Enrollment

To compute features, the SIFT algorithm was used. First, the sample images are read from the character folder through the sub-function `image_loader()`, and are given a `BINARY_INVERSE` threshold to make the character white (255) and the background black (0). This was used to match the thresholding in the Detection section. Next, `sift_function()` is used to perform the SIFT algorithm on the sample characters. Keypoints and descriptors are extracted, and are consolidated for all characters in the `kp_sample` and `des_sample` lists. The descriptor list is used in Recognition to compare test image characters to the sample ones.

Detection

The detection code makes use of Connected Components Labelling (CCL) to identify characters. This section is not complete, but the description of what was intended to be done, as well as the errors that occurred will be explained.

The test image is first thresholded as `BINARY_INVERSE` to allow the characters to be white and the background to be black, as would be neeed for CCL. This threshold also makes the pixel intensities binary (either 0 or 255). The `first_pass()` function is used to attach a label to all of the characters. It loops over each row and column of the image matrix, and uses several if statements to label each pixel. The labels are recorded in the `img_label` array, which is the same size as the test image array. Three major scenarios are considered for labelling – first column (no left neighbor), first column (no top neighbor), and middle pixels. Labels are chosen based on CCL algorithm. When both the top and left neighbors are white pixels, a parent-child situation is created. In this case, the `union` array is used to record a parent-child tuple. The `second_pass()` function is used to merge all children labels with parent labels, using `np.where()` to search and replace all instances of the child label to the parent label, making use of the aforementioned `union` array.

The `character_detection()` function was supposed to be used to use to extract each character from the `img_label` array, by forming a 'box' where the vertices are at the first instance of a given label and the last instance of that label `(min(row),min(column))`. The locations were found using `np.where()` which gives the locations of all instances of a given label. However, when these boxes were plotted using `cv2.rectangle()`, the following was obtained:

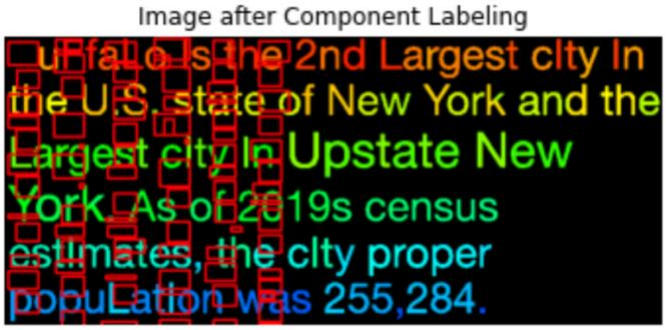

Figure 1: CCL Bounding Boxes Error

As can be seen, each character is a single solid color, implying that the labelling was successful, but the boxes are placed messily and do not surround each character.[1] The inability to select each character made it impossible to progress. So, to make up for this, `cv2.contours()` and `cv2.boudingRect()` were used to extract characters from the test image. The same SIFT features were applied to this by splicing the pixels of each box.

Recognition

For this section, the Brute Force matching with SIFT descriptors were, and ratio test was supposed to be used but I was unable to implement it. This essentially used `cv2.BFMatcher` to perform brute force matching using SIFT descriptors, and used `bf.knnMatch()`, with a ratio test of 0.75n to choose the two closest distance matches.

[1] This image may be seen by uncommenting the code for the colorize_ccl function, but it has been commented since it was obtained from online. It is merely a visual tool to view the output of CCL