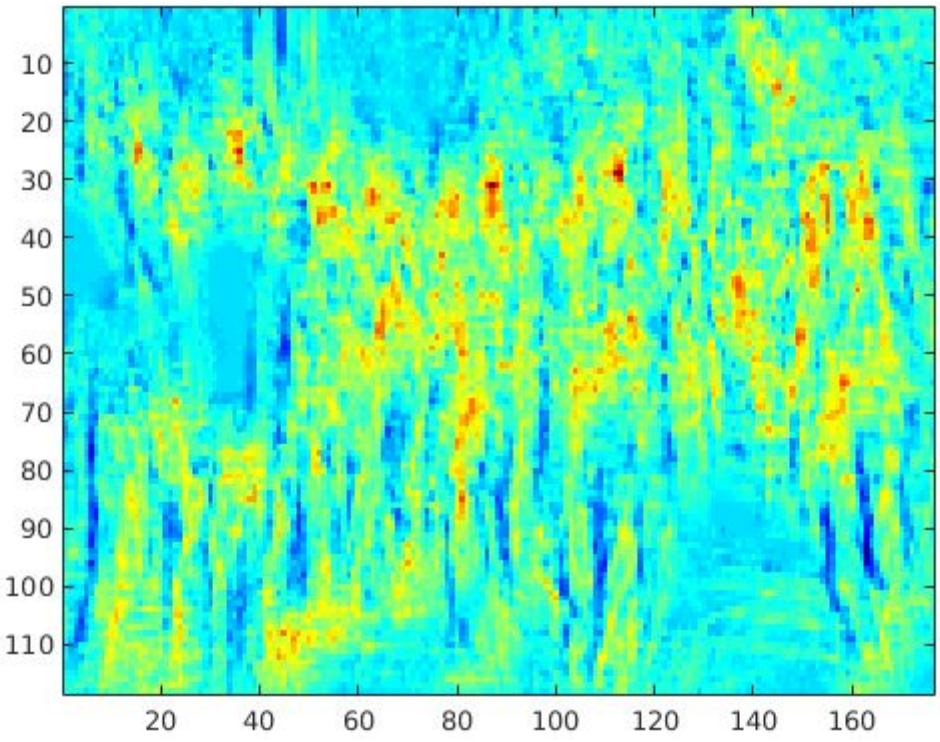

Figure 13c) Heatmap of Correlation for test0.jpg using Positive Negative Template Learning

Figure 13d) Detection for `test0.jpg` using Positive Negative Template Learning

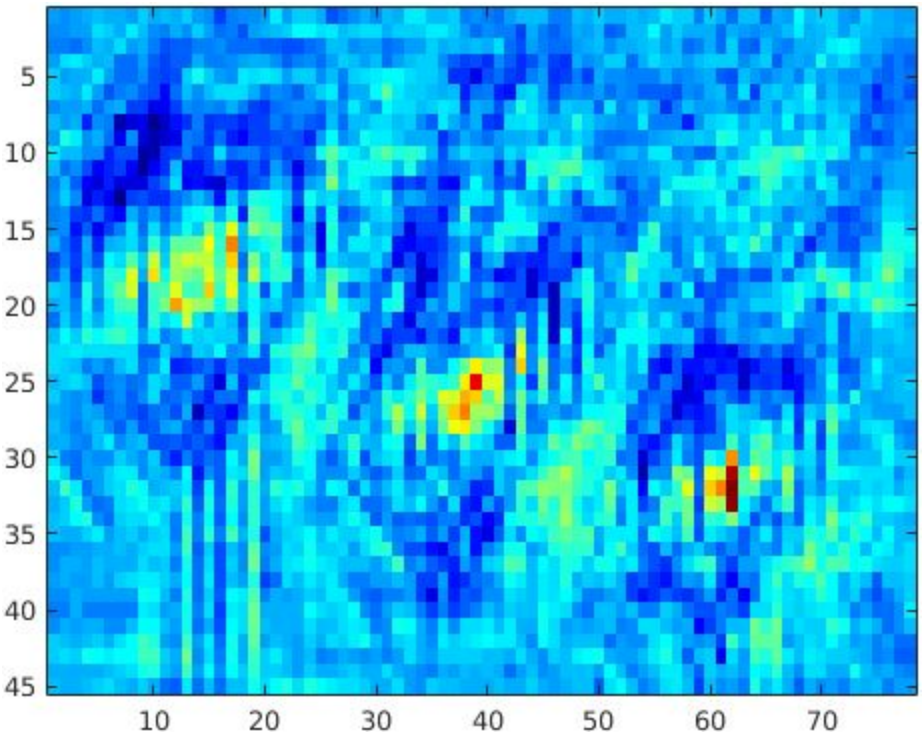

Figure 13e) Correlation Heatmap for `stop_sign_test2.jpg` Positive-Negative Template Learning

Figure 13f) Top Five Detections for `stop_sign_test2.jpg` using Positive-Negative Template Learning

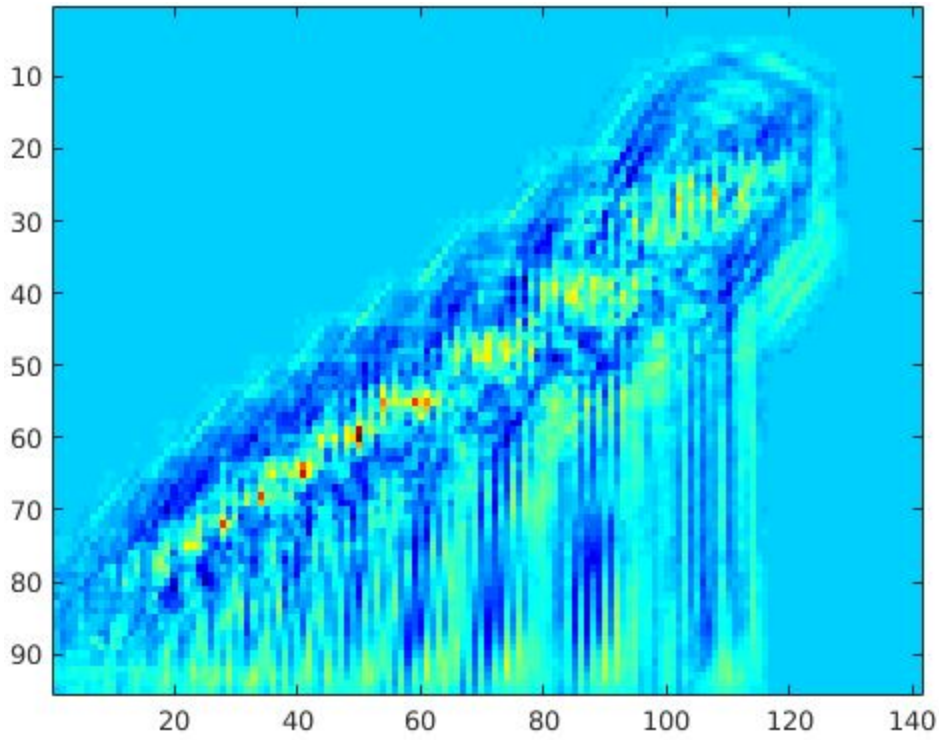

Figure 13g) Correlation Heatmap for `stop_sign_test3.jpg` Positive-Negative Template Learning

Figure 13h) Top Five Detections for `stop_sign_test3.jpg` using Positive Negative Template Learning

2.4: LDA Template

I implemented the functionality in `tl_lda.m`. I used the equation shown below from the homework prompt with insight from [1].

$$T_{final} = \Sigma_{neg}^{-1}(\mu_{pos} - \mu_{neg}) \quad \text{where} \quad \Sigma_{set} = \frac{1}{|set|} \sum_{i \in set} (T_i - \mu_{set})(T_i - \mu_{set})^T + \lambda I_d$$

Figure 14 shows the heatmap of correlation and window detection on the test images for the crosswalk sign, the faces, and the stop sign using a lambda of 0.0001.

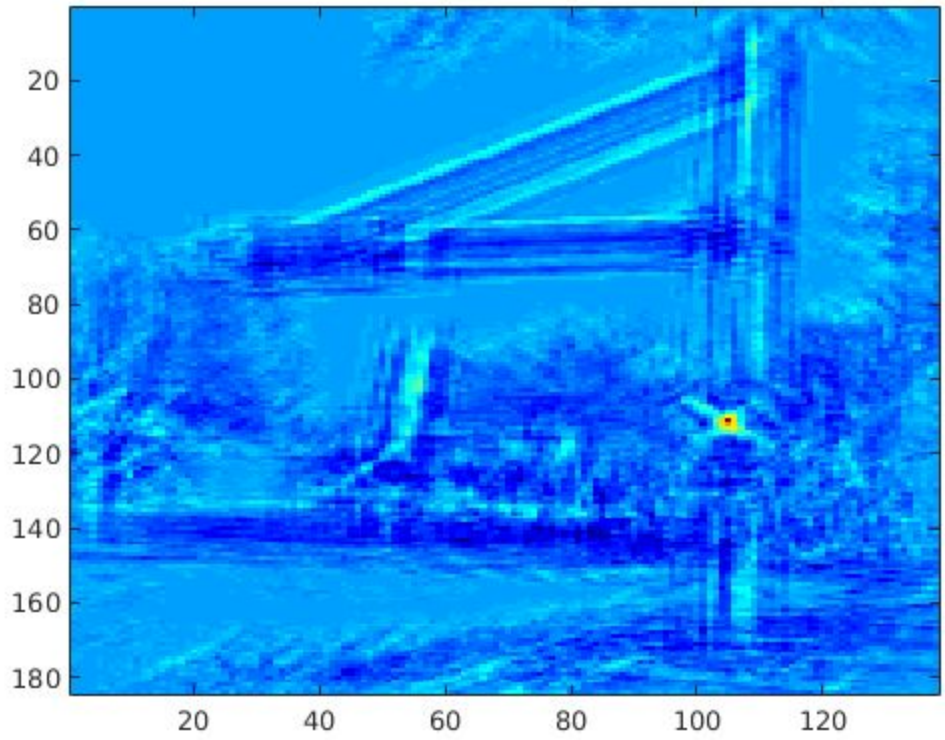

Figure 14a) Heatmap of Correlation for `test3.jpg` using LDA Template Learning

Figure 14b) Detection for `test3.jpg` using LDA Template Learning

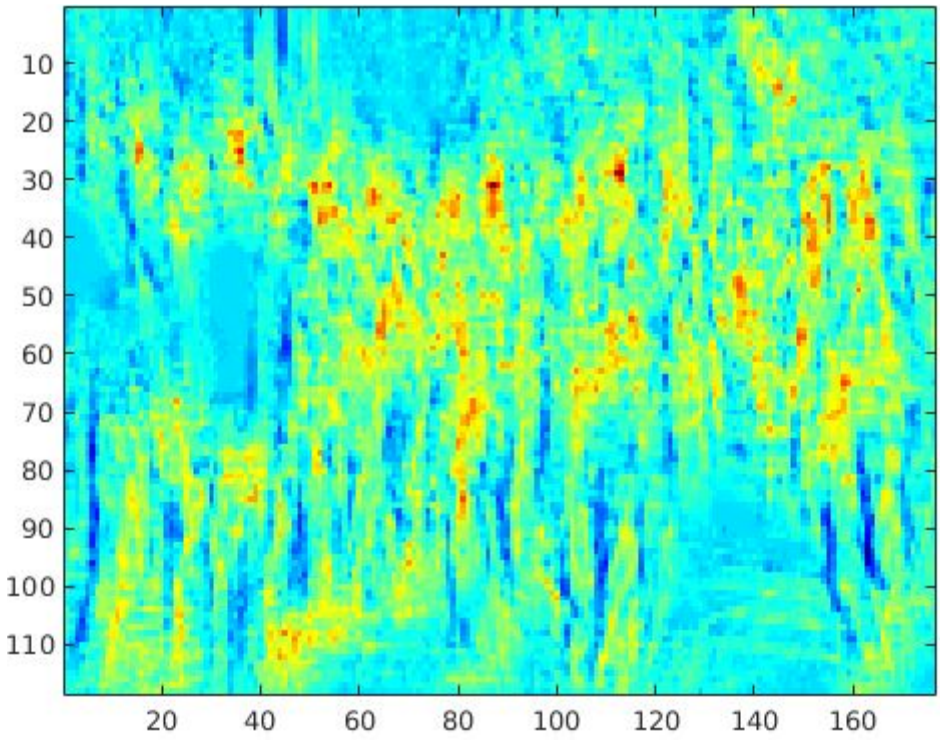

Figure 14c) Heatmap of Correlation for `test0.jpg` using LDA Template Learning

Figure 14d) Detection for `test0.jpg` using LDA Template Learning

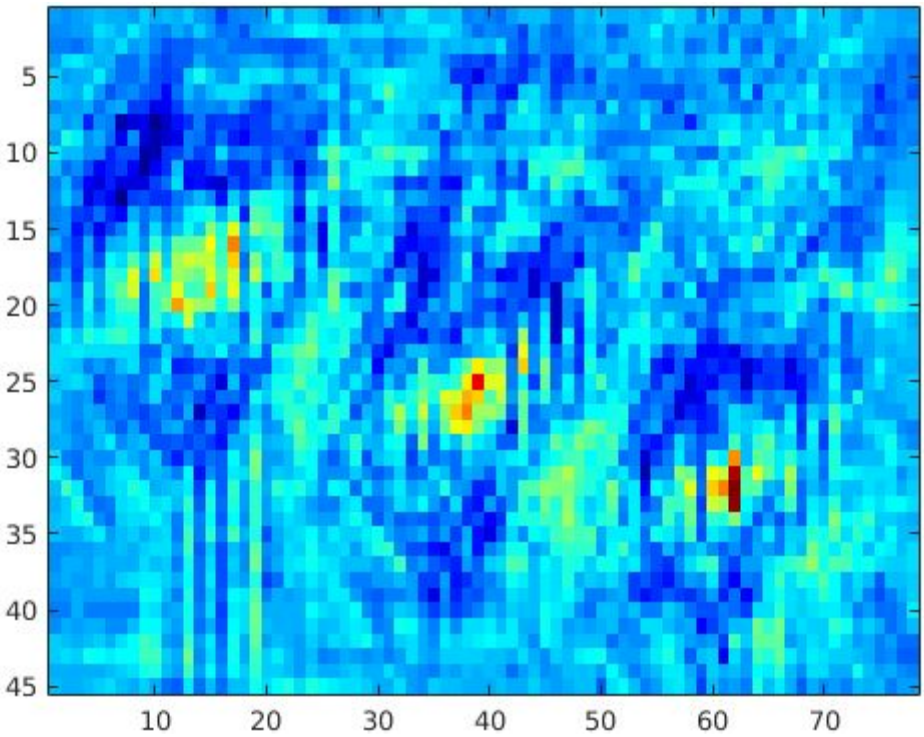

Figure 14e) Correlation Heatmap for `stop_sign_test2.jpg` using LDA Template Learning

Figure 14f) Top Five Detections for `stop_sign_test2.jpg` using LDA Template Learning

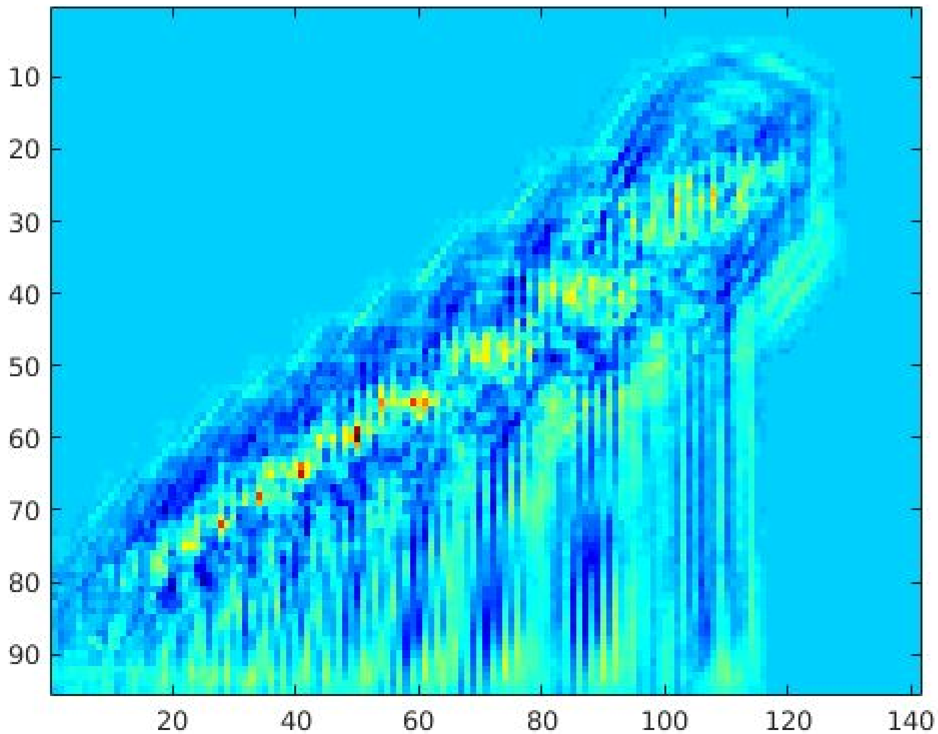

Figure 14g) Correlation Heatmap for `stop_sign_test3.jpg` LDA Template Learning

Figure 14h) Top Five Detections for `stop_sign_test3.jpg` using LDA Template Learning

2.5: Multi-Scale Detection

I implemented this functionality in `multiscale_detect.m`. For this code, I used the templates gathered from Section 2. I used the LDA method from 2.2 to generate my template. I then gathered `ndet` number of the highest detections using maximum suppression for each image size.

I then took the `ndet` most confident ones based on score and then returned those answers.

Figure 15 shows multi-scale detection on the street sign data set:

Figure 15) Multi-Scale Detection for `test3.jpg`

I also ran the multi-scale detection algorithm on faces with two pieces of test data as shown in Figure 16 and 17:

Figure 16) Multi-Scale Detection of Faces

Figure 17) Multi-Scale Detection of Faces

The multi-scale detection of faces is not as effective for a number of reasons. The main reason is that the contours goten by the sign are much clearer than the contours gathered by the faces. Another problem is that the faces are not always facing forward. This will cause it to be different than the template.

Figure 18 shows the training examples used on the faces which can be found in the .mat file `template_images_face_pos.mat`:

Figure 18) Training Examples of the Faces

Figure 19 shows the Training Examples used on the signs in .mat file `template_images_pos.mat`

Figure 19) Training Examples of Signs

I also used multi-scale detection on the stop signs with the top five detections shown on the test image in Figure 20:

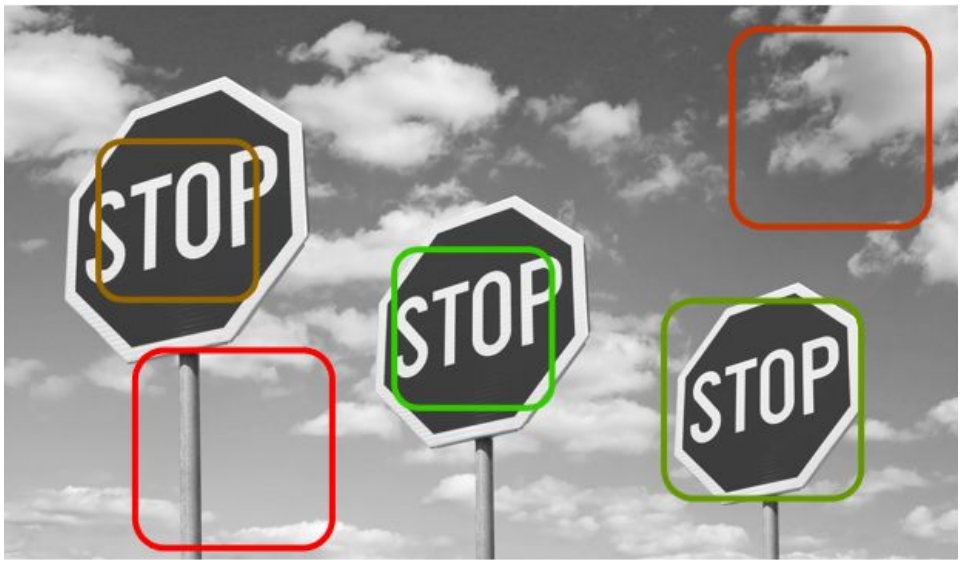

Figure 20a) Multi-Scale Detection of Stop Signs for `stop_sign_test2.jpg`

Figure 20b) Multi-Scale Detection of Stop Signs for `stop_sign_test3.jpg`

2.6: Extra Credit - Mixture of Templates

Here I used two different templates. The first template used faces that were forward facing and the second template used faces that were facing down or sideways.

Figure 21 shows the training examples of side faces which is found in `template_images_side_face_pos.mat`:

Figure 21) Training Image for Side Faces

Figure 22 shows the results for multi-template learning:

Figure 22) Top Five Results using Multi-Templating

Here are the top five results for the multi-template learning. Comparing this to figure 14 in section 2.5, using multi-templating got rid of the false positive next to the feet. This has given a result of the top five detections being all faces with no false positives.

The reason for this is that by using a multiple template method, you can create multiple templates that relate to the different orientations that a face can take in the scenes.

It also helps when the average has only one orientation. This makes each of the templates “cleaner” whenever you average them. Then each of the templates are stronger.

Another example to illustrate this can be shown from the test image using multi-scale single template below in Figure 23:

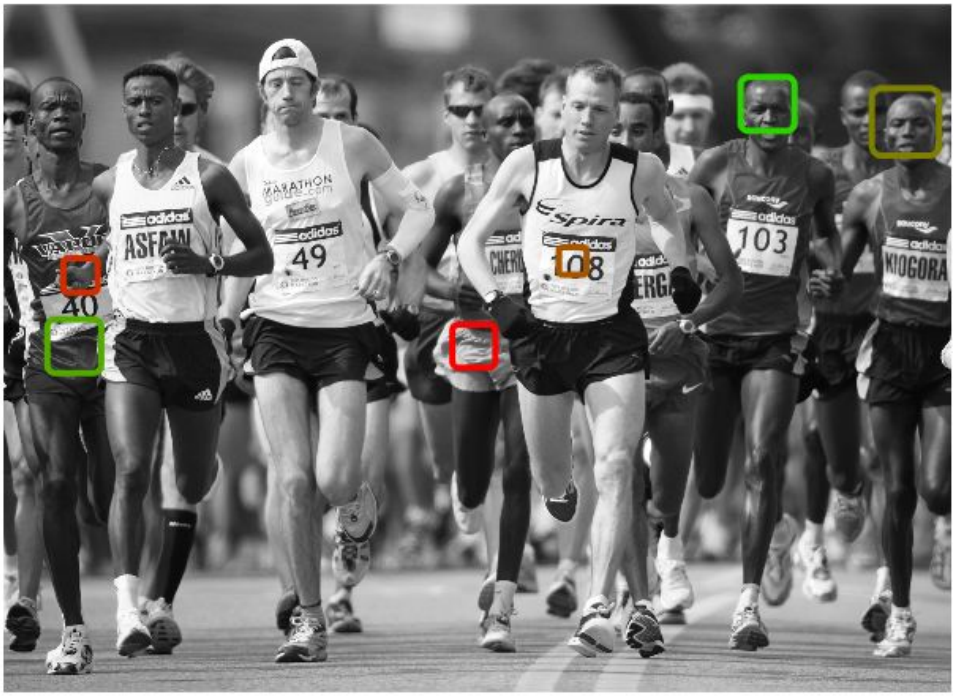

Figure 23) Detection using Multi-Scale Single Template

Figure 24 shows an example after using multi-templating:

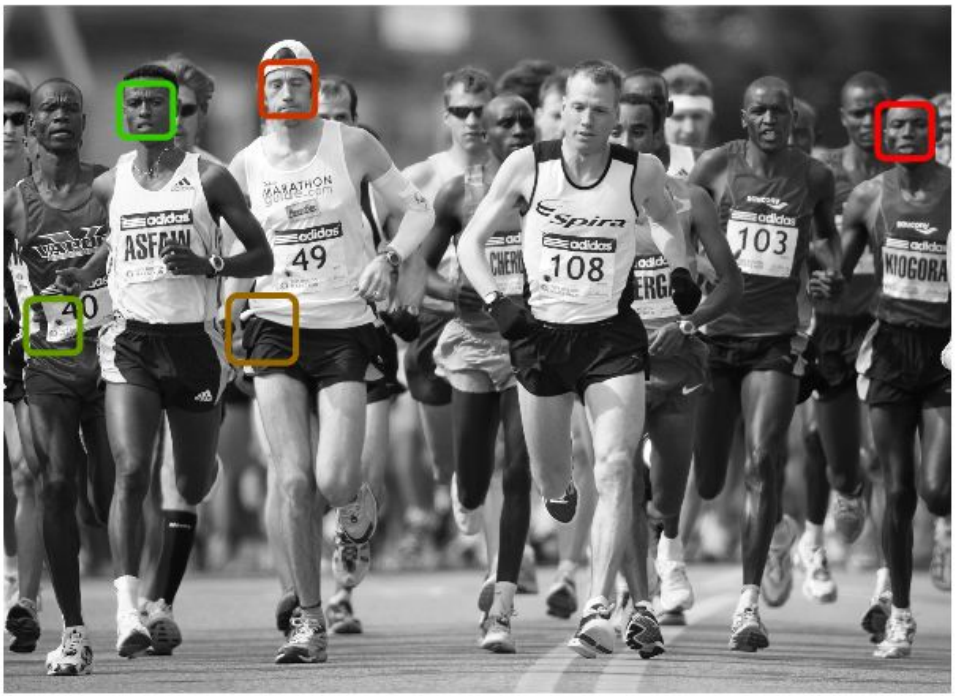

Figure 24) Detection Using Multi-Scale Multiple Templating

As can be seen, using multi-templating generated more correct detections and fewer false positives for the reasons given above for the first image.

Resources

[1] Discriminative Decorrelation for Clustering and Classification