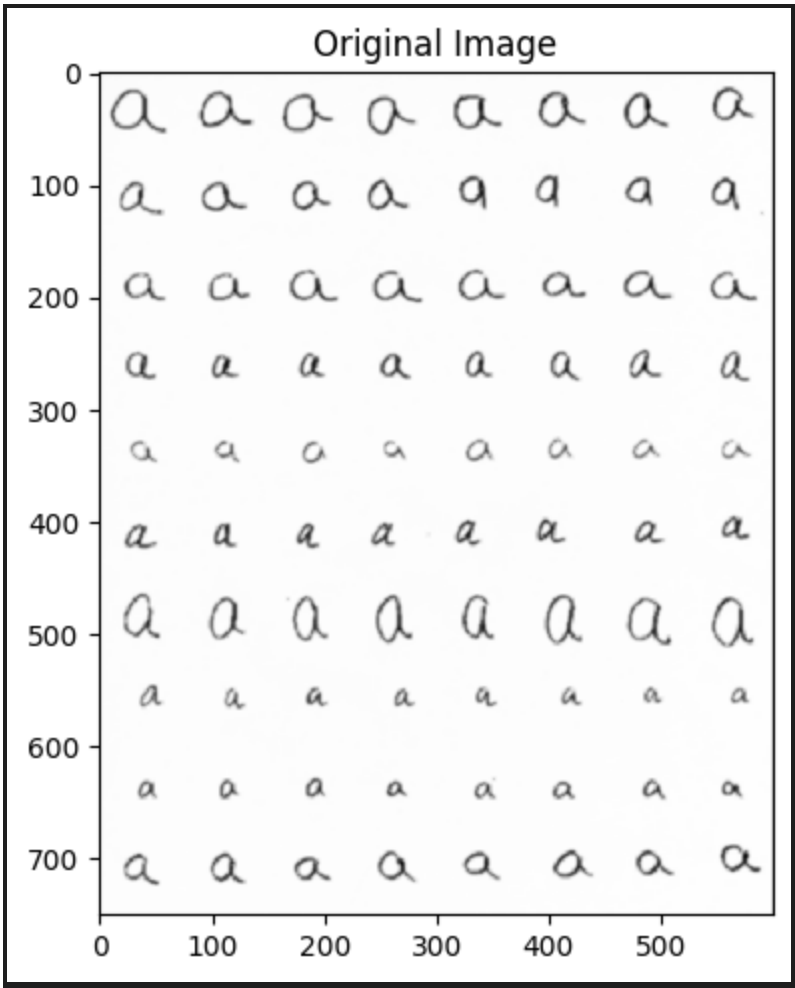
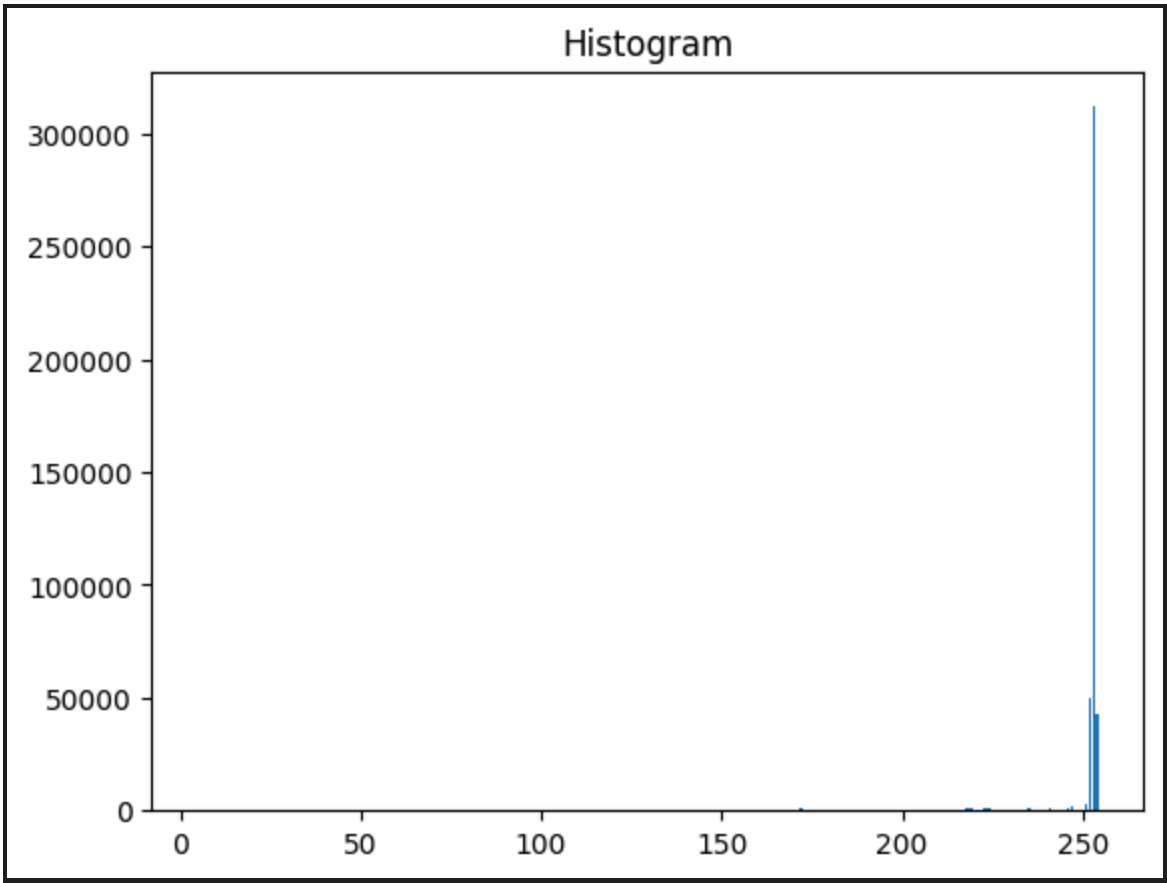
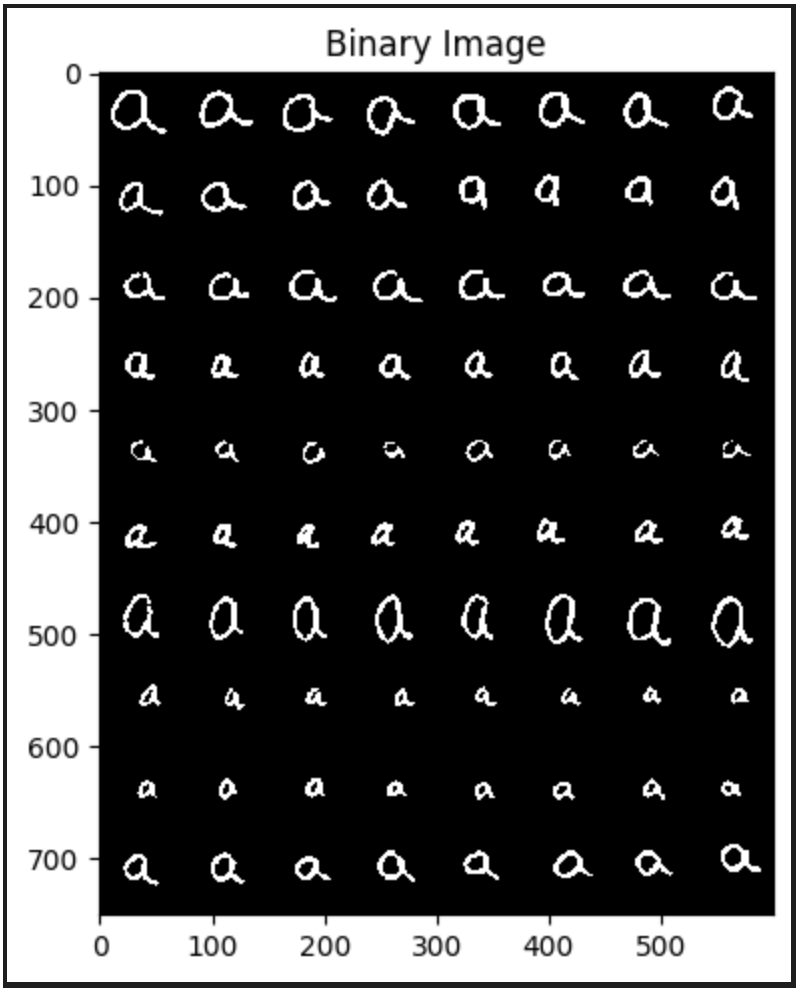
```
import numpy as np
from sklearn.metrics import confusion_matrix
from scipy.spatial.distance import cdist
from skimage.measure import label, regionprops, moments, moments_central,
moments_normalized, moments_hu
from skimage import io, exposure
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle
import pickle

import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.simplefilter(action='ignore', category=UserWarning)



#Enhancments
from sklearn.svm import SVC
from skimage.feature import hog
from sklearn.neighbors import KNeighborsClassifier
from skimage.morphology import closing, square
from skimage.filters import threshold_otsu
from skimage.filters import gaussian, median
from skimage.transform import resize
# Reading File and Shape
currentImage = io.imread('/a.bmp');
print(currentImage.shape)
#Visualize Image
io.imshow(currentImage)
plt.title('Original Image')
io.show()
```



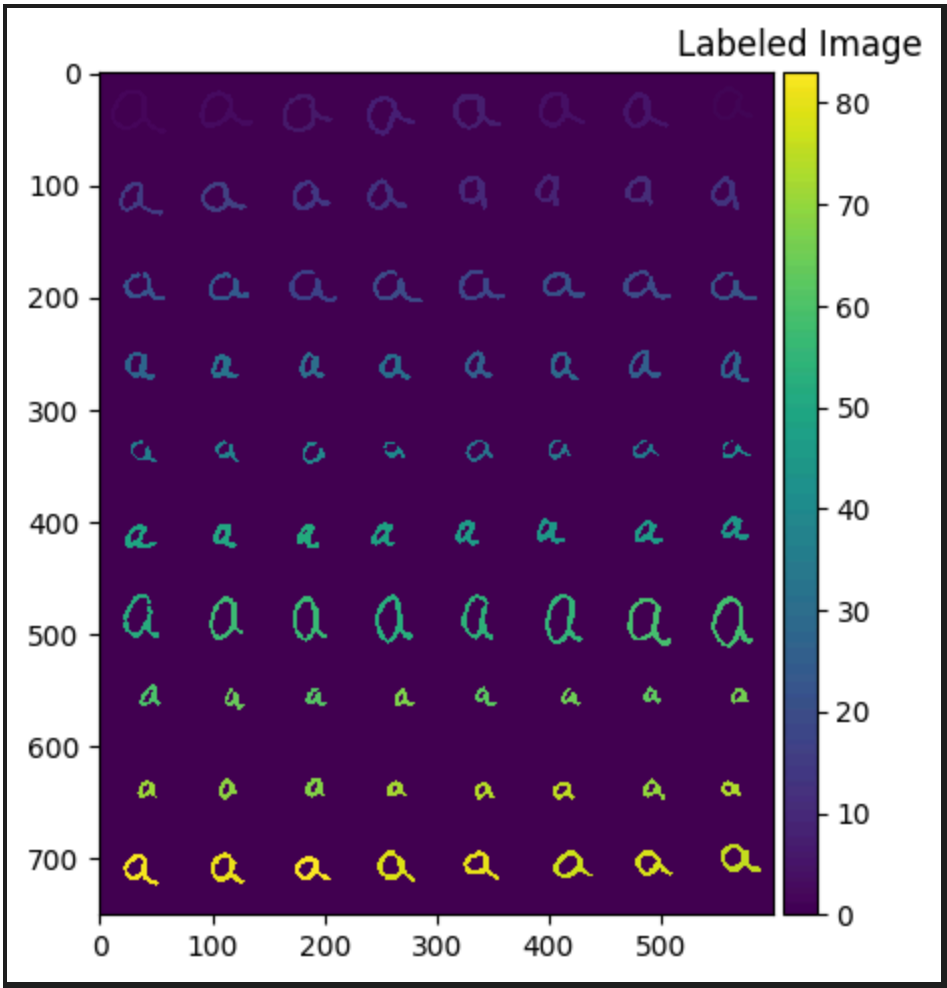

```
#Image Histogram

hist = exposure.histogram(currentImage)
plt.bar(hist[1],hist[0])
plt.title('Histogram')
plt.show()
```




```
#Binarization by Thresholding

th = 200
img_binary = (currentImage < th).astype(np.double)
io.imshow(img_binary)
plt.title('Binary Image')
io.show()
```

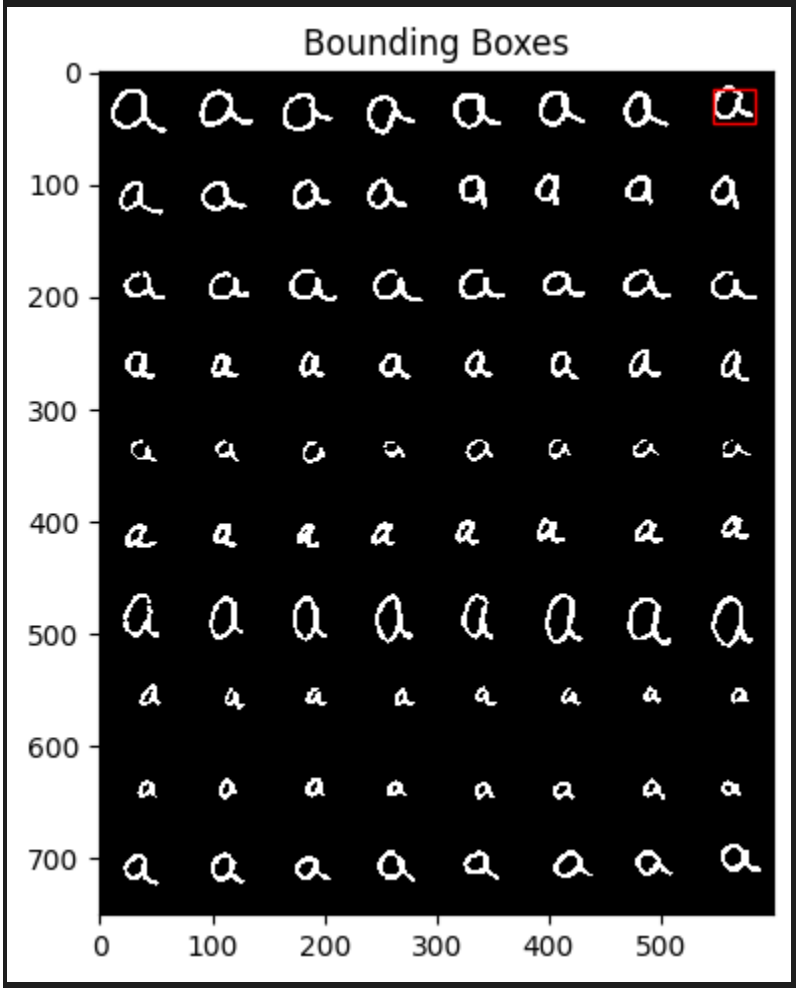



```
img_label = label(img_binary,background = 0)
io.imshow(img_label)
plt.title('Labeled Image')
io.show()
print(np.amax(img_label))
```




```
regions = regionprops(img_label)
io.imshow(img_binary)
ax=plt.gca()

Features = []
for props in regions:
  minr,minc,maxr,maxc = props.bbox
  if (maxr-minr) <= 10 or (maxc-minc) <= 10:
    continue
  roi = img_binary[minr:maxr,minc:maxc]
  m = moments(roi)
  cc = m[0,1]/m[0,0]
  cr = m[1,0]/m[0,0]
  mu=moments_central(roi,center=(cr,cc))
  nu=moments_normalized(mu)
  hu=moments_hu(nu)
  Features.append(hu)
```

```
  ax.add_patch(Rectangle((minc, minr), maxc - minc, maxr - minr,
fill=False, edgecolor='red', linewidth=1))
  ax.set_title('Bounding Boxes')
  io.show()
```




```
Images =
["/a.bmp","/d.bmp","/f.bmp","/h.bmp","/k.bmp","/m.bmp","/n.bmp","/o.bmp","
/p.bmp","/q.bmp","/r.bmp","/s.bmp","/u.bmp","/w.bmp","/x.bmp","/z.bmp",]

def BuildFeatureDatabase(Images, ShowImage):
  Features = []
  ClassLabel = []
  for Image in Images:
    curImg = io.imread(Image);
    #curImg = median(curImg, np.ones((2, 2)))
    #curImg = gaussian(curImg, sigma=1)
    #io.imshow(curImg)
    #plt.title('Original Image')
```

```
    #if ShowImage:
      #io.show()

    #thresholding
    #th = threshold_otsu(curImg)
    th = 195
    img_binary = (curImg < th).astype(np.double)
    #img_binary = closing(img_binary, square(1))
    #img_binary = median(curImg, np.ones((2, 2)))
    #if ShowImage:
      #io.imshow(img_binary)
    #plt.title('Binary Image')
    #if ShowImage:
      #io.show()
    #Labeling
    img_label = label(img_binary,background = 0)
    #if ShowImage:
      #io.imshow(img_label)
   # plt.title('Labeled Image')
    #if ShowImage:
      #io.show()

    # Bounding Boxes
    regions = regionprops(img_label)
    if ShowImage:
      io.imshow(img_binary)
    ax=plt.gca()

    for props in regions:
      minr,minc,maxr,maxc = props.bbox
      if (maxr-minr) >= 100 or (maxc-minc) >= 100 or (maxr-minr) <= 10 or
(maxc-minc) <= 10:
        continue
      roi = img_binary[minr:maxr,minc:maxc]
      m = moments(roi)
      cc = m[0,1]/m[0,0]
      cr = m[1,0]/m[0,0]
      mu=moments_central(roi,center=(cr,cc))
      nu=moments_normalized(mu)
      hu=moments_hu(nu)
```

```
      ax.add_patch(Rectangle((minc, minr), maxc - minc, maxr - minr,
fill=False, edgecolor='red', linewidth=1))
      Features.append(hu)
      ClassLabel.append(Image[1])
    if ShowImage:
      ax.set_title('Bounding Boxes')
    if ShowImage:
      io.show()

  NPFeatures = np.asarray(Features)
  #print(NPFeatures)
  FeatureMean = np.mean(NPFeatures)
  #print(FeatureMean)
  FeatureSTD = np.std(NPFeatures)
  #print(FeatureSTD)

  NormalizedFeatures = ((NPFeatures-FeatureMean) / FeatureSTD)
  #print(len(NormalizedFeatures))
  return NormalizedFeatures,ClassLabel,FeatureMean,FeatureSTD

def BuildHOGFeatureDatabase(Images, ShowImage):
    Features = []
    ClassLabel = []

    for Image in Images:
        curImg = io.imread(Image)

        curImg_resized = resize(curImg, (128, 128))

        io.imshow(curImg)
        plt.title('Original Image')
        if ShowImage:
            io.show()

        # Thresholding
        th = 195
        img_binary = (curImg_resized < th).astype(np.double)
        img_binary = closing(img_binary, square(1))
        if ShowImage:
            io.imshow(img_binary)
```

```
        plt.title('Binary Image')
        if ShowImage:
            io.show()

        # Labeling
        img_label = label(img_binary, background=0)
        if ShowImage:
            io.imshow(img_label)
        plt.title('Labeled Image')
        if ShowImage:
            io.show()

        # HOG Feature Extraction
        fd, hog_image = hog(curImg_resized, orientations=9,
pixels_per_cell=(8, 8), cells_per_block=(2, 2), visualize=True)

        Features.append(fd)

        ClassLabel.append(Image[1])

        if ShowImage:
            io.imshow(hog_image)
            plt.title('HOG Image')
            io.show()

    NPFeatures = np.array(Features)
    FeatureMean = np.mean(NPFeatures, axis=0)
    FeatureSTD = np.std(NPFeatures, axis=0)
    NormalizedFeatures = (NPFeatures - FeatureMean) / FeatureSTD

    return NormalizedFeatures, ClassLabel, FeatureMean, FeatureSTD

def EvalTraining(FeatList,labels):
  labels = np.array(labels)
  D = cdist(FeatList,FeatList)
  io.imshow(D)
  plt.title('Distance Matrix')
  io.show()
  D_index = np.argsort(D,axis=1)
  #print(D_index)
```

```
  DMatch = D_index[:,1]
  #print(DMatch)
  matchingLabel = labels[DMatch]
  #print(matchingLabel)
  correctMatches = matchingLabel == labels
  accuracy = np.mean(correctMatches)
  #print(accuracy)
  #print(FeatList)
  #print(labels)
  #for i in range(10):
    #print(f"True Label: {labels[i]}, Predicted Label:
{matchingLabel[i]}")
  confM = confusion_matrix(labels,matchingLabel)
  io.imshow(confM)
  plt.title('Confusion Matrix')
  io.show()

def Recognition(TrainingFeats,FeatureMean,FeatureSTD,labels,ShowImage):
  TestingFeatures = []
  curImg = io.imread('/test1.bmp');
  #curImg = median(curImg, np.ones((2, 2)))

  #curImg = gaussian(curImg, sigma=1)

  if ShowImage:
    io.imshow(curImg)
    plt.title('Original Image')
  if ShowImage:
    io.show()
  th = 230
  #th = threshold_otsu(curImg)
  img_binary = (curImg < th).astype(np.double)
  #img_binary = closing(img_binary, square(1))
  #img_binary = median(curImg, np.ones((2, 2)))
  if ShowImage:
    io.imshow(img_binary)
    plt.title('Binary Image')
  if ShowImage:
    io.show()
  #Labeling
```

```
  img_label = label(img_binary,background = 0)
  if ShowImage:
    io.imshow(img_label)
    plt.title('Labeled Image')
  if ShowImage:
    io.show()

  # Bounding Boxes
  regions = regionprops(img_label)
  if ShowImage:
    io.imshow(img_binary)
  ax=plt.gca()
  for props in regions:
    minr,minc,maxr,maxc = props.bbox
    if (maxr-minr) <= 1 or (maxc-minc) <= 1:
      continue
    roi = img_binary[minr:maxr,minc:maxc]
    m = moments(roi)
    cc = m[0,1]/m[0,0]
    cr = m[1,0]/m[0,0]
    mu=moments_central(roi,center=(cr,cc))
    nu=moments_normalized(mu)
    hu=moments_hu(nu)
    ax.add_patch(Rectangle((minc, minr), maxc - minc, maxr - minr,
fill=False, edgecolor='red', linewidth=1))
    TestingFeatures.append(hu)
  if ShowImage:
    ax.set_title('Bounding Boxes')
  if ShowImage:
    io.show()
  #print(TestingFeatures)
  #print(FeatureMean,FeatureSTD)
  NPFeatures = np.asarray(TestingFeatures)
  NormalizedFeatures = ((NPFeatures-FeatureMean) / FeatureSTD)
  #print(NormalizedFeatures)

  labels = np.array(labels)
  D = cdist(NormalizedFeatures,TrainingFeats)
  if ShowImage:
    io.imshow(D)
```

```
    plt.title('Distance Matrix')
  if ShowImage:
    io.show()
  D_index = np.argsort(D,axis=1)
  #print(D_index)
  DMatch = D_index[:,0]
  #print(DMatch)
  Prediction = labels[DMatch]
  print(Prediction)
  return Prediction


def KNNRecognition(TrainingFeats, FeatureMean, FeatureSTD, labels,
ShowImage):
    TestingFeatures = []
    curImg = io.imread('/test1.bmp')
    if ShowImage:
        io.imshow(curImg)
        plt.title('Original Image')
    if ShowImage:
        io.show()
    th = 230
    img_binary = (curImg < th).astype(np.double)
    if ShowImage:
        io.imshow(img_binary)
        plt.title('Binary Image')
    if ShowImage:
        io.show()

    # Labeling
    img_label = label(img_binary, background=0)
    if ShowImage:
        io.imshow(img_label)
        plt.title('Labeled Image')
    if ShowImage:
        io.show()

    # Bounding Boxes
    regions = regionprops(img_label)
    if ShowImage:
```

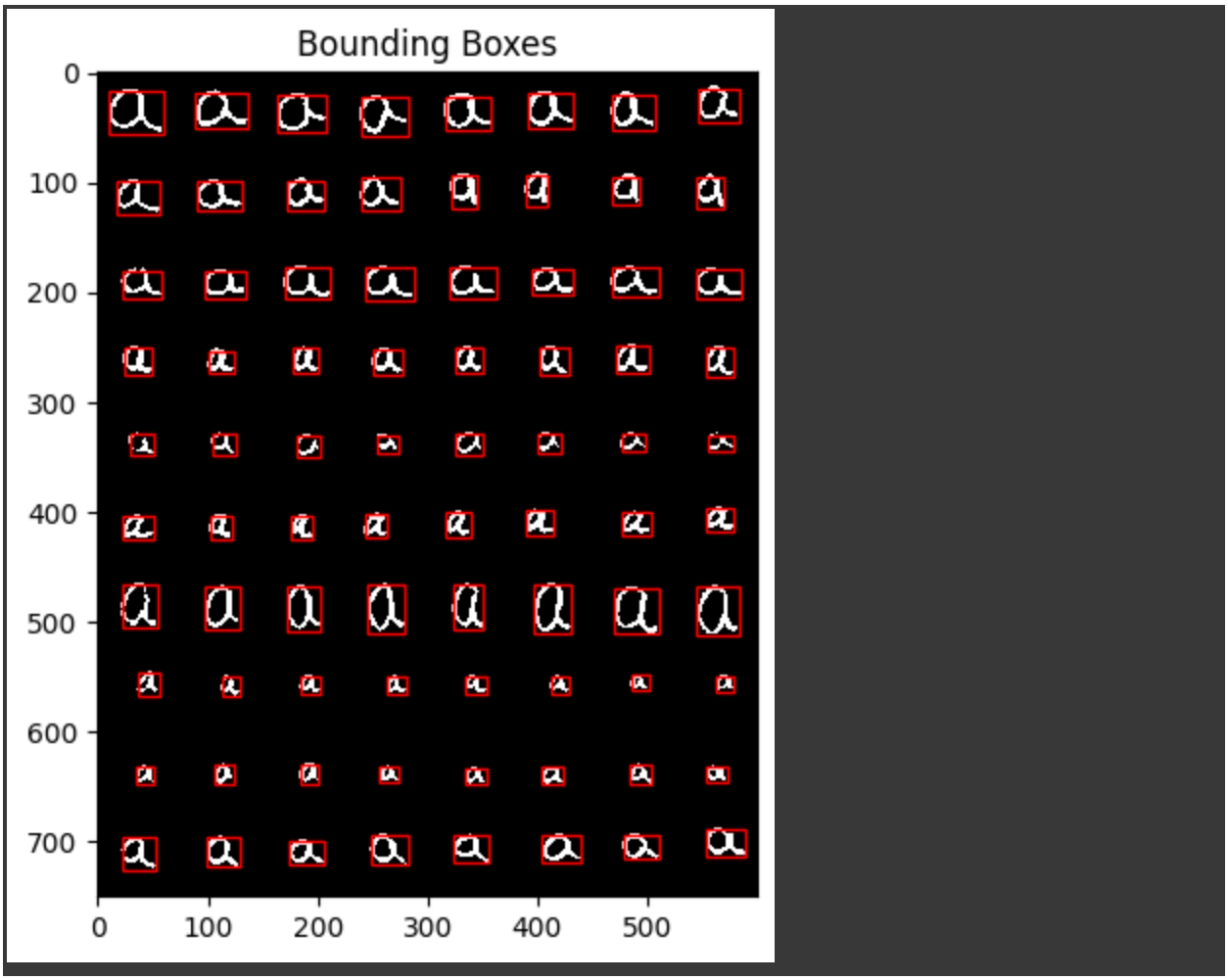
```
        io.imshow(img_binary)
    ax = plt.gca()
    print("Number of Test components" , np.amax(img_label))
    for props in regions:
        minr, minc, maxr, maxc = props.bbox
        if (maxr - minr) <= 1 or (maxc - minc) <= 1:
            continue
        roi = img_binary[minr:maxr, minc:maxc]
        m = moments(roi)
        cc = m[0, 1] / m[0, 0]
        cr = m[1, 0] / m[0, 0]
        mu = moments_central(roi, center=(cr, cc))
        nu = moments_normalized(mu)
        hu = moments_hu(nu)
        ax.add_patch(Rectangle((minc, minr), maxc - minc, maxr - minr,
fill=False, edgecolor='red', linewidth=1))
        TestingFeatures.append(hu)
    if ShowImage:
        ax.set_title('Bounding for Test Image')
    if ShowImage:
        io.show()

    NPFeatures = np.asarray(TestingFeatures)
    NormalizedFeatures = (NPFeatures - FeatureMean) / FeatureSTD

    # Train and predict with k-NN
    labels = np.array(labels)
    knn = KNeighborsClassifier(n_neighbors=3, metric='manhattan')  # Using
k=3
    knn.fit(TrainingFeats, labels)
    KNNprediction = knn.predict(NormalizedFeatures)

    #print("k-NN Predictions: ", KNNprediction)
    return KNNprediction



from skimage.feature import hog
from skimage.transform import resize
import numpy as np
```

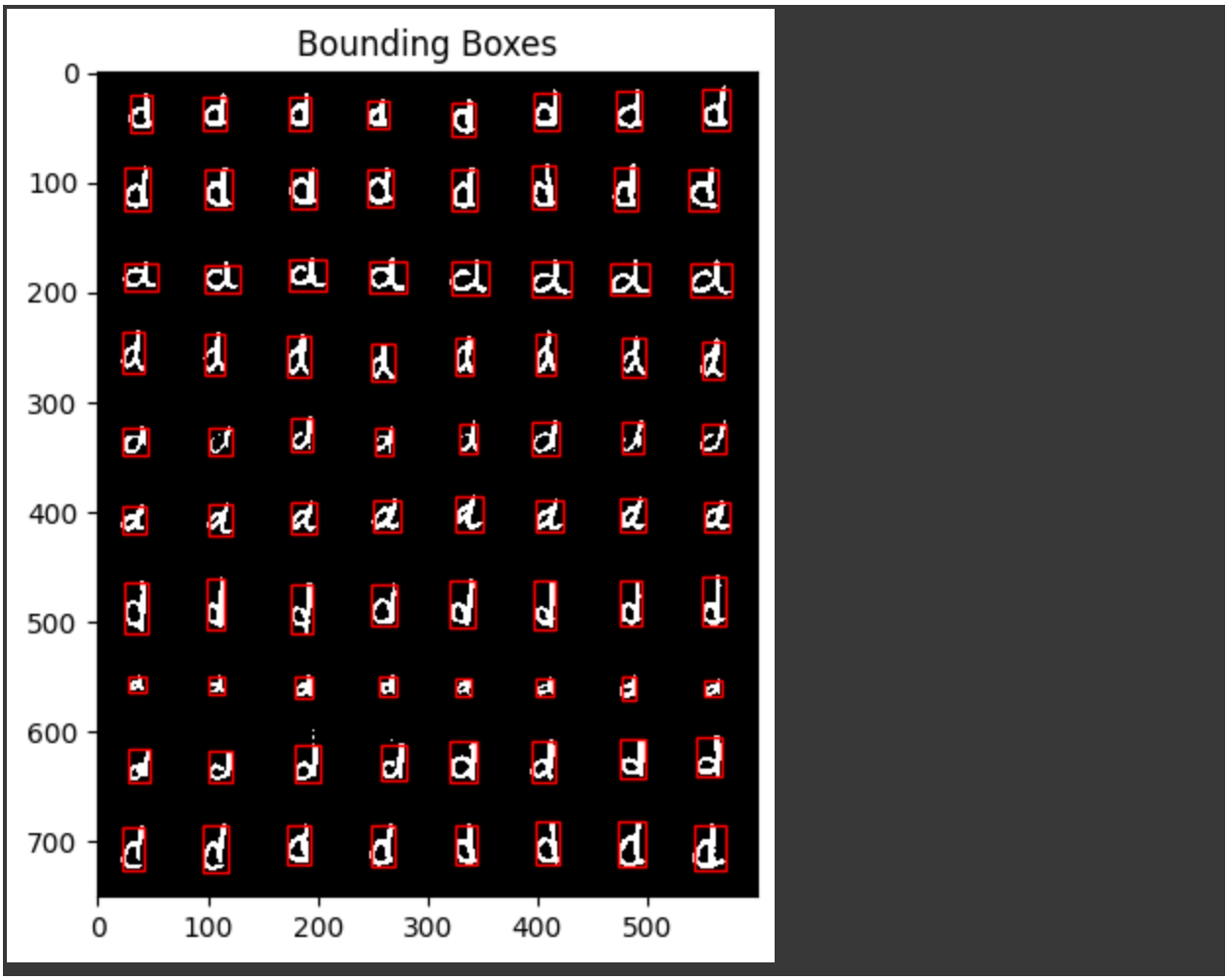
```
def KNNRecognitionHOG(TrainingFeats, FeatureMean, FeatureSTD, labels,
ShowImage):
    TestingFeatures = []
    curImg = io.imread('/test1.bmp')

    if ShowImage:
        io.imshow(curImg)
        plt.title('Original Image')
        io.show()

    curImg_resized = resize(curImg, (128, 128))

    th = 230
    img_binary = (curImg_resized < th).astype(np.double)

    # Labeling and feature extraction
    img_label = label(img_binary, background=0)
    if ShowImage:
        io.imshow(img_label)
        plt.title('Labeled Image')
        io.show()

    # HOG Feature Extraction
    fd, hog_image = hog(curImg_resized, orientations=9,
pixels_per_cell=(8, 8), cells_per_block=(2, 2), visualize=True)
    TestingFeatures.append(fd)

    if ShowImage:
        io.imshow(hog_image)
        plt.title('HOG Image')
        io.show()

    NPFeatures = np.asarray(TestingFeatures)

    NormalizedFeatures = (NPFeatures - FeatureMean) / FeatureSTD

    # Train and predict with k-NN
    labels = np.array(labels)
    knn = KNeighborsClassifier(n_neighbors=3)  # Using k=3
```

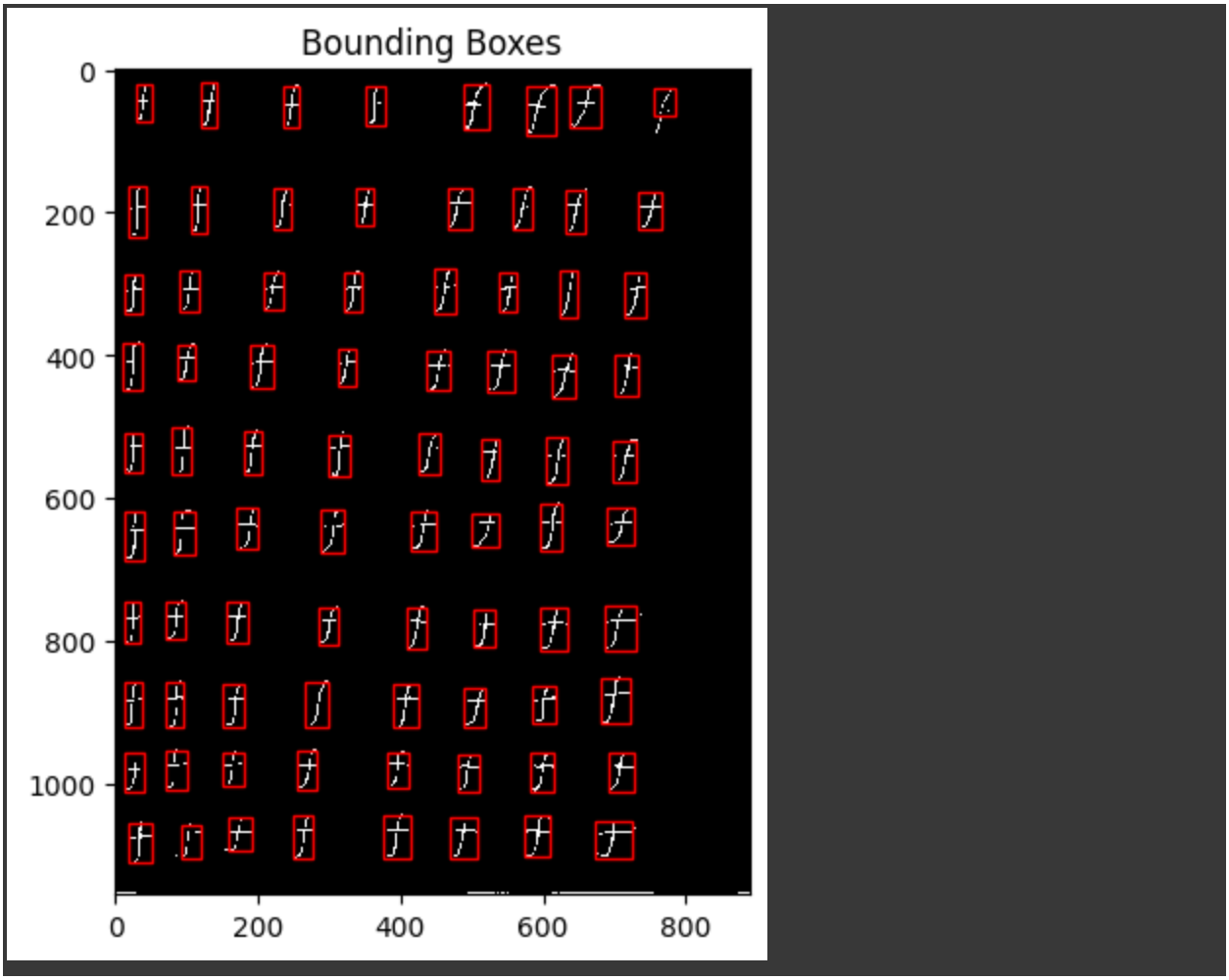
```
    knn.fit(TrainingFeats, labels)
    KNNprediction = knn.predict(NormalizedFeatures)

    #print("k-NN Predictions: ", KNNprediction)
    return KNNprediction



def GroundTruth(Prediction):
  pkl_file = open('/test_gt_py3.pkl', 'rb')
  mydict = pickle.load(pkl_file)

  #DECODE BYTE STRING INTO REGULAR STRING BECAUSE ITS WRONG
  mydict = {key.decode(): value for key, value in mydict.items()}
  #print(mydict.keys())
  pkl_file.close()
  classes = mydict['classes']
  locations = mydict['locations']

  PredictionLabels = np.array(Prediction)
  #print(PredictionLabels)
  GroundTruthLabels = np.array(classes)
  #print(GroundTruthLabels)

  accuracy = np.mean(GroundTruthLabels == PredictionLabels)
  #for i in range(len(PredictionLabels)):
    #print(f"True: {GroundTruthLabels[i]}, Predicted:
{PredictionLabels[i]}")
  print("TESTING ACCURACY: ", "{:.5f}".format(accuracy))



FeatList,labels,MEAN,STD = BuildFeatureDatabase(Images,ShowImage=True)
EvalTraining(FeatList,labels)
P = KNNRecognition(FeatList,MEAN,STD,labels,ShowImage=True)
GroundTruth(P)
```

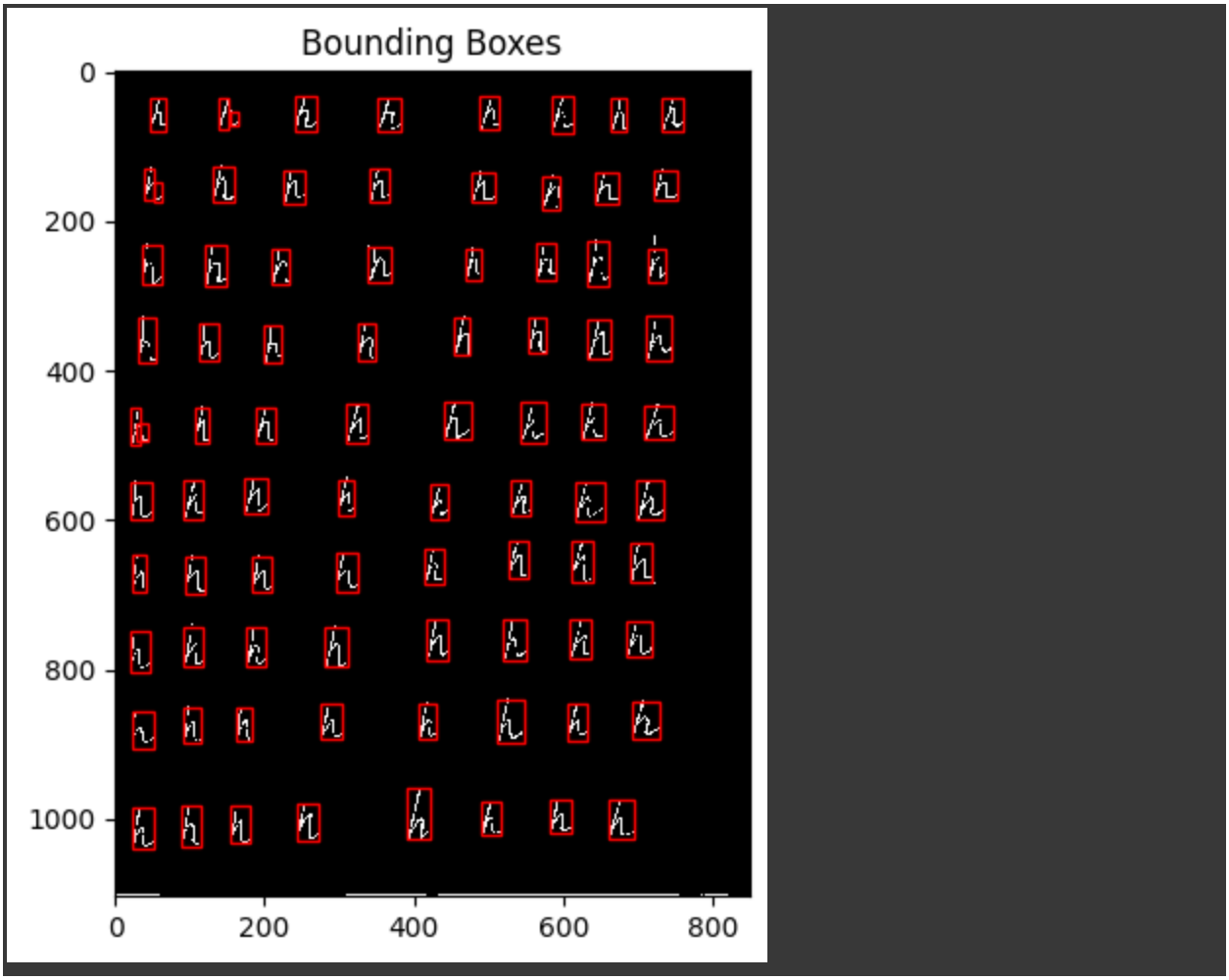
```
Reported Values:

```

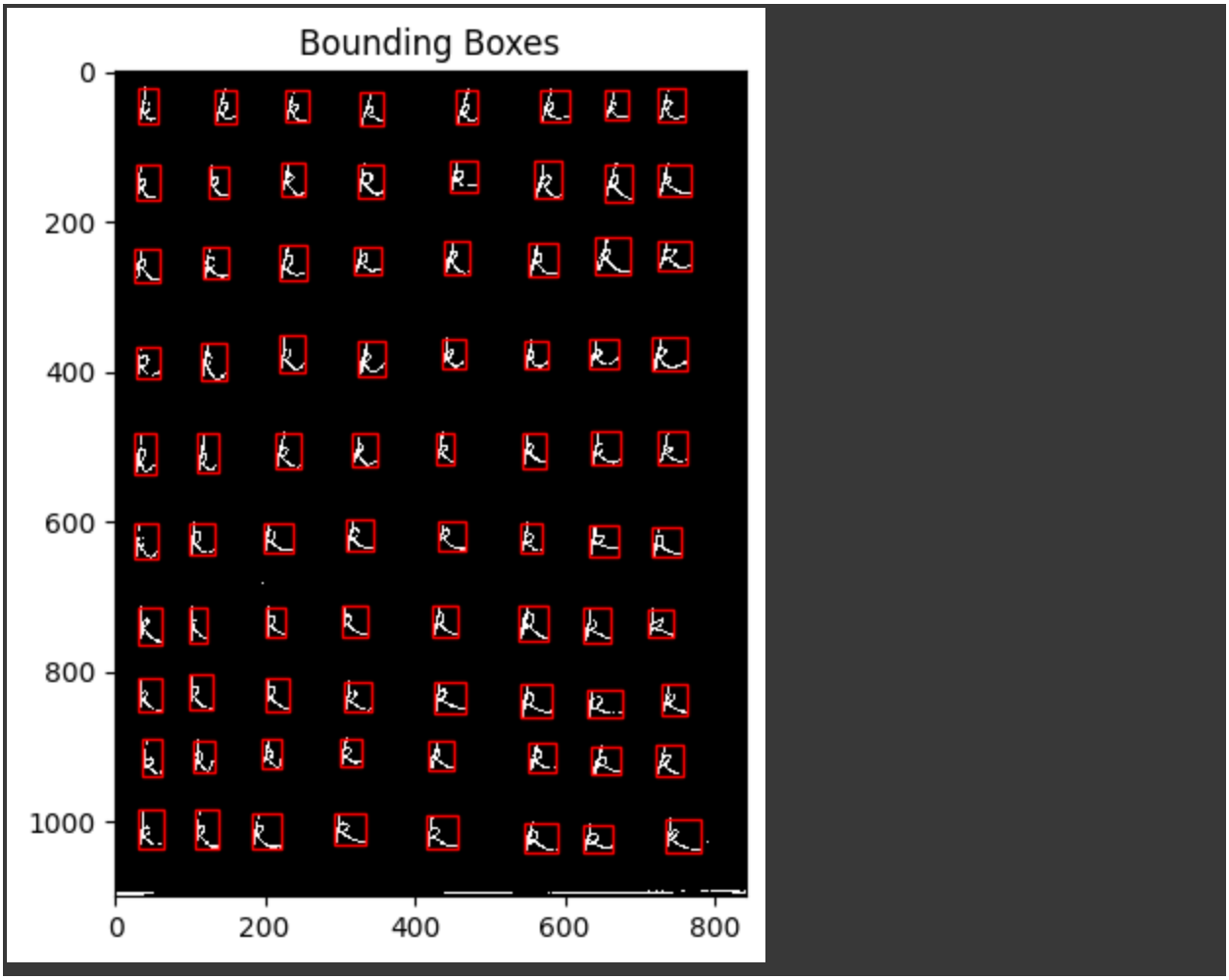
```
*   Threshold Value: Training 195, Testing 230
*   Number of Components for Test Image: 71
*   Original Recognition Rate for test Image: 0.27143
*   Improvement 1: Refined Thresholds (Originally both 200) => 0.40000
*   Improvement 2: Adaptive Tresholds => DOWN to 0.30000
*   Improvement 3: Closing Training Images => No Improvements
*   Improvement 4: Guassian Blurring before binarization => DOWN TO
0.24286
*   Improvement 5: Median Filter => No improvements
*   Improvement 6: 3-nearest neighbors w/ manhattan distance Classifier =>
same at 0.40000
*   Improvement 7: Changing accepted noise size => No Improvements
*   Improvement 8: Using HOG (Histogram of Oriented Gradients) Features
with skimage => DOWN TO 0.10000

Overall, 0.40000 is really the best the character recognizer is going to
get. All attempts at reasonable improvement yielded either negative or
neutral results.
```

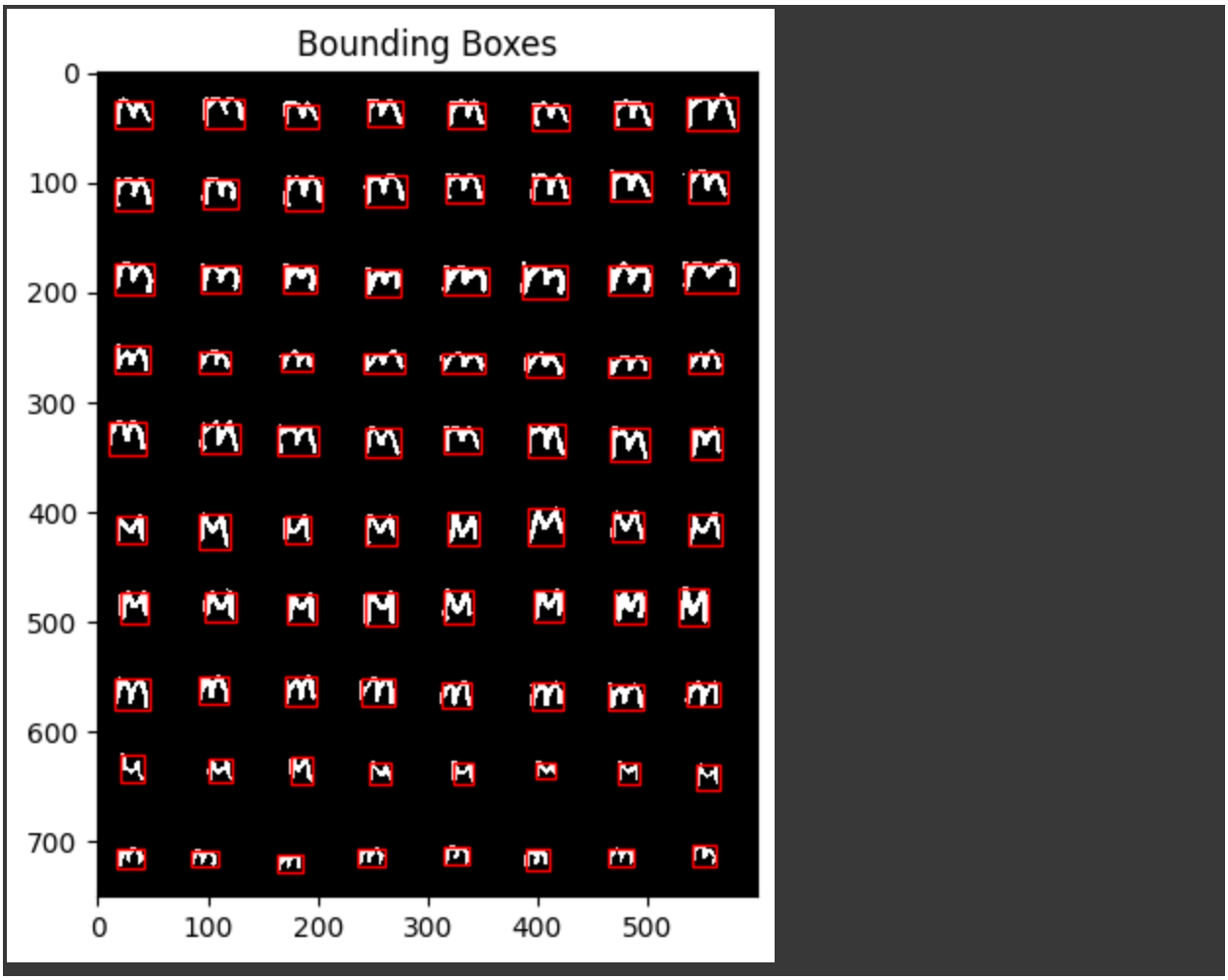

Bounding Boxes
0
100
200
300
400
500
600
700
0
100
200
300
400
500

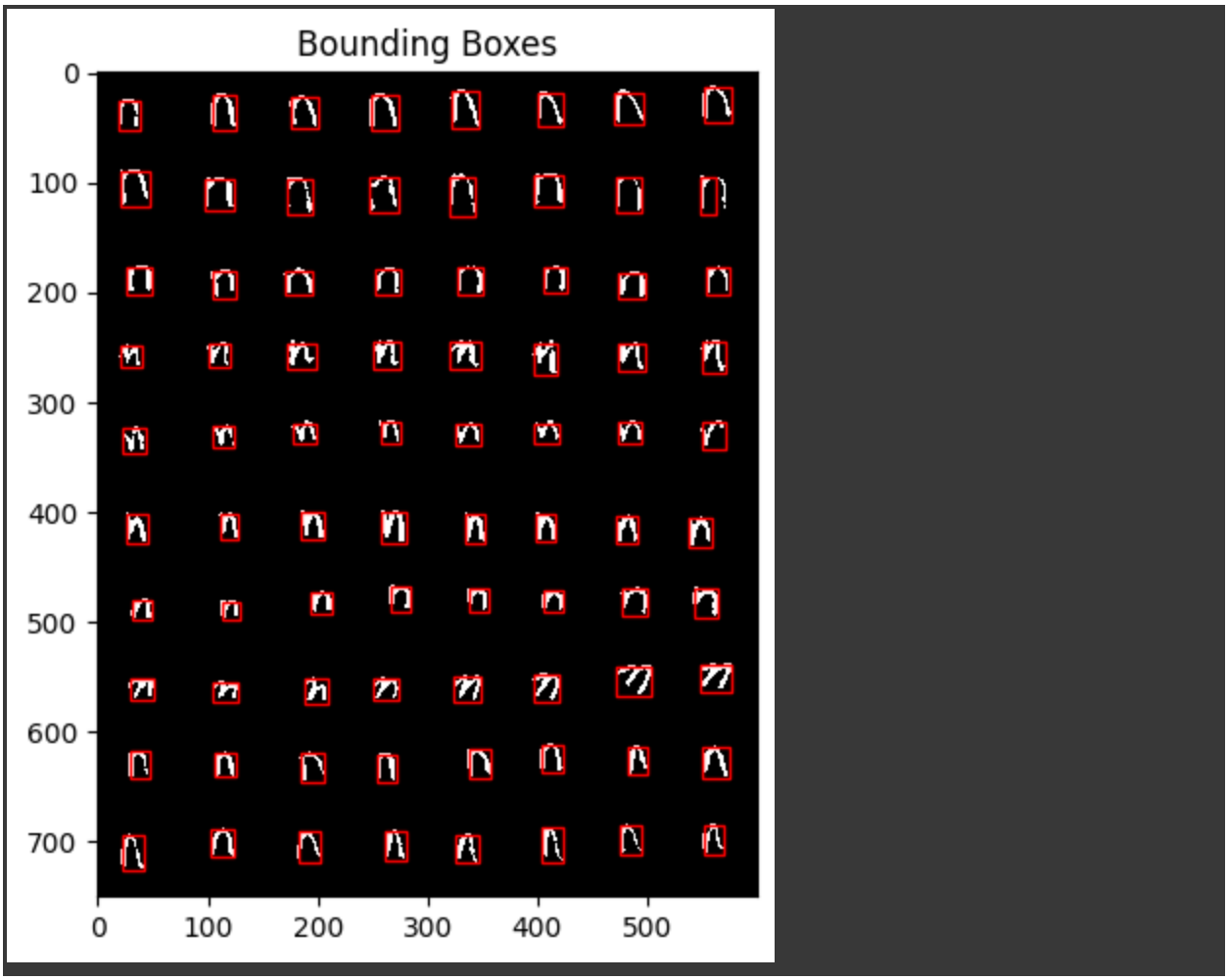

Bounding Boxes
0
100
200
300
400
500
600
700
0
100
200
300
400
500

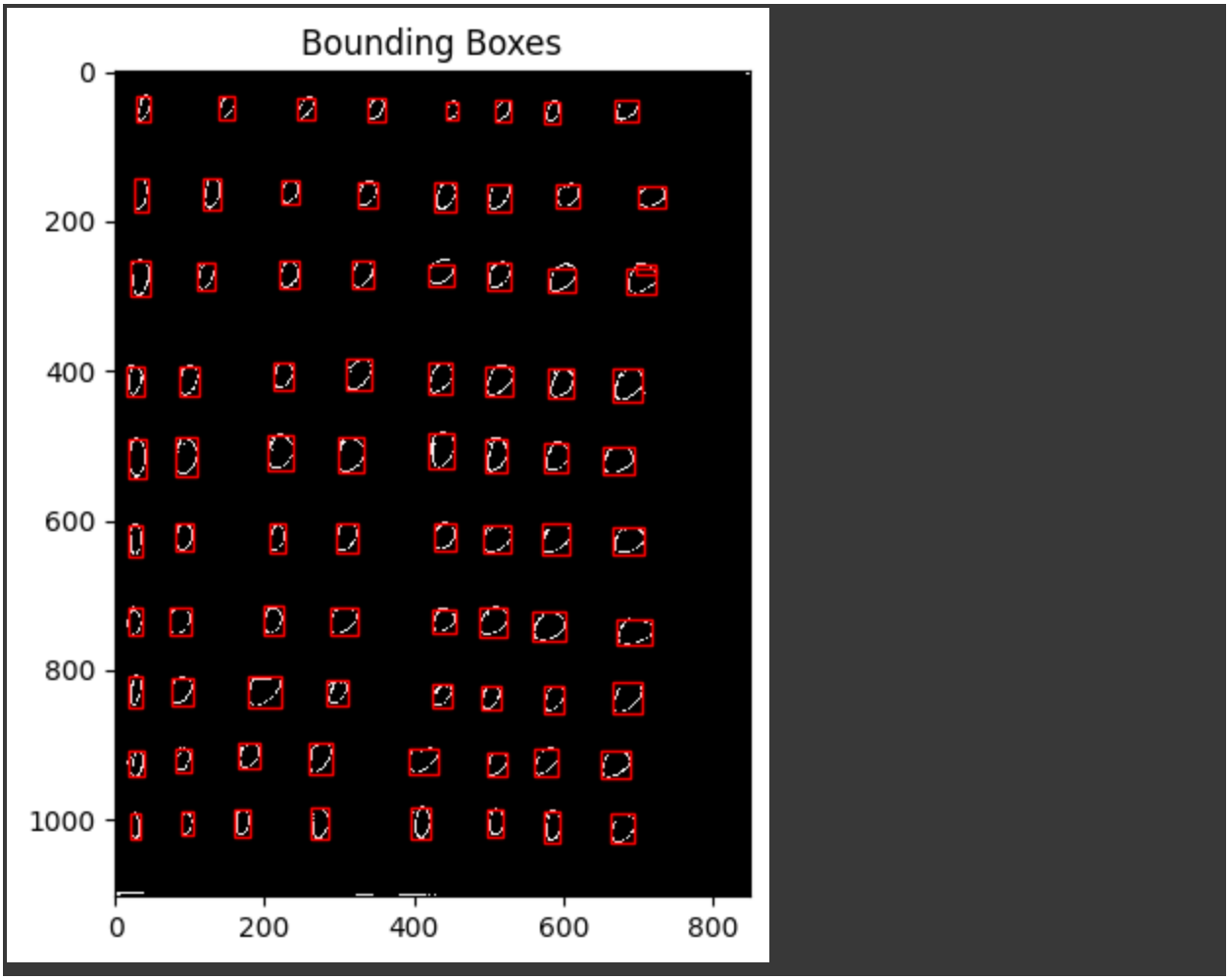

Bounding Boxes
0
200
400
600
800
1000
0
200
400
600
800

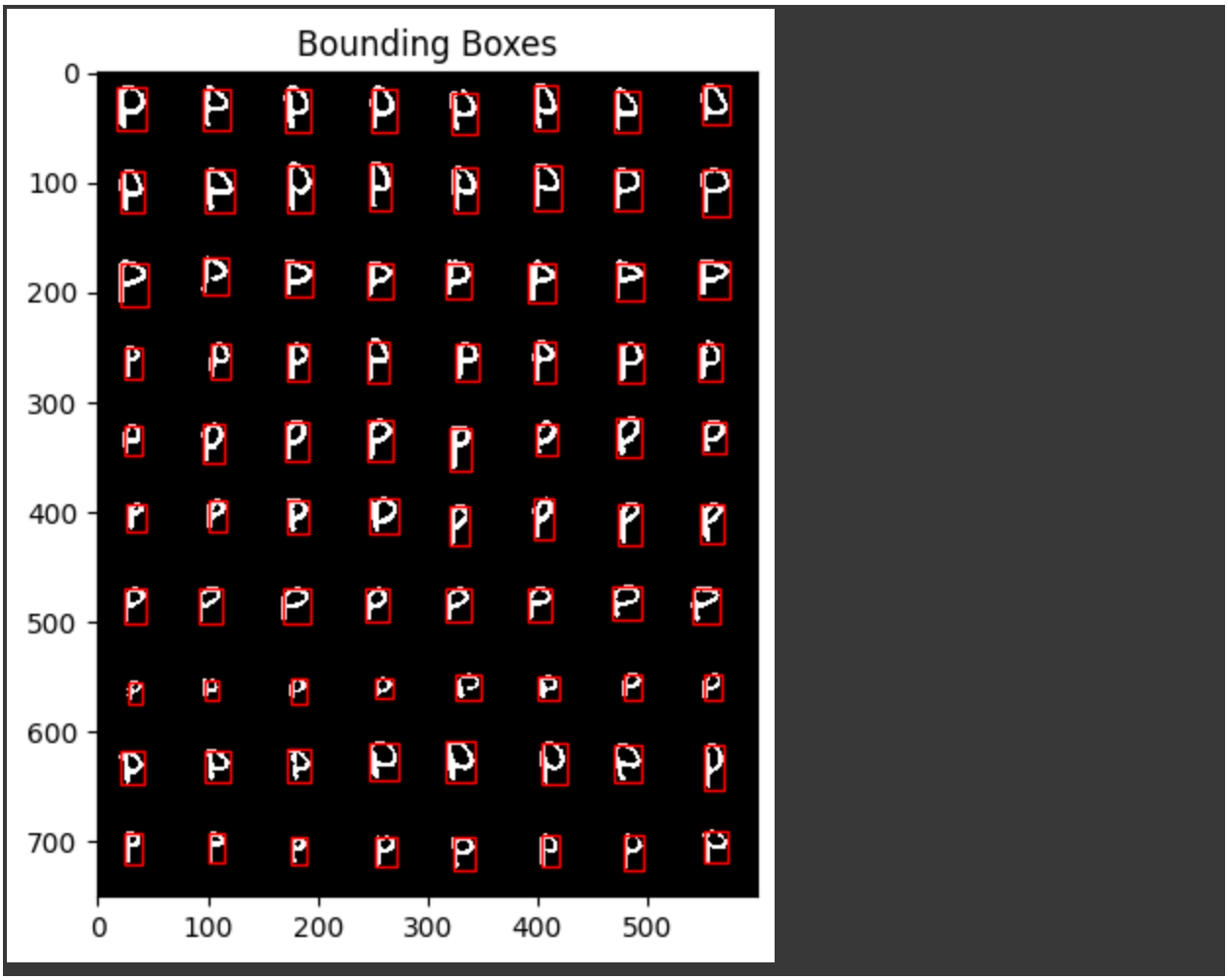

Bounding Boxes
0
200
400
600
800
1000
0
200
400
600
800

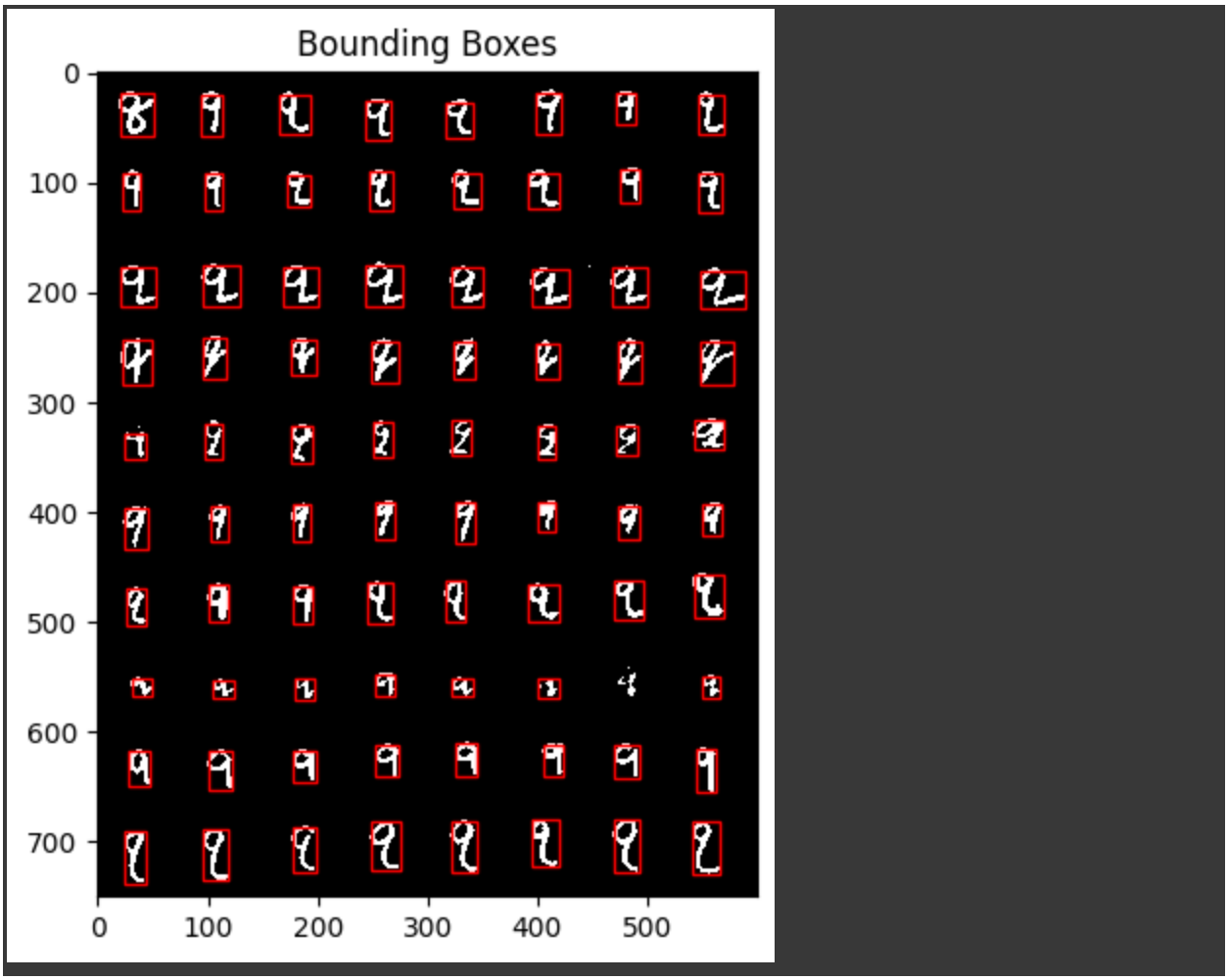

Bounding Boxes
0
200
400
600
800
1000
0
200
400
600
800

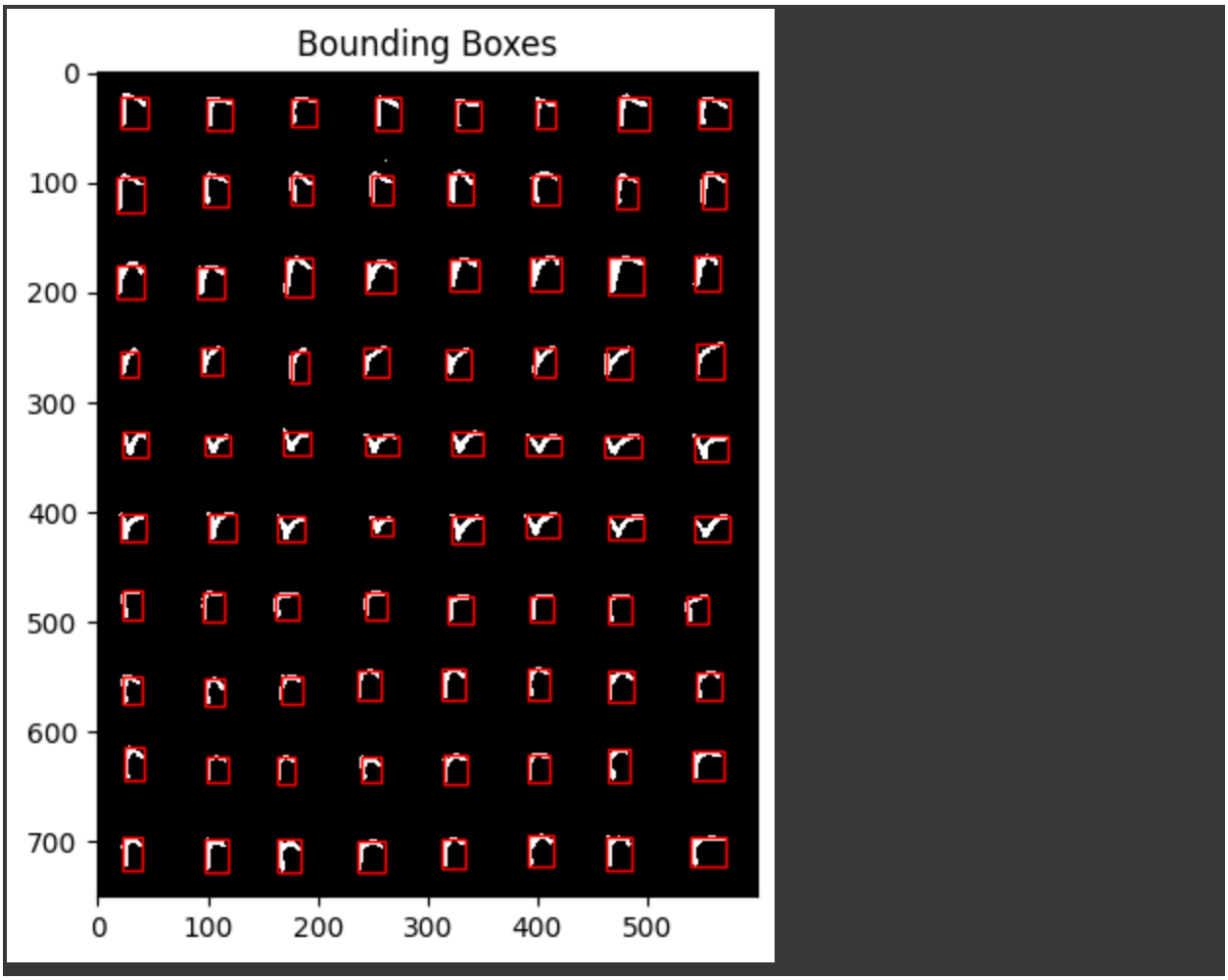

Bounding Boxes
0
100
200
300
400
500
600
700
0
100
200
300
400
500

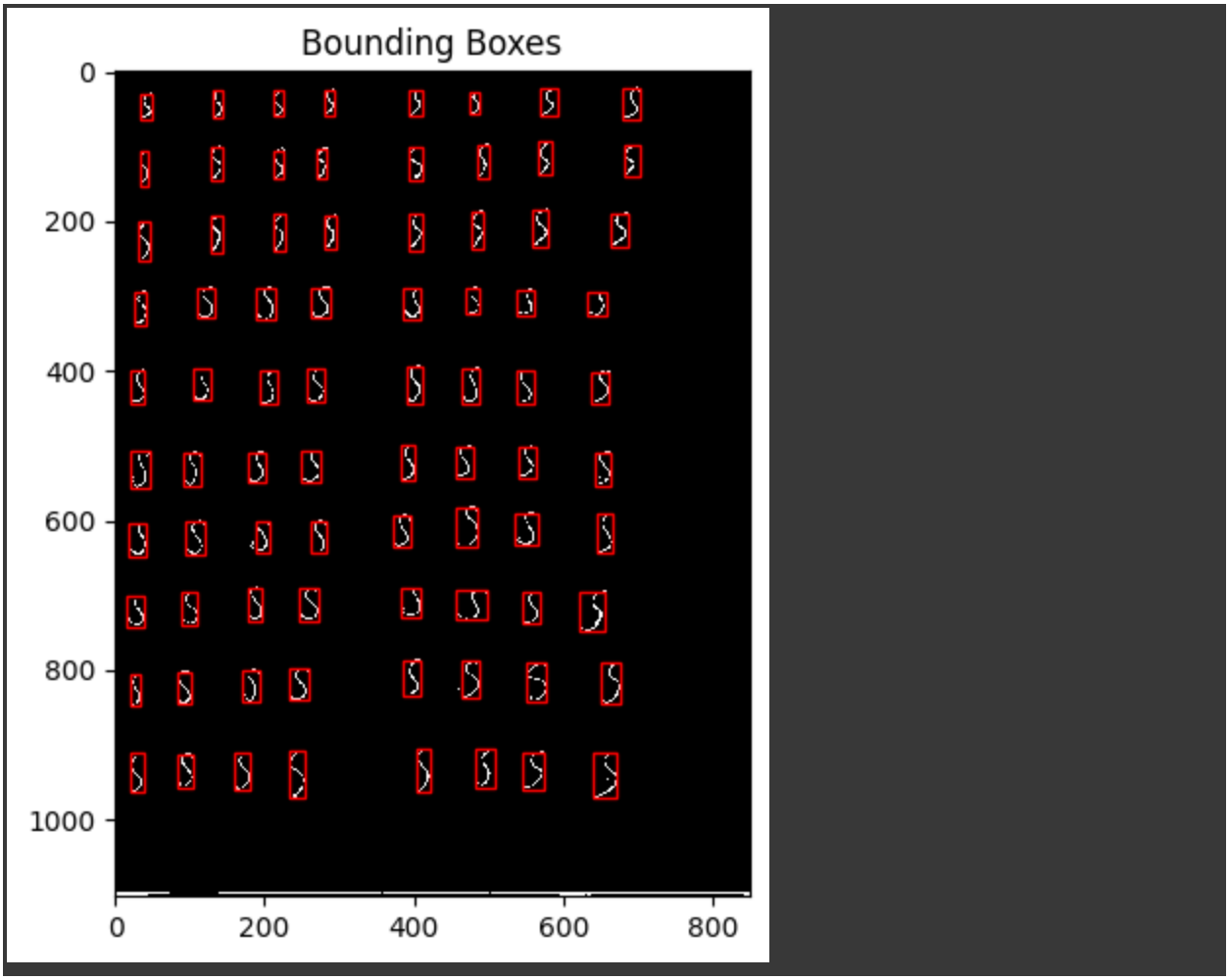

Bounding Boxes
0
100
200
300
400
500
600
700
0
100
200
300
400
500

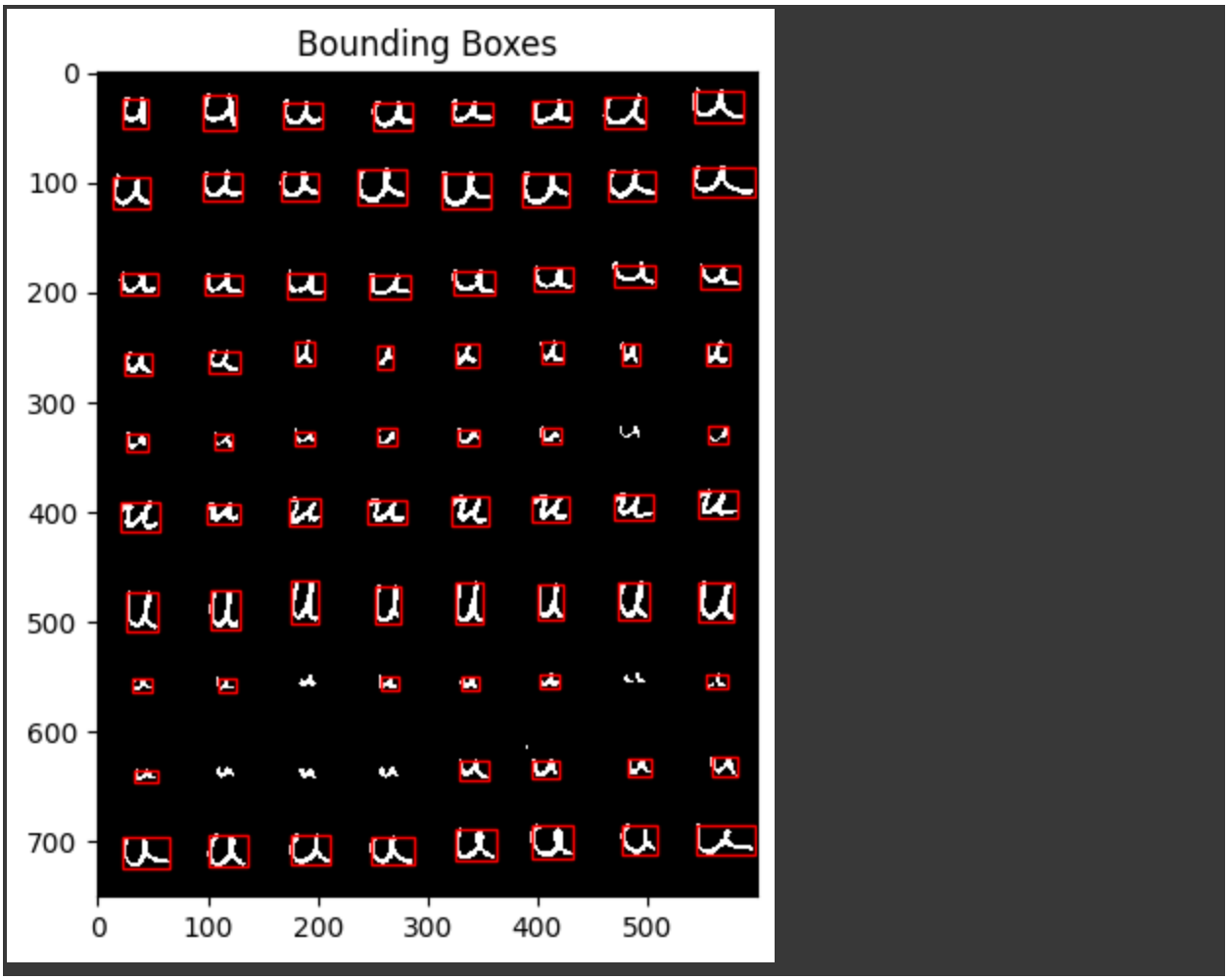

Bounding Boxes
0
200
400
600
800
1000
0
200
400
600
800

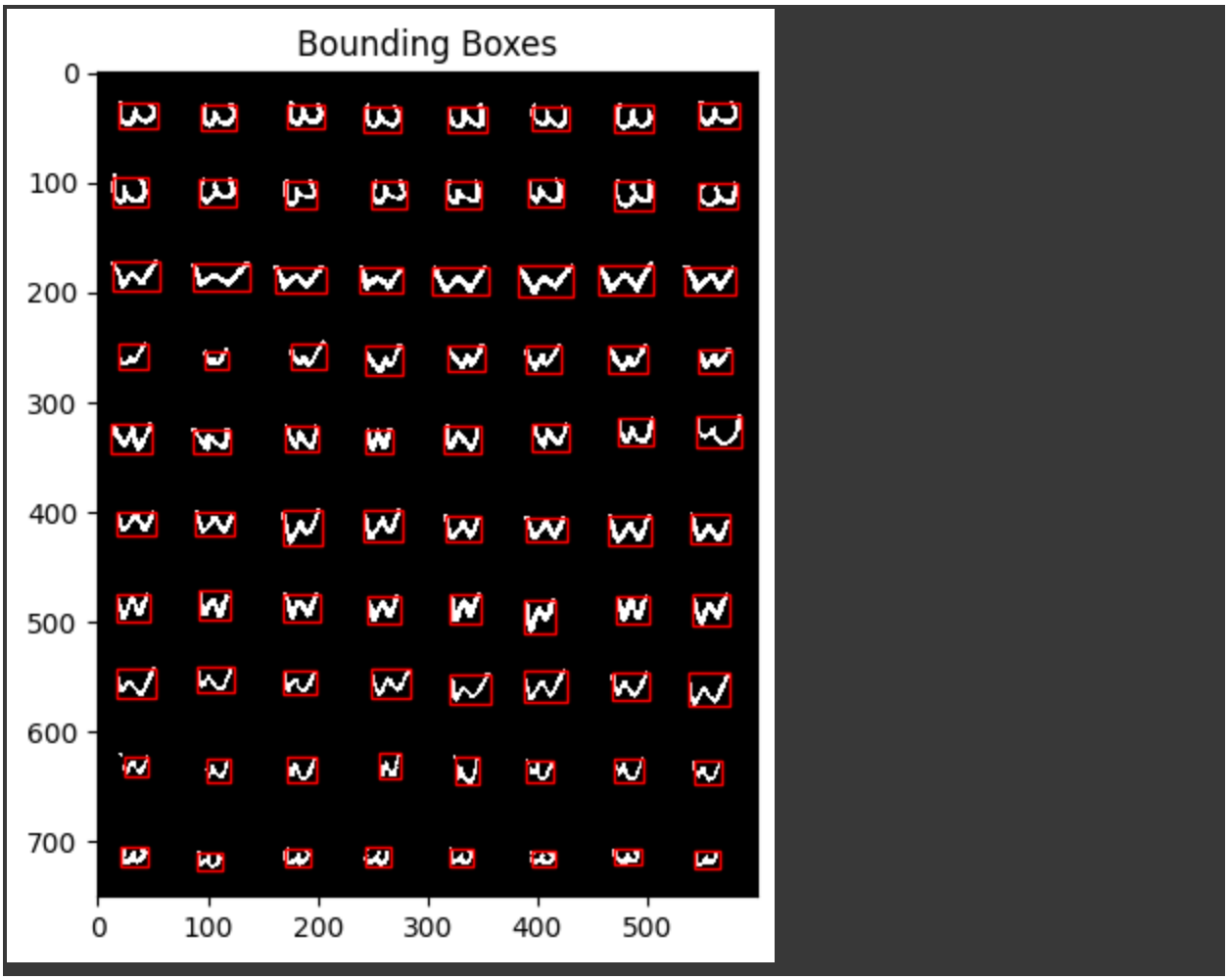

Bounding Boxes
0
100
200
300
400
500
600
700
0
100
200
300
400
500

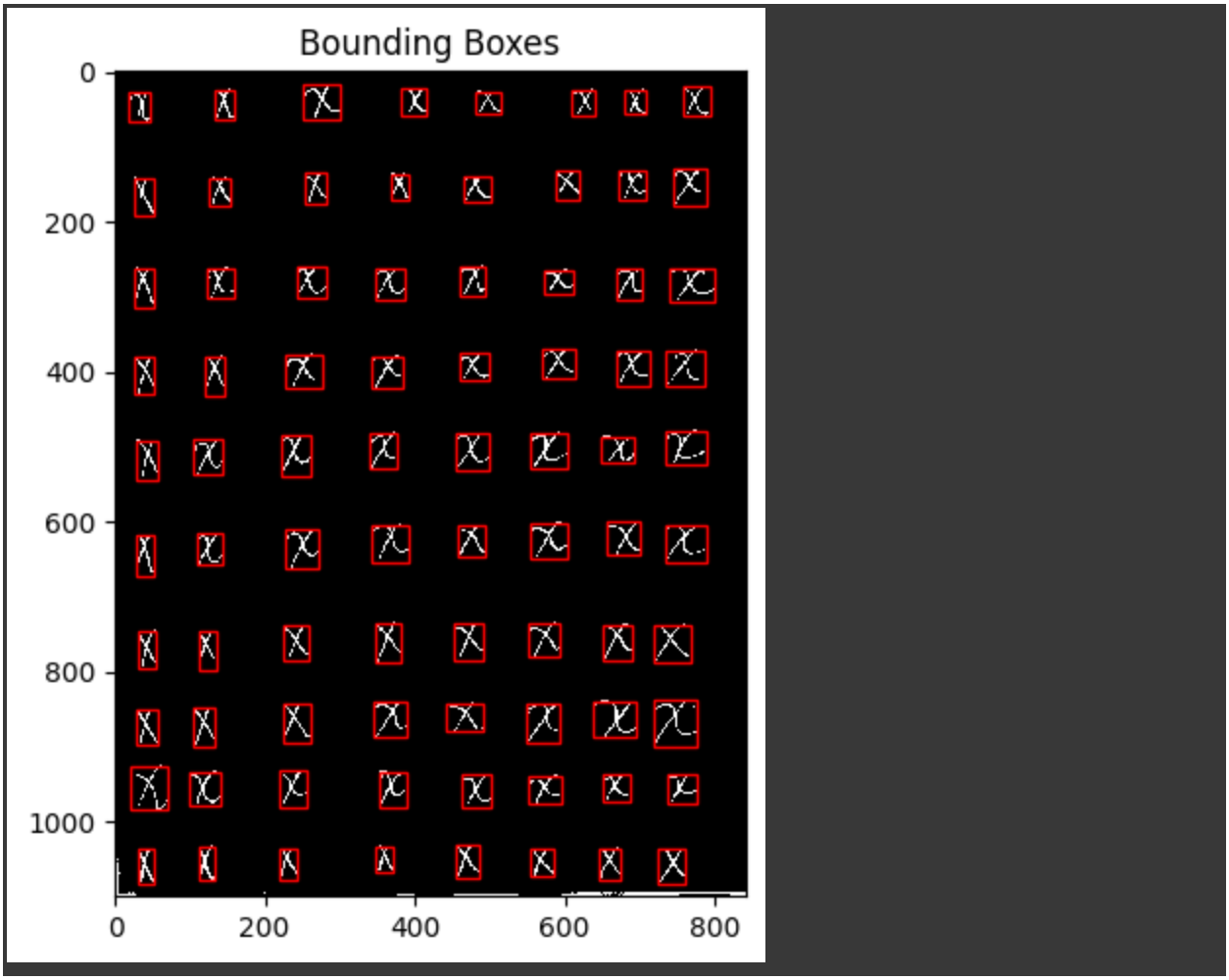

Bounding Boxes
0
100
200
300
400
500
600
700
0
100
200
300
400
500

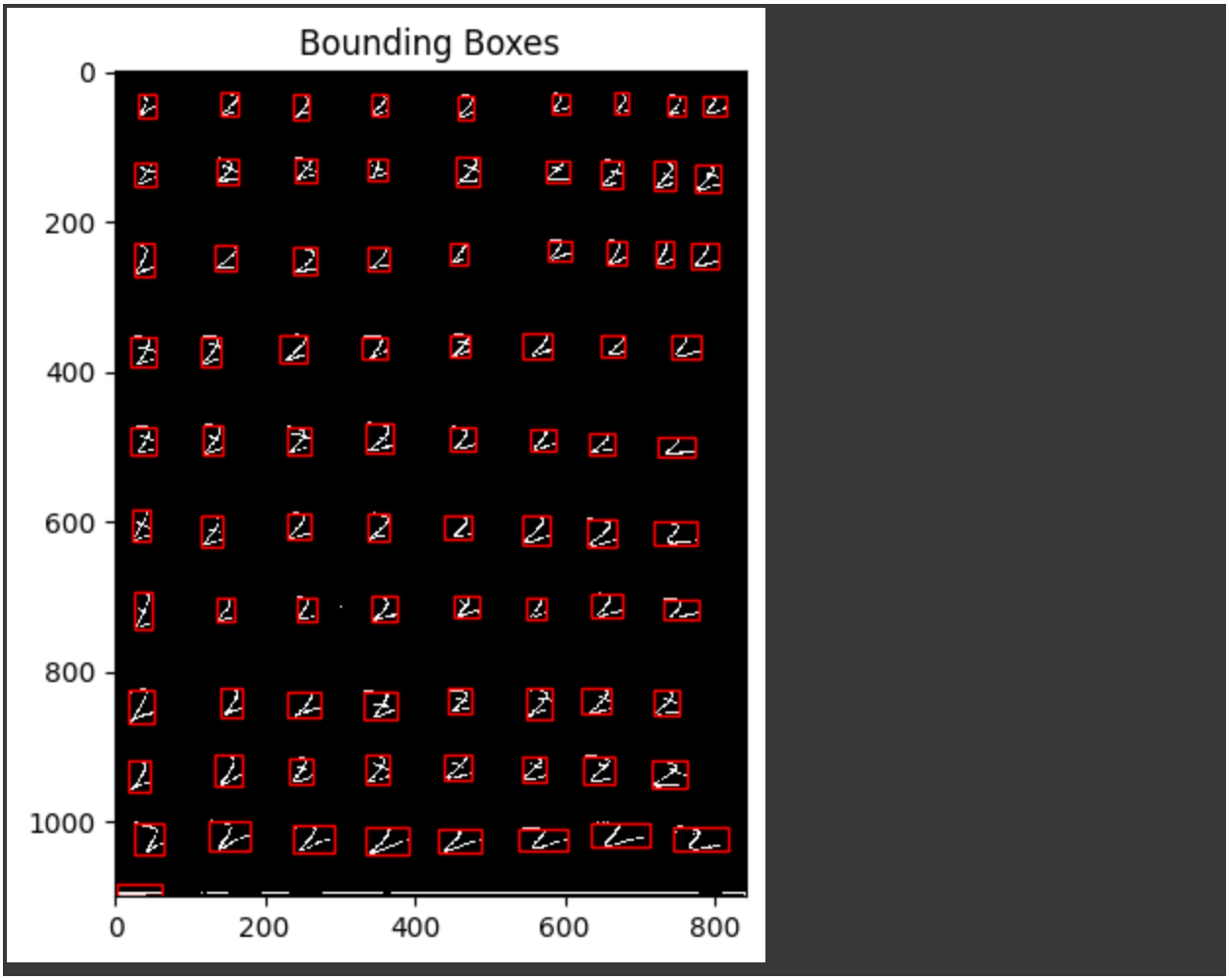

Bounding Boxes
0
100
200
300
400
500
600
700
0
100
200
300
400
500

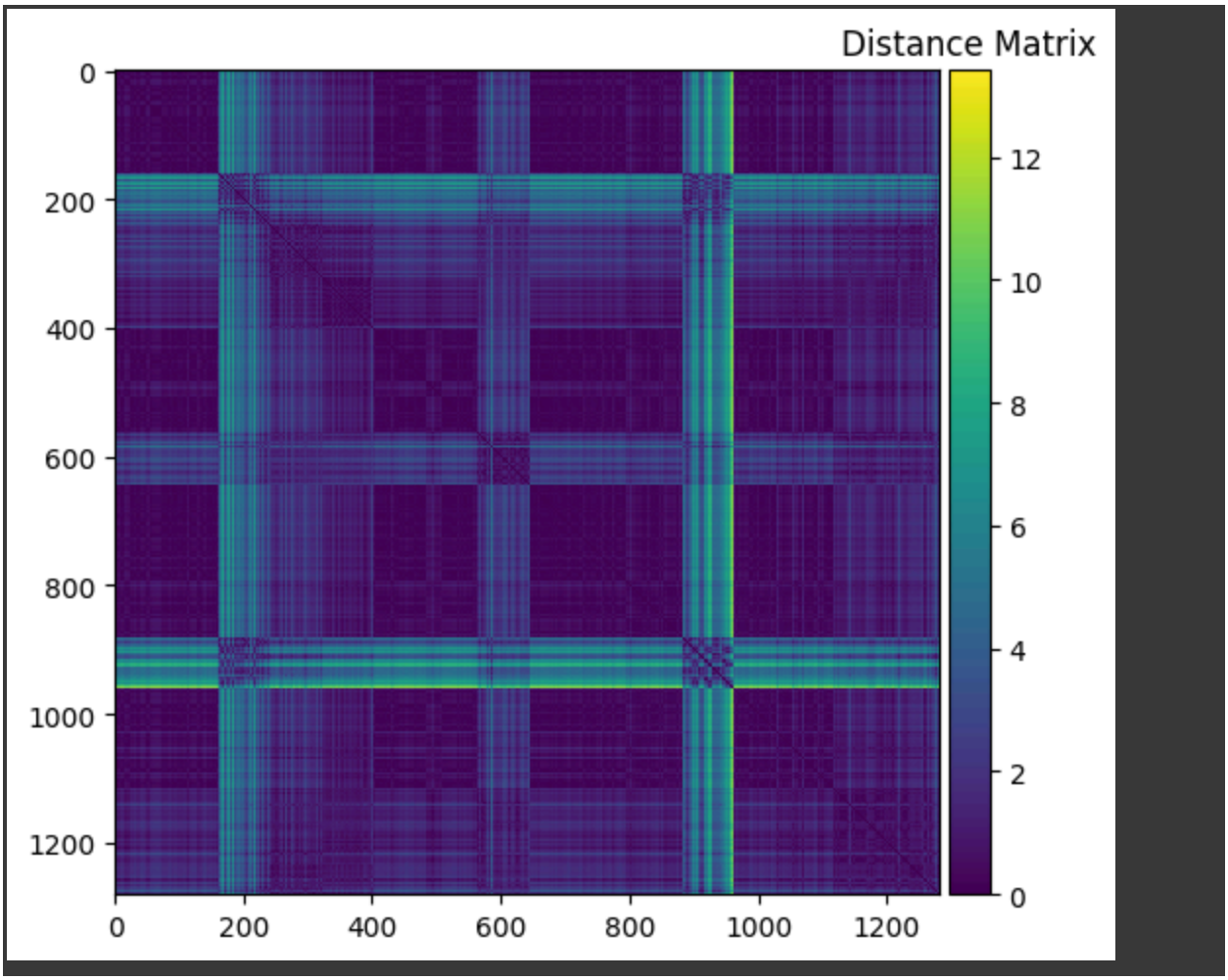

Bounding Boxes
0
200
400
600
800
1000
0
200
400
600
800

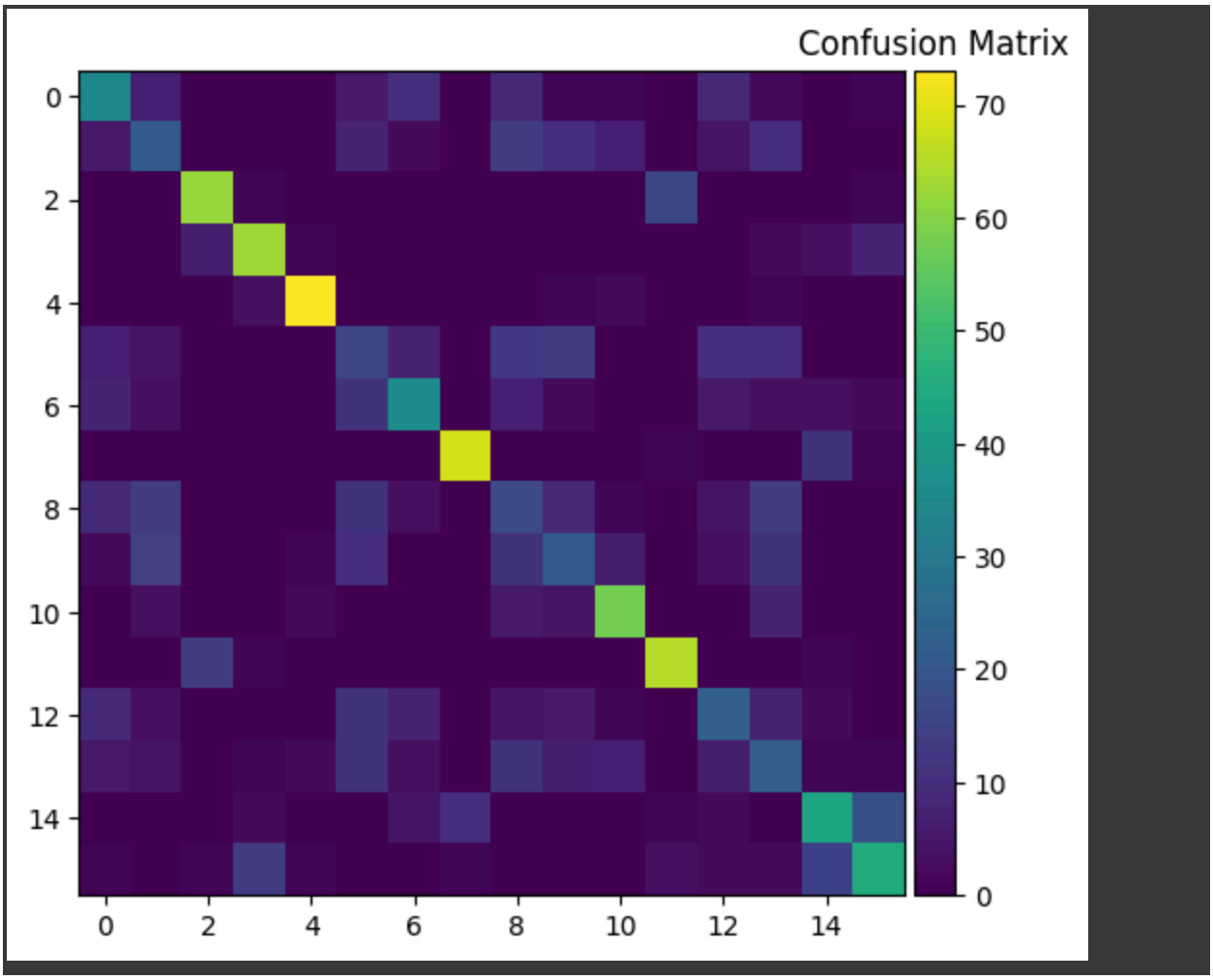

Bounding Boxes
0
100
200
300
400
500
600
700
0
100
200
300
400
500

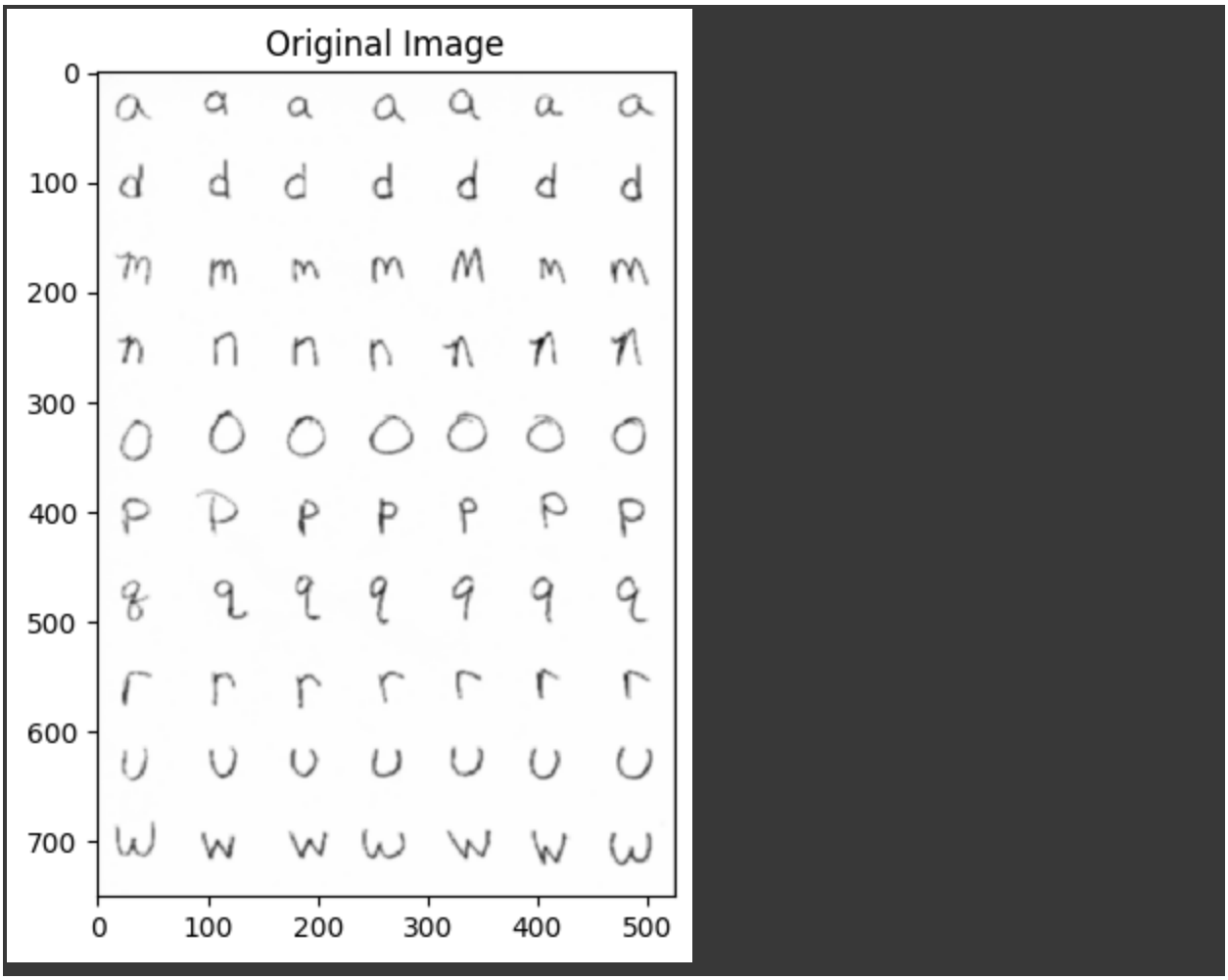

Bounding Boxes
0
100
200
300
400
500
600
700
0
100
200
300
400
500

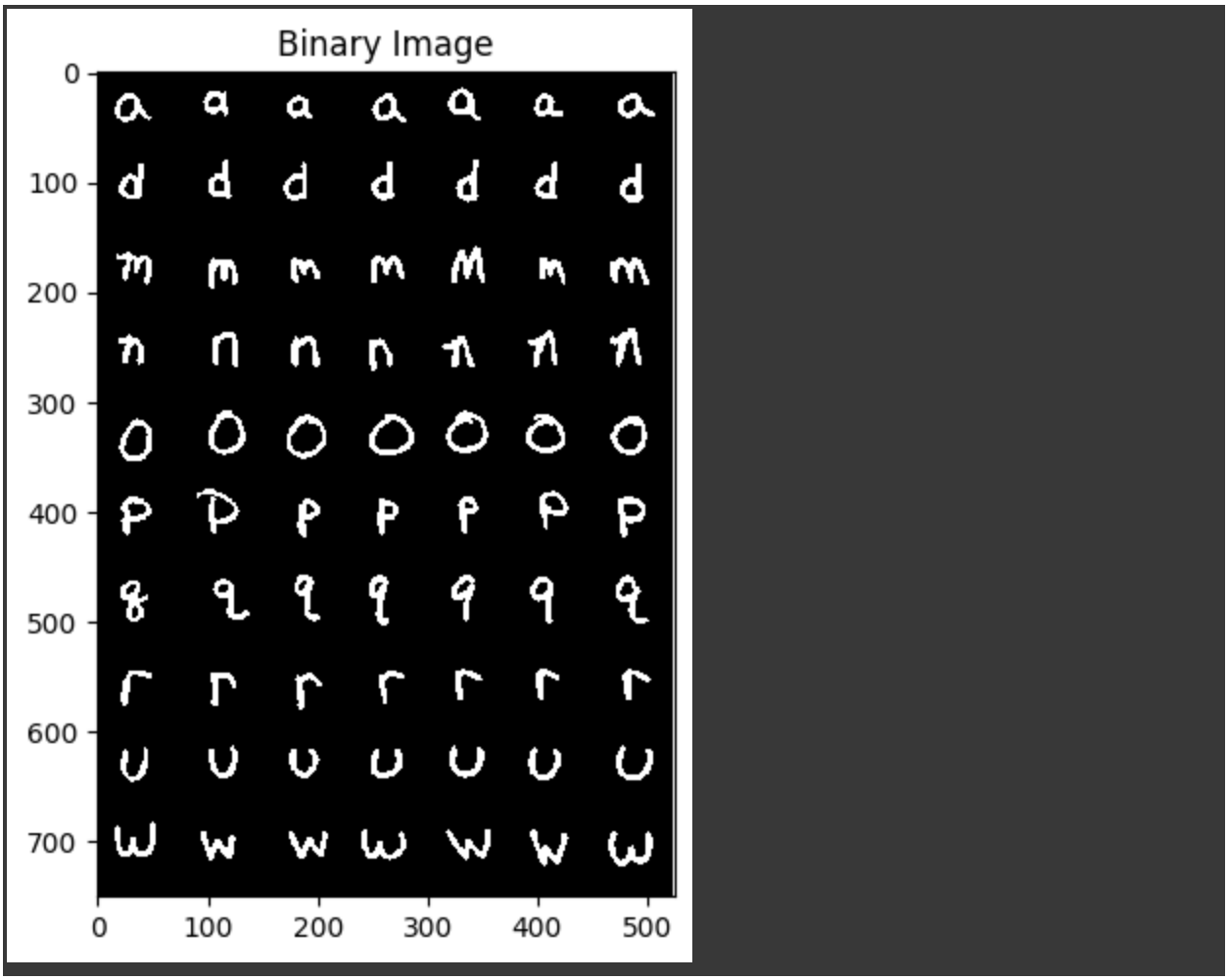

Bounding Boxes
0
200
400
600
800
1000
0
200
400
600
800

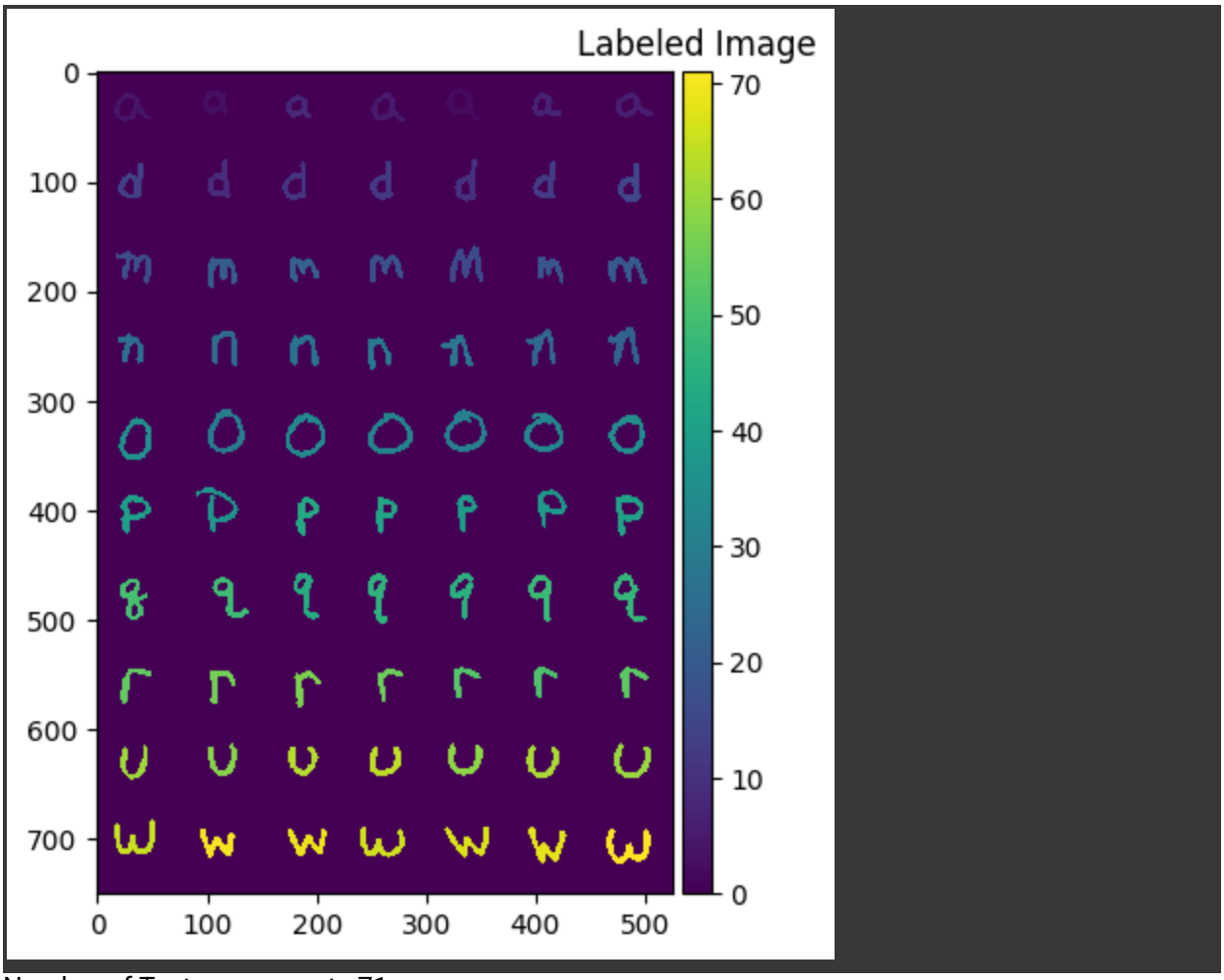

Bounding Boxes
0
200
400
600
800
1000
0
200
400
600
800


Distance Matrix
0
200
400
600
800
1000
1200
0
200
400
600
800
1000
1200
0
2
4
6
8
10
12

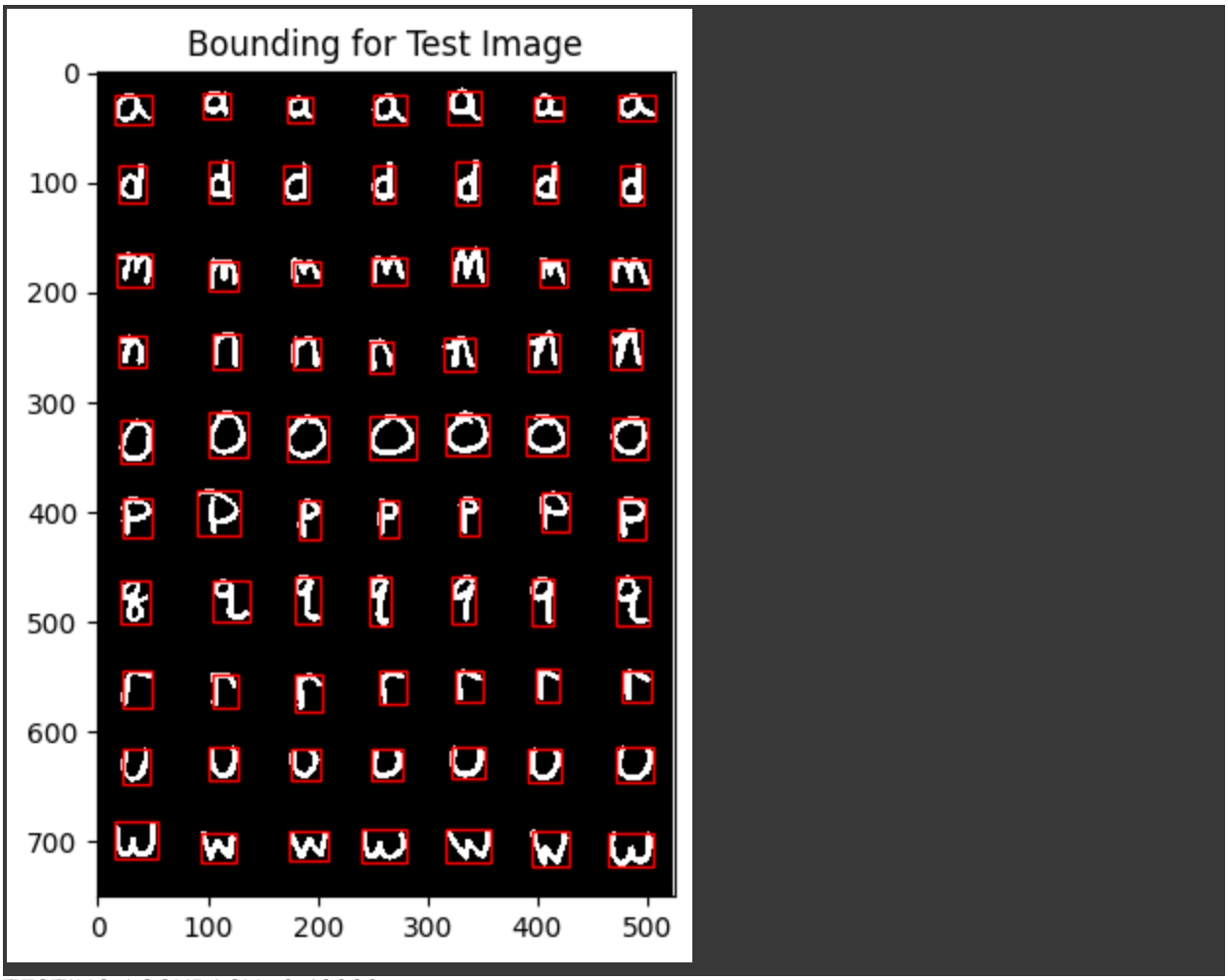

Confusion Matrix
0
2
4
6
8
10
12
14
70
60
50
40
30
20
10
0


Original Image
0
100
200
300
400
500
600
700
0
100
200
300
400
500


Binary Image
0
100
200
300
400
500
600
700
0
100
200
300
400
500



Number of Test components 71



TESTING ACCURACY: 0.40000