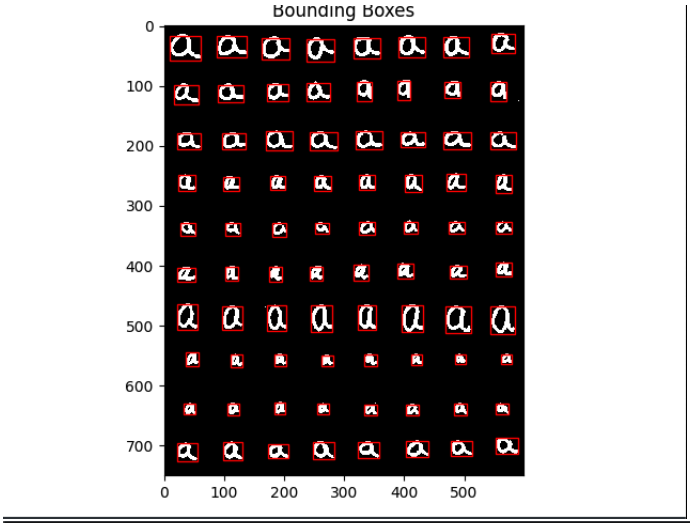
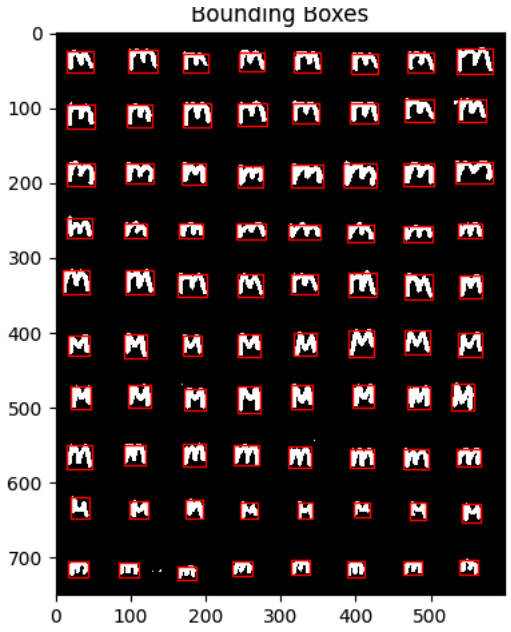
Imaging and Multimedia

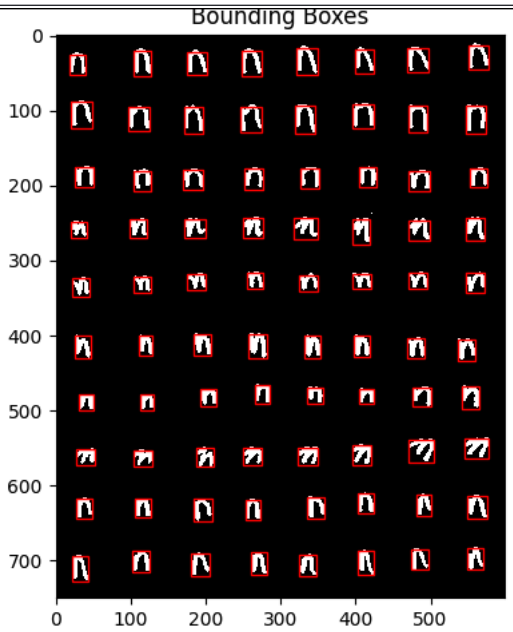
Joshua Redona

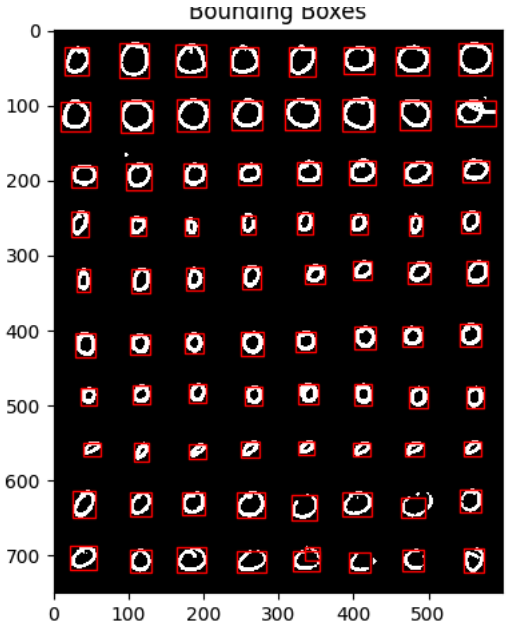
December 11, 2023

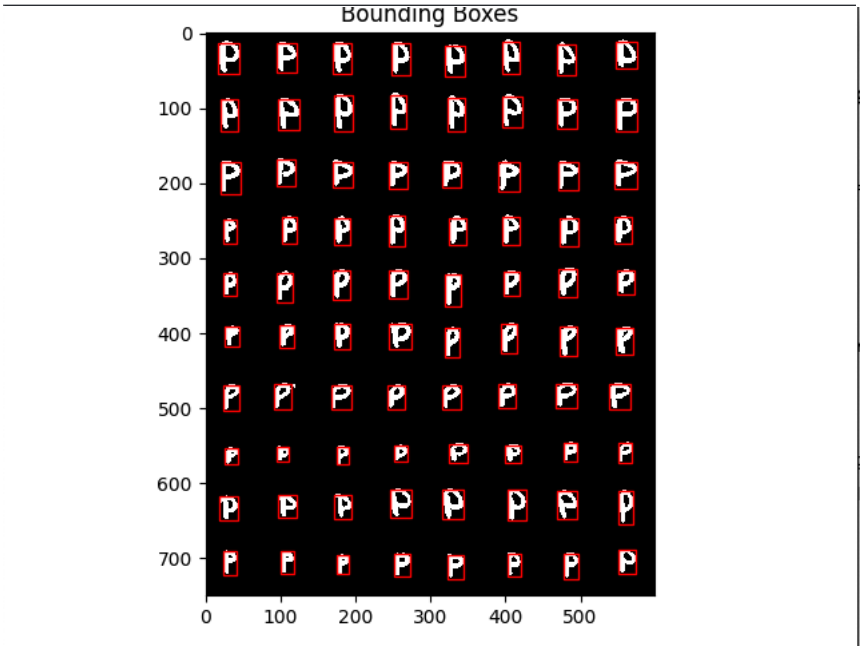
Training Images

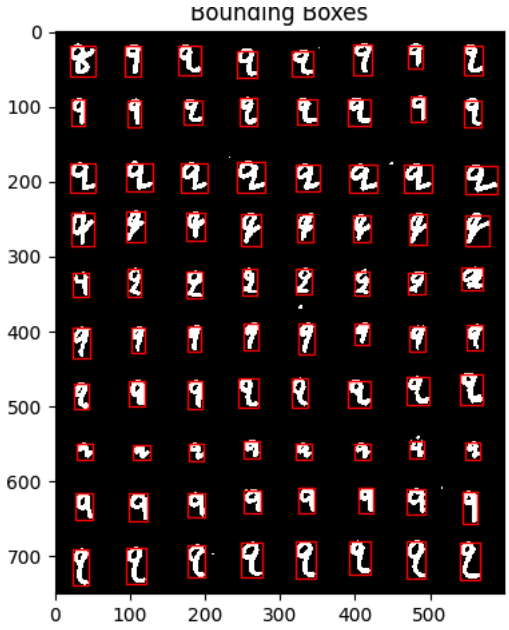


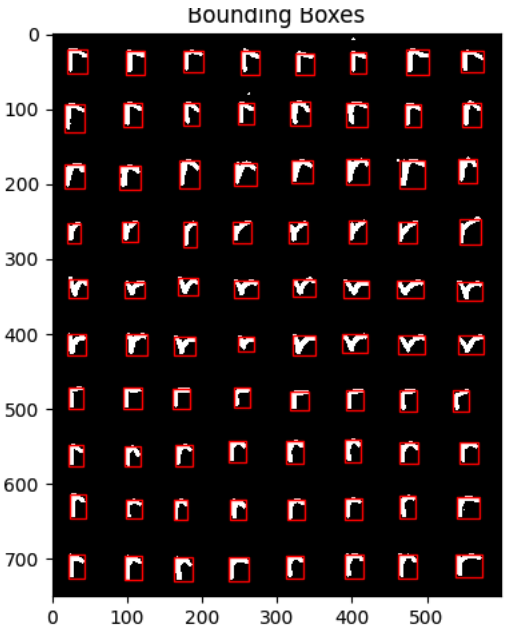

Bounding Boxes
0
100
200
300
400
500
600
700
0
100
200
300
400
500

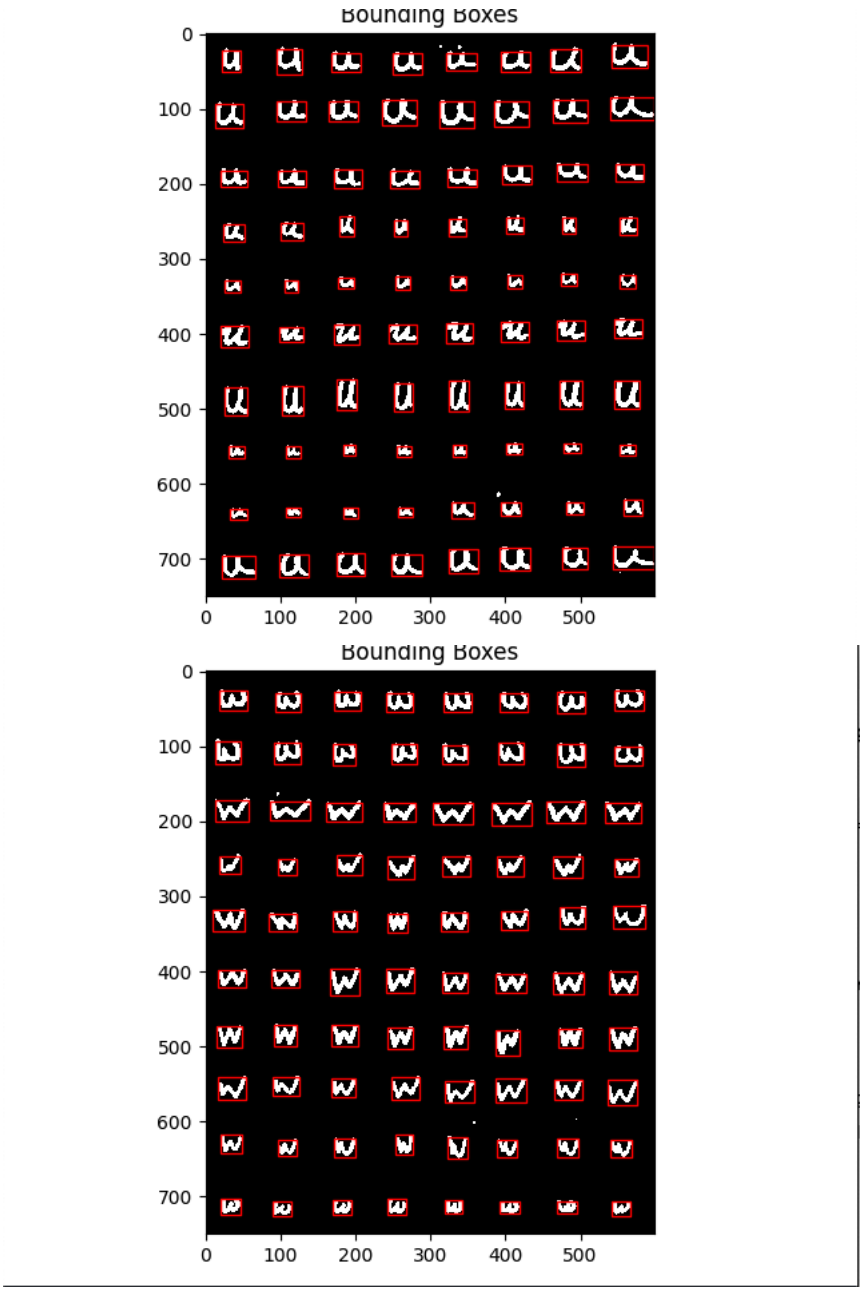

Bounding Boxes
0
100
200
300
400
500
600
700
0
100
200
300
400
500


Bounding Boxes
0
100
200
300
400
500
600
700
0
100
200
300
400
500

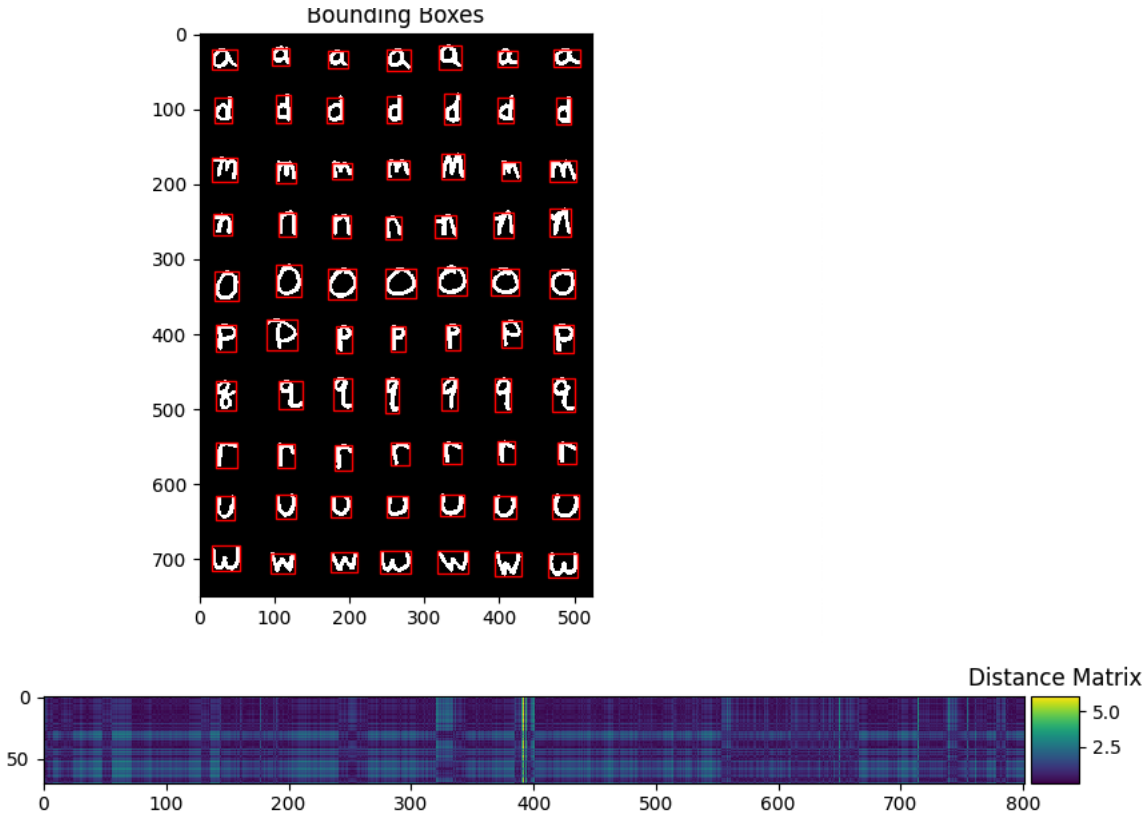

Bounding Boxes
0
100
200
300
400
500
600
700
0
100
200
300
400
500


Bounding Boxes
0
100
200
300
400
500
600
700
0
100
200
300
400
500


Bounding Boxes
0
100
200
300
400
500
600
700
0
100
200
300
400
500


Bounding Boxes
0
100
200
300
400
500
600
700
0
100
200
300
400
500
Bounding Boxes
0
100
200
300
400
500
600
700
0
100
200
300
400
500

Recognition Phase, No Enhancements



To start, I chose $th = 236$ to be my threshold, for no particular reason other than reducing the amount of connected components visually when drawing the bounding boxes.

Following the assignment instructions exactly, I ended up with an accuracy of 0.405 for the training data and 0.371 for the test data. For 10 characters, it's noticably better than random guessing, however without any enhancements to the data set or the method of prediction, it can be improved.

The problems that stick out to me come from the confusion matrix. First, I notice that a good number of 'a's are recognized as 'd', and many of the 'd's are recognized as 'p' or 'q'. This is the first issue I wanted to tackle.

Enhancememnts

The first small improvement I made was the classifier. Specifically, I chose to use the k-nearest neighbors approach that was described in the doc, with some success with $k = 3$. Additionally, I noticed that the 'o' character training image was much less cohesive than the rest, so I singled it out and applied binary closing to it. The

result was a negligible bump to the training data accuracy to 0.400, and no change with the test data accuracy.

To address the issue with 'a's and 'd's, I noticed that occassionally, the region being used to extract features would clip the top of the 'a', and I thought this would maybe make more 'a's be recognized as 'd's. To counteract this, I simply added a few more pixels above and to the right to each region. The result is again, a negligible bump in accuracy to the training data to 0.414 and no change in the training data accuracy. Surprisingly, this had no effect on how many 'a's were recognized as 'd's.

Following this up, I attempted to single out both the 'a' and 'd' training images for binary morphology. This resulted in an increase in the training accuracy to 0.421, but a significant decrease in test data accuracy to 0.357. The model no longer tries to predict 'd' as one of the characters being read in the test image, so I can only assume this modification made it more difficult for any recognition of 'd's. Of course, since this brings the accuracy of the training data down, I scrapped this bit of code.