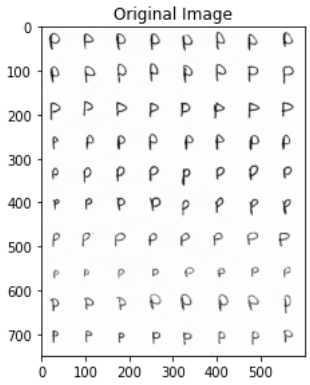
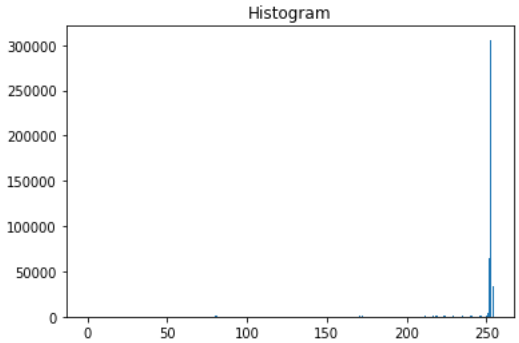
```
import os
import sys
import numpy as np
from sklearn.metrics import confusion_matrix
from scipy.spatial.distance import cdist
from skimage.measure import label, regionprops, moments, moments_central, moments_normalized, moments_hu
from skimage import io, exposure
import matplotlib.pyplot as plt
from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


def getFeatures(file_path, display_plot):
    filenames = []
    #Get all the files in folder
    for root_path, child_dirs, files in os.walk(file_path):
        for f in files:
            if f[0:4] != 'test':
                filenames.append(f)

    th = 225

    Features = []
    fList = []
    for fname in filenames:
        img = io.imread(file_path + fname)
        img_binary = (img < th).astype(np.double)

        img_label = label(img_binary, background=0)
        print('\n')
        print(fname)
        print('shape: ')
        print(img.shape)
        print('\n')
        if display_plot:
            io.imshow(img)
            plt.title('Original Image')
            io.show()

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

            io.imshow(img_binary)
            plt.title('Binary Image')
            io.show()

            io.imshow(img_label)
            plt.title('Labeled Image')
            io.show()

        regions = regionprops(img_label)
        # find the threshold used to remove the small noise
        thre_noise = {'height':[10.0, 80.0], 'width':[12.0, 85.0]}
        io.imshow(img_binary)
        ax = plt.gca()
        for props in regions:
            minr, minc, maxr, maxc = props.bbox

            # apply size thresh
            if (maxr - minr < thre_noise['height'][0] or maxc - minc < thre_noise['width'][0]
            or maxr - minr > thre_noise['height'][1] or maxc - minc > thre_noise['width'][1]):
                continue

            if display_plot:
                ax.add_patch(Rectangle((minc, minr), maxc - minc, maxr - minr, fill=False,
                             edgecolor='red', linewidth=1))
            roi = img_binary[minr:maxr, minc:maxc]
            m = moments(roi)
            cr = m[0, 1] / m[0, 0]
            cc = m[1, 0] / m[0, 0]
            mu = moments_central(roi, center=(cr, cc))
            nu = moments_normalized(mu)
            hu = moments_hu(nu)
```

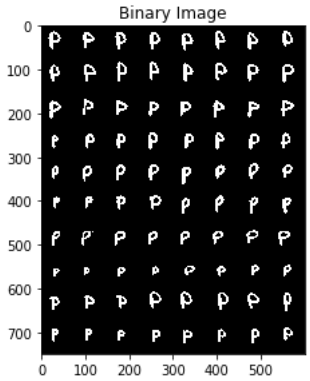
```
77
78              Features.append(hu)
79              fList.append(fname[0])
80          if display_plot:
81              plt.title('Bounding Boxes')
82              io.show()
83
84      # normalization
85      feature_array = np.array(Features)
86      print('\nFeatures array shape: ')
87      print(feature_array.shape)
88      expect = np.mean(feature_array, axis=0, dtype=np.double)
89      s_deviation = np.std(feature_array, axis=0, dtype=np.double)
90      norm_Features = []
91      for i in range(feature_array.shape[0]):
92          feature_array[i] -= expect
93          feature_array[i] /= s_deviation
94          norm_Features.append(feature_array[i])
95
96      if display_plot:
97          drawBounding(file_path, norm_Features, fList, expect, s_deviation)
98
99      return norm_Features, fList, expect, s_deviation
100
101
102 def drawBounding(file_path, Features, fList, mean, std):
103     # file_path = './H1-16images/'
104     # find all files with names
105     filenames = []
106     # validate in Unix or Windows platform
107     for root_path, child_dirs, files in os.walk(file_path):
108         for f in files:
109             if f[0:4] != 'test':
110                 filenames.append(f)
111
112     th = 225
113     num_correct = 0.0
114     num_total = len(fList)
115
116     for fname in filenames:
117         img = io.imread(file_path + fname)
118         img_binary = (img < th).astype(np.double)
119
120         img_label = label(img_binary, background=0)
121
122         regions = regionprops(img_label)
123         # threshold size for boxes
124         thre_noise = {'height':[10.0, 80.0], 'width':[10.0, 80.0]}
125         io.imshow(img_binary)
126         ax = plt.gca()
127
128         for props in regions:
129             minr, minc, maxr, maxc = props.bbox
130
131             # size threshold
132             if (maxr - minr < thre_noise['height'][0] or maxc - minc < thre_noise['width'][0]
133             or maxr - minr > thre_noise['height'][1] or maxc - minc > thre_noise['width'][1]):
134                 continue
135
136             ax.add_patch(Rectangle((minc, minr), maxc - minc, maxr - minr, fill=False,
137                         edgecolor='red', linewidth=1))
138             #print minc, minr, maxc, maxr, maxr-minr, maxc-minc
139             roi = img_binary[minr:maxr, minc:maxc]
140             m = moments(roi)
141             cr = m[0, 1] / m[0, 0]
142             cc = m[1, 0] / m[0, 0]
143             mu = moments_central(roi, center=(cr, cc))
144             nu = moments_normalized(mu)
145             hu = moments_hu(nu)
146
147             # change to other feature to accerate
148             norm_hu = hu - mean
149             norm_hu /= std
150             D = cdist([norm_hu], Features)
151             # print D
152
153             D_index = np.argsort(D, axis=1)
```

```
            # get the 2nd index of each row
            Ypred = fList[D_index[0][1]]
            if fname[0] == Ypred:
                num_correct += 1
            plt.text(maxc, minr, Ypred, bbox=dict(facecolor='red', alpha=0.5))
            # inner for end
        t = 'Bounding Boxes: ' + fname[0]
        plt.title(t)
        io.show()
    accuracy_rate = num_correct / num_total
    print(accuracy_rate)


# use to test in this file
def train_predict(file_path, display_plot):
    print(file_path)
    Features, fList, mean, std = getFeatures(file_path, display_plot)
    D = cdist(Features, Features)
    if display_plot:
        io.imshow(D)
        plt.title('Distance Matrix')
        io.show()

    D_index = np.argsort(D, axis=1)
    #2nd index
    Ypred = [fList[i[1]] for i in D_index]

    num_correct = 0.0
    num_total = len(Ypred)
    for i in range(num_total):
        if fList[i] == Ypred[i]:
            num_correct += 1
    accuracy_rate = num_correct / num_total
    print(accuracy_rate)

    confM = confusion_matrix(fList, Ypred)
    if display_plot:
        io.imshow(confM)
        plt.title('Confusion Matrix')
        io.show()

    return Ypred, confM, accuracy_rate



if __name__ == "__main__":
    file_path = '/content/images/'
    #file_path = sys.argv[1]
    display_plot = True
    Ypred, confM, accuracy = train_predict(file_path, display_plot)

```

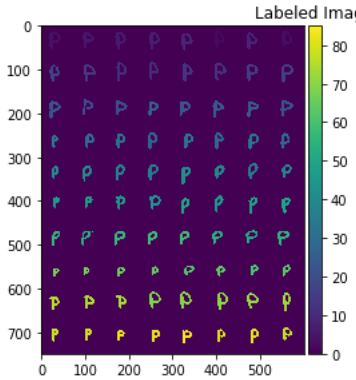
```
/content/images/


p.bmp
shape:
(750, 600)
```

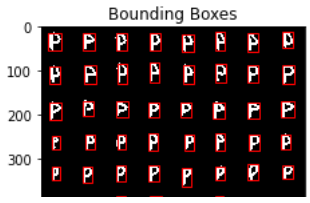


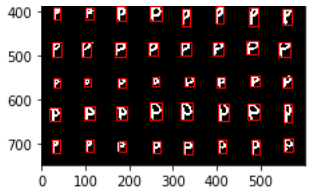




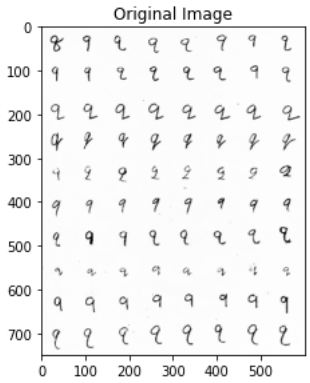

```
/usr/local/lib/python3.8/dist-packages/skimage/io/_plugins/matplotlib_plugin.py:15
  lo, hi, cmap = _get_display_range(image)
```

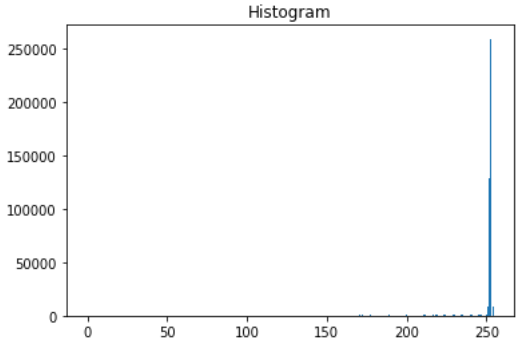


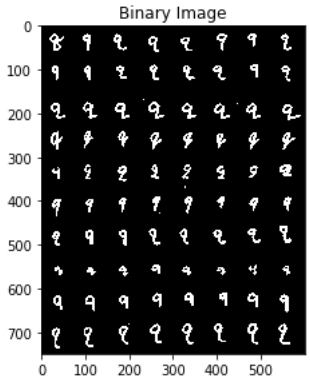


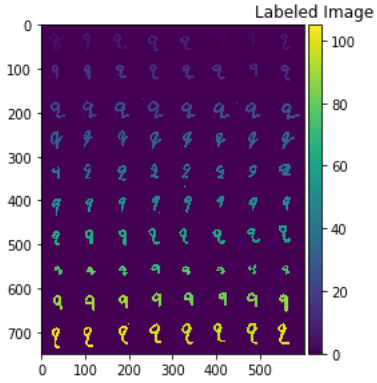


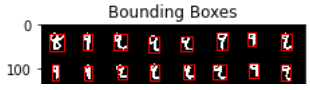
```
q.bmp
shape:
(750, 600)
```

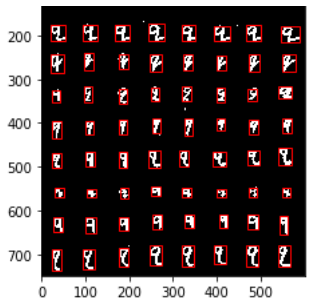




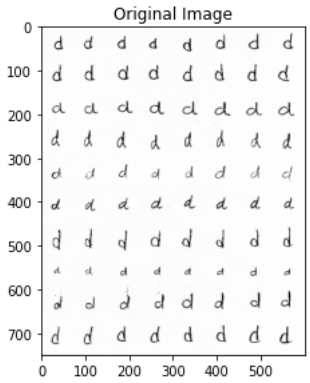


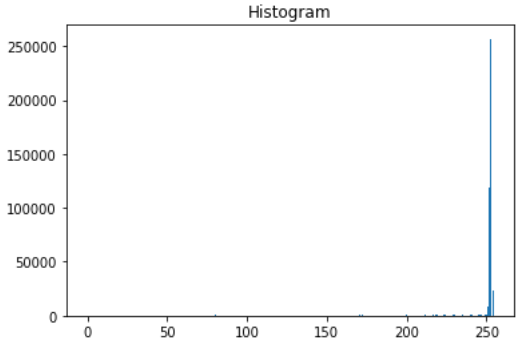


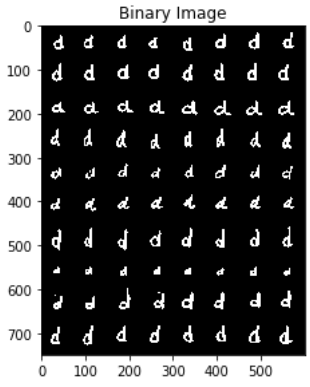


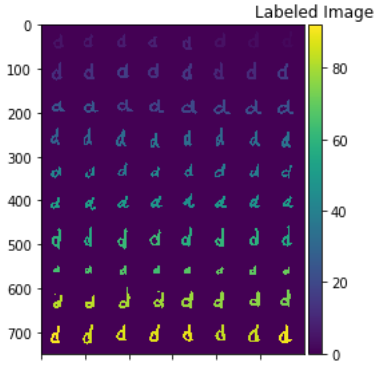


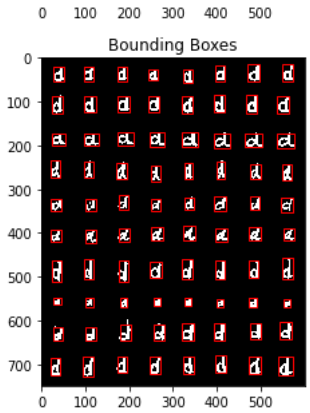
```
d.bmp
shape:
(750, 600)
```



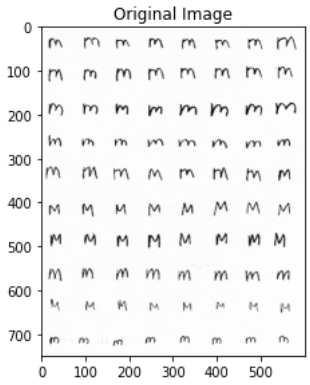


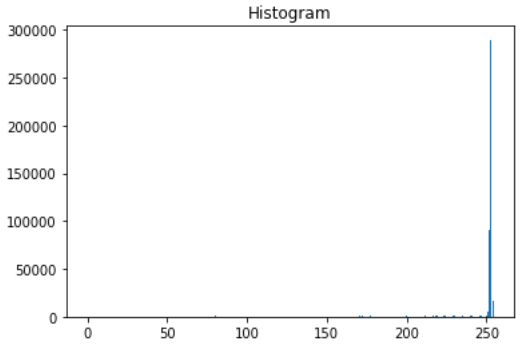


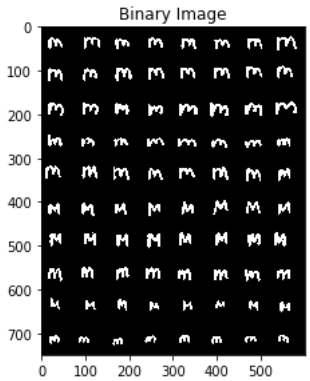


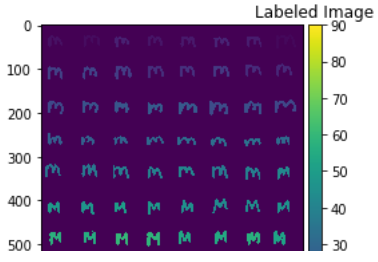


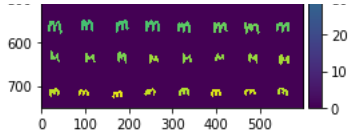

```
m.bmp
shape:
(750, 600)
```

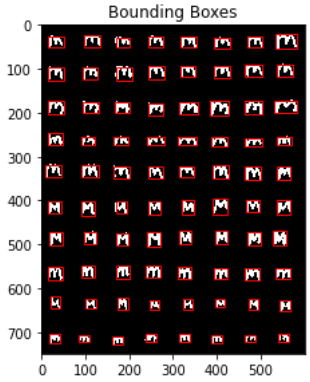




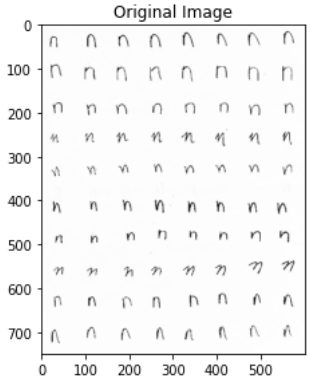


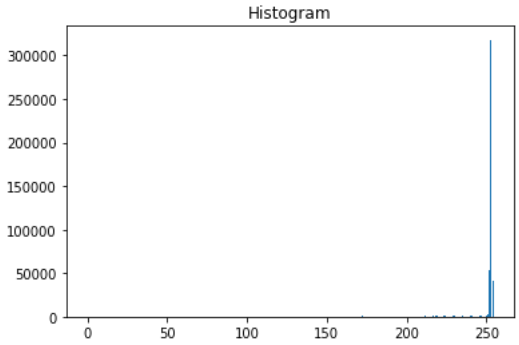


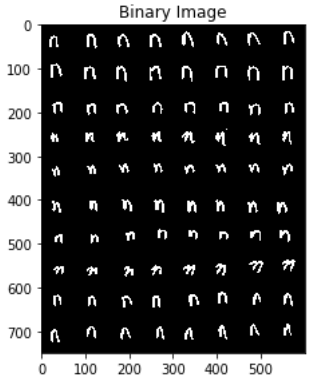


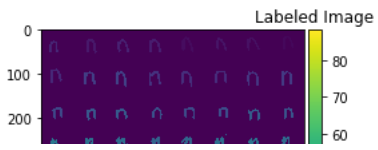


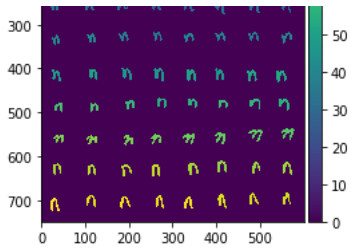

```
n.bmp
shape:
(750, 600)
```

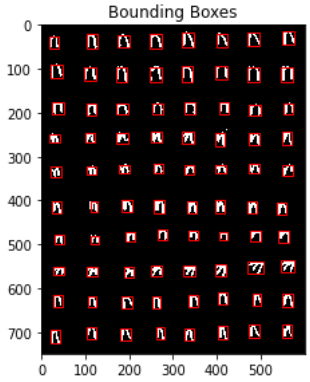




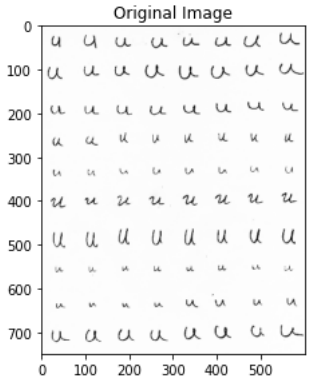


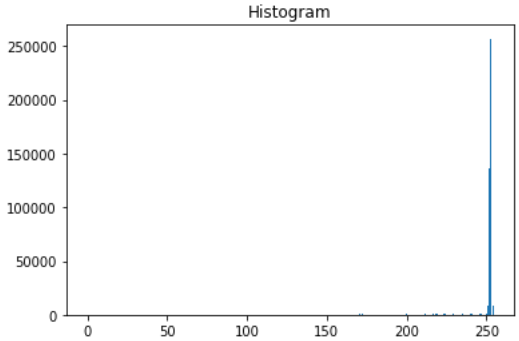


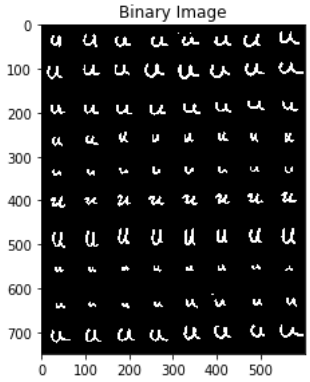


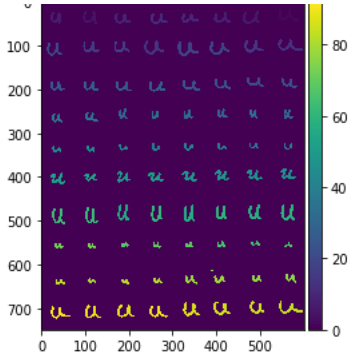


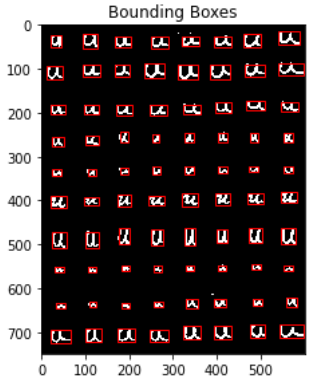

```
u.bmp
shape:
(750, 600)
```



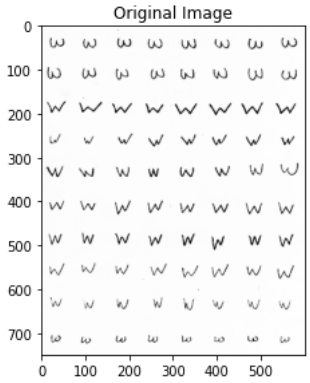


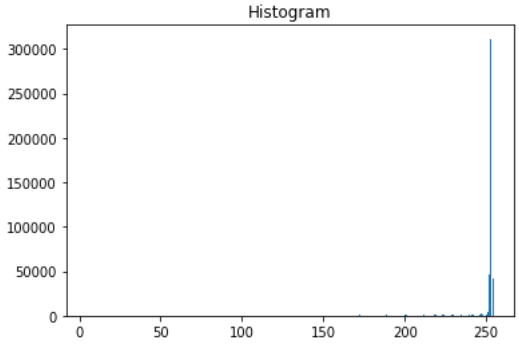


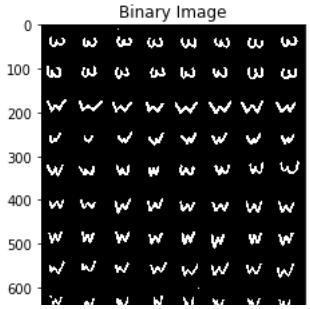


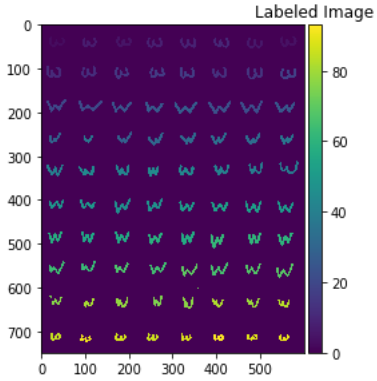


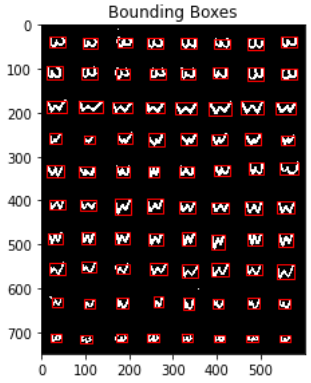

```
w.bmp
shape:
(750, 600)
```



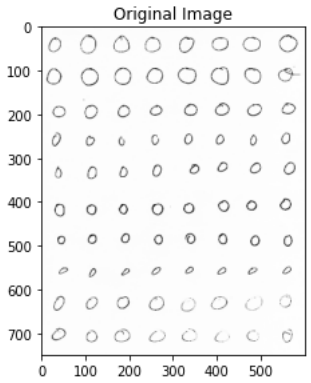


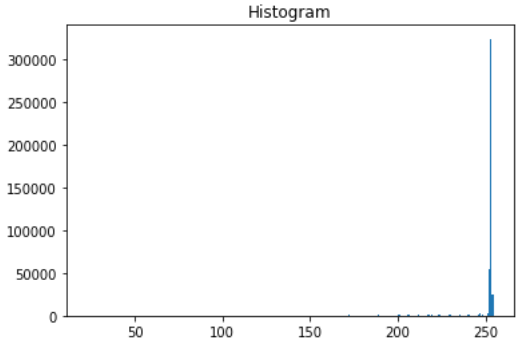


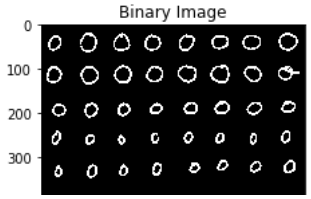


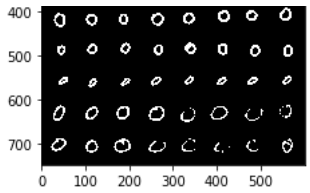


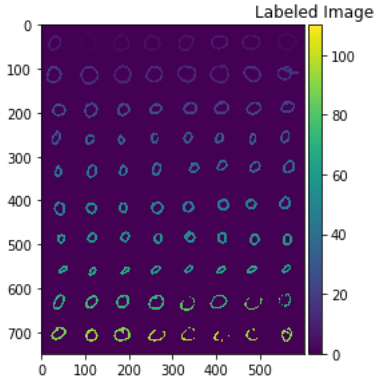

```
o.bmp
shape:
(750, 600)
```

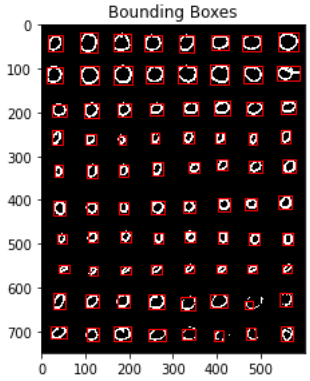




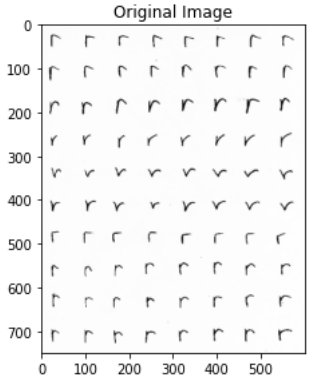


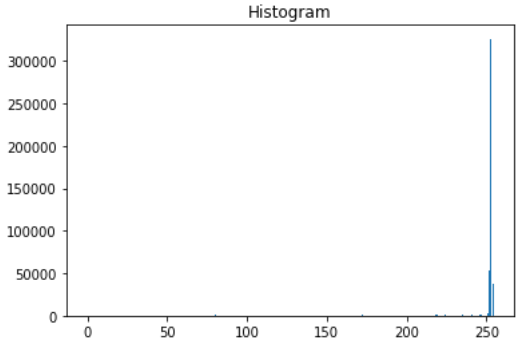


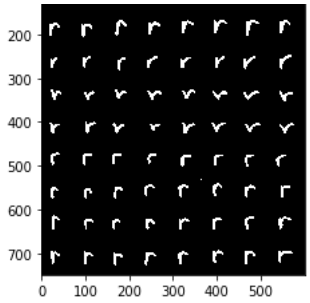


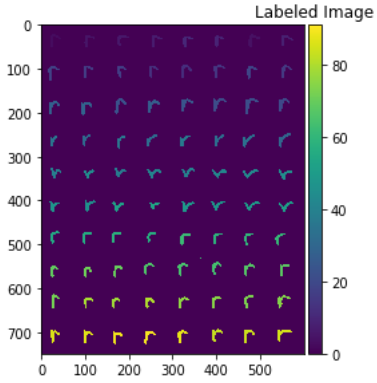


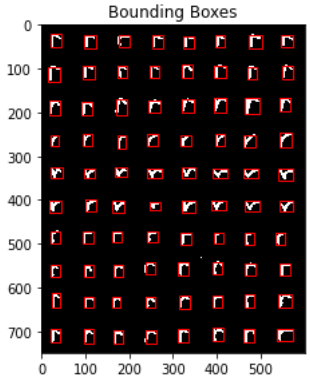

```
r.bmp
shape:
(750, 600)
```



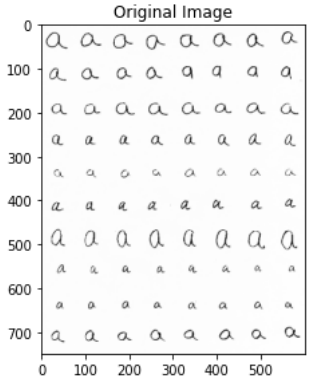


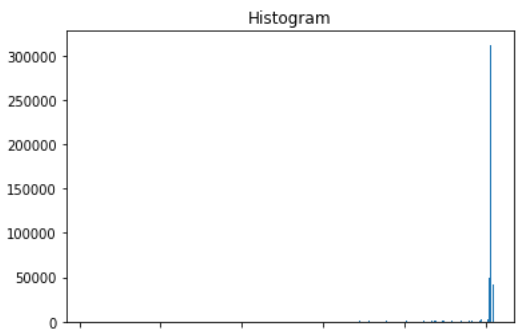


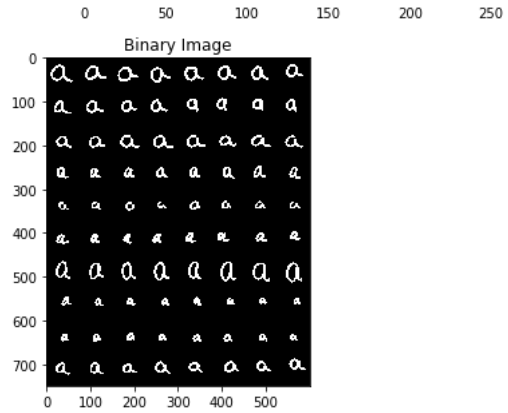


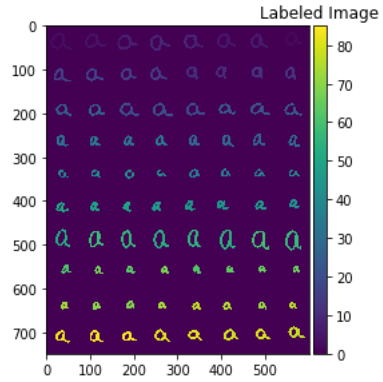


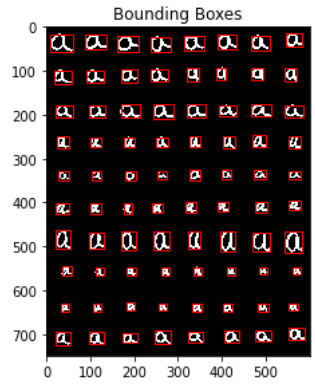

```
a.bmp
shape:
(750, 600)
```



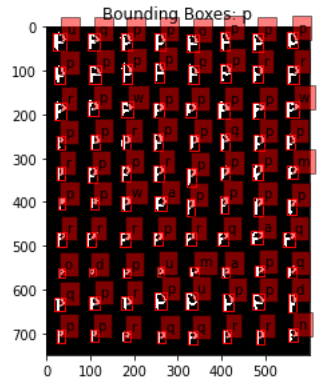


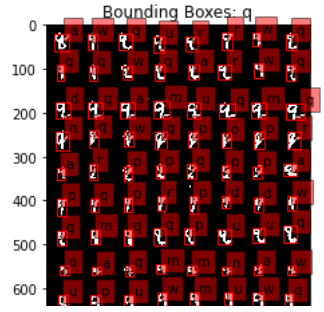


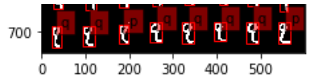


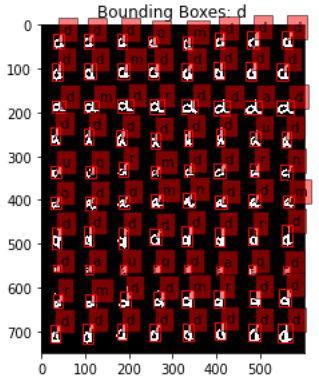


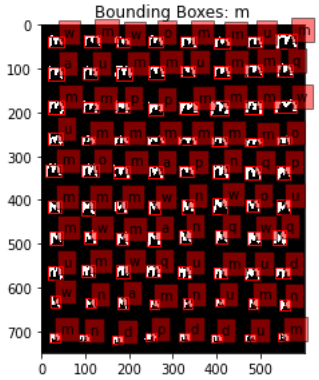

```
Features array shape:
(800, 7)
```

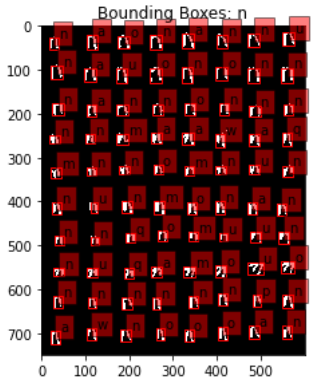


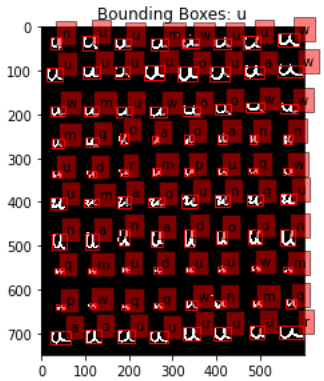


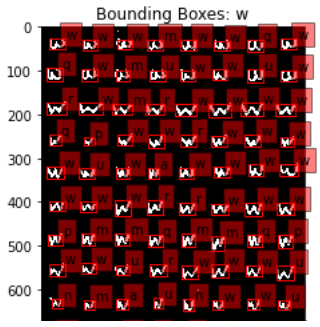


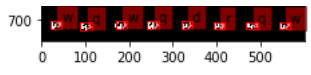

Bounding Boxes: d

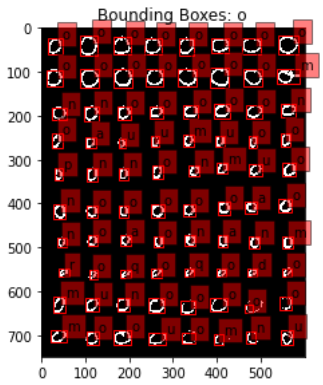

Bounding Boxes: m

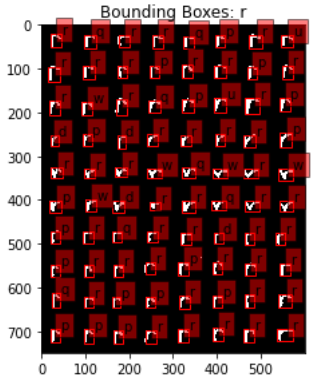

Bounding Boxes: n

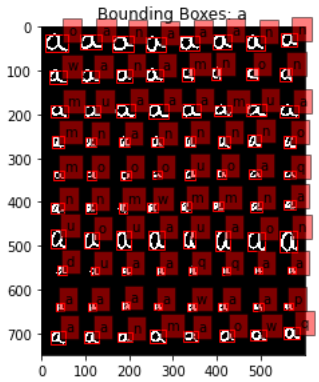

Bounding Boxes: u


Bounding Boxes: w

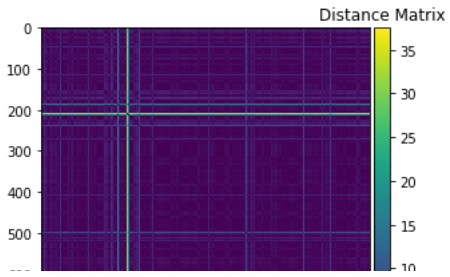









```
0.465
/usr/local/lib/python3.8/dist-packages/skimage/io/_plugins/matplotlib_plugin.py:15
  lo, hi, cmap = _get_display_range(image)
```

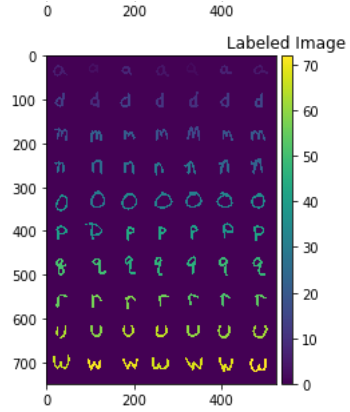


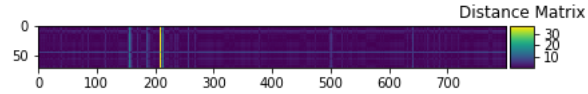

```
import os
import sys
import numpy as np
from sklearn.metrics import confusion_matrix
from scipy.spatial.distance import cdist
from skimage.measure import label, regionprops, moments, moments_central, moments_normalized, moments_hu
from skimage import io, exposure
import matplotlib.pyplot as plt
from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


```

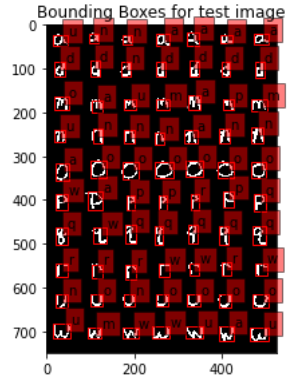
```
def testFeatures(filename, display_plot=False):

    th = 225
    result = {}
    Features = []
    coordinate = []
    img = io.imread(filename)

    print(img.shape)

    img_binary = (img < th).astype(np.double)
    result['img_binary'] = img_binary

    img_label = label(img_binary, background=0)

    # show plots
    if display_plot:
        io.imshow(img)
        plt.title('Original Image')
        io.show()

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

        io.imshow(img_binary)
        plt.title('Binary Image')
        io.show()

        io.imshow(img_label)
        plt.title('Labeled Image')
        io.show()

    regions = regionprops(img_label)
    # find the threshold used to remove the small noise
    thre_noise = {'height':[10.0, 80.0], 'width':[12.0, 85.0]}
    io.imshow(img_binary)
    ax = plt.gca()
    for props in regions:
        # coordinate of the pixels
        minr, minc, maxr, maxc = props.bbox

        # use if to remove too small or too large region
        if (maxr - minr < thre_noise['height'][0] or maxc - minc < thre_noise['width'][0]
        or maxr - minr > thre_noise['height'][1] or maxc - minc > thre_noise['width'][1]):
            continue

        if display_plot:
            ax.add_patch(Rectangle((minc, minr), maxc - minc, maxr - minr, fill=False,
                        edgecolor='red', linewidth=1))
        #print minc, minr, maxc, maxr, maxr-minr, maxc-minc
        roi = img_binary[minr:maxr, minc:maxc]
        m = moments(roi)
        cr = m[0, 1] / m[0, 0]
        cc = m[1, 0] / m[0, 0]
        mu = moments_central(roi, center=(cr, cc))
        nu = moments_normalized(mu)
        hu = moments_hu(nu)

        Features.append(hu)
        coordinate.append([minr, minc, maxr, maxc])

    if display_plot:
        plt.title('Bounding Boxes')
        io.show()

    result['Features'] = Features
    result['Coordinate'] = coordinate

    return result


def normalized(features, mean, std):
    # normalization
    feature_array = np.array(features)
    norm_features = []
```

```
    for i in range(feature_array.shape[0]):
        feature_array[i] -= mean
        feature_array[i] /= std
        norm_features.append(feature_array[i])

    return norm_features


def predict(test_filename, file_path, display_plot):
    train_features, Label, mean, std = getFeatures(file_path, display_plot)
    dataset = testFeatures(test_filename, display_plot)

    # normalization for test data features
    test_features = normalized(dataset['Features'], mean, std)

    D = cdist(test_features, train_features)
    if display_plot:
        io.imshow(D)
        plt.title('Distance Matrix')
        io.show()

    D_index = np.argsort(D, axis=1)
    Ypred = [Label[i[0]] for i in D_index]
    if display_plot:
        TestBound(dataset['img_binary'], Ypred, dataset['Coordinate'])

    return Ypred, dataset['Coordinate'], D


def TestBound(img_binary, Ypred, Coordinate):

    io.imshow(img_binary)
    ax = plt.gca()
    for i in range(len(Ypred)):
        y = Ypred[i]
        c = Coordinate[i]
        minr, minc, maxr, maxc = c[0], c[1], c[2], c[3]

        ax.add_patch(Rectangle((minc, minr), maxc - minc, maxr - minr, fill=False,
                    edgecolor='red', linewidth=1))
        plt.text(maxc, minr, y, bbox=dict(facecolor='red', alpha=0.5))
    plt.title('Bounding Boxes for test image')
    io.show()


def score(testName, Ypred, Coordinate):
    pkl_file = open(file_path+testName, 'rb')
    #print(pickle.load(open(file_path+testName, 'rb')))
    mydict = pickle.load(pkl_file)
    pkl_file.close()
    classes = mydict[b'classes']
    location = mydict[b'locations']


    num_correct = 0.0
    num_total = len(classes)
    for i in range(num_total):
        for j in range(len(Coordinate)):
            cenc = location[i][0]
            cenr = location[i][1]
            minr, minc = Coordinate[j][0], Coordinate[j][1]
            maxr, maxc = Coordinate[j][2], Coordinate[j][3]
            if cenr < minr or cenr > maxr or cenc < minc or cenc > maxc:
                continue

            if classes[i] == Ypred[j]:
                num_correct += 1
                break
    accuracy_rate = num_correct / num_total

    return accuracy_rate


if __name__ == "__main__":
    #file_path = sys.argv[1]
    file_path='/content/images/'
```

```
    display_plot = True
    test_1 = file_path + 'test.bmp'
    test_result = 'test_gt_py3.pkl'

    Ypred, Coordinate, D = predict(test_1, file_path, display_plot)

    accuracy = score(test_result, Ypred, Coordinate)
    print(accuracy)

```








```
0.5428571428571428
```