# Code

## train.py

#Takes console input and adds given label and file to features database

#database stored in file "db.dat"

import numpy as np

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 os

def train(symbol, img\_name, plot):

img = io.imread(img\_name)

if plot:

print img.shape

io.imshow(img)

plt.title('Original Image')

io.show()

if plot:

hist=exposure.histogram(img)

plt.bar(hist[1],hist[0])

plt.title('Histogram')

plt.show()

th=211

img\_binary= (img<th).astype(np.double)

#io.imsave(symbol+"\_binary.jpg", img\_binary, None, quality=100)

#"""

if plot:

io.imshow(img\_binary)

plt.title('Binary Image')

io.show()

#"""

img\_label = label(img\_binary,background=0)

if plot:

io.imshow(img\_label)

plt.title('Labeled Image')

io.show()

print np.amax(img\_label)

regions = regionprops(img\_label)

if plot:

io.imshow(img\_binary)

ax=plt.gca()

plt.title('Bounding Boxes')

db=[]

if not os.path.exists('db.dat'):

print('Creating database')

else:

with open('db.dat', 'rb') as rfp:

db=pickle.load(rfp)

Features = []

for props in regions:

minr, minc, maxr, maxc = props.bbox

area=(maxr-minr)\*(maxc-minc)

if area<150 or maxr-minr<=9 or maxc-minc<=9 or area>10000:

continue

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,cr,cc)

nu=moments\_normalized(mu)

hu=moments\_hu(nu)

#find percentage of bb taken up by character

"""

white=0.0

for i in range(maxr-minr):

for j in range(maxc-minc):

if roi[i][j]==1:

white+=1

white=white/area

hu=np.append(hu,[white])

"""

Features.append(hu)

if plot:

ax.add\_patch(Rectangle((minc,minr), maxc - minc, maxr - minr, fill=False, edgecolor='red', linewidth=1 ))

if plot:

io.show()

data=[symbol, Features]

db.append(data)

with open('db.dat', 'wb') as wfp:

pickle.dump(db,wfp)

char=raw\_input('Input character:')

while len(char)!=1 or not char.isalpha():

char=raw\_input('Input must be single alphabetic character:')

fname=raw\_input('Input training file:')

plot=raw\_input('Display plots? (y/n):')

while plot!='y' and plot!='n':

plot=raw\_input('Display plots? Please enter y/n:')

if plot=='y':

plot=True

else:

plot=False

train(char,fname,plot)

print("Training data saved to db.dat")

## recognize.py

#Normalizes features in "db.dat" and creates distance and confusion matrices

#called after train.py

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 math

import os

def char\_value(char\_list, value):

for i in range(len(char\_list)):

if value<=char[i][1]:

return char[i][0]

return 0

def normalize(fname, db):

count=0

total=0

for i in range(len(db)): #characters

for j in range(len(db[i][1])): #regions

for k in range (7): #features

total+=db[i][1][j][k]

count+=1

mean=total/count

ndb=db[:]

total=0

for i in range(len(db)): #characters

for j in range(len(db[i][1])): #regions

for k in range (7): #features

ndb[i][1][j][k]= ndb[i][1][j][k]-mean

total+=math.pow(ndb[i][1][j][k],2)

std=math.sqrt(total/count)

for i in range(len(db)): #characters

for j in range(len(db[i][1])): #regions

for k in range (7): #features

ndb[i][1][j][k]= ndb[i][1][j][k]/std

#ndb[i][1][j][7]=ndb[i][1][j][7]/100.0

ndb.append([mean,std])

with open(fname, 'wb') as wfp:

pickle.dump(ndb,wfp)

#"""

with open('db.dat', 'rb') as rfp:

db=pickle.load(rfp)

normalize('ndb.dat',db)

#"""

with open('ndb.dat', 'rb') as rfp:

ndb=pickle.load(rfp)

#create 2D feature matrix

Features=[]

for i in range(len(ndb)-1): #characters

for j in range(len(ndb[i][1])): #regions

Features.append(ndb[i][1][j])

#get absolute character range index for Features

char=[]

total=0

for i in range(len(ndb)-1):

total+=len(ndb[i][1])

if i==0:

total=total-1

char\_index=[ndb[i][0],total]

char.append(char\_index)

D=cdist(Features,Features)

io.imshow(D)

plt.title('Distance Matrix')

io.show()

#find second smallest index in each row of D without sort

D\_index=[]

for i in range(len(D)):

min=1000

index=-1

for j in range(len(D[i])):

if i!=j and D[i][j]<min:

min=D[i][j]

index=j

D\_index.append(index)

Ytrue = []

Ypred = []

count=0.0

for i in range(len(D\_index)):

Ytrue.append(char\_value(char,i))

Ypred.append(char\_value(char,D\_index[i]))

confM=confusion\_matrix(Ytrue, Ypred)

io.imshow(confM)

plt.title('Confusion Matrix')

io.show()

## test.py

#extracts features from "test.bmp", normalizes based on mean and standard deviation stored in nbd.dat

#creates distance matrix between test features and training data features

#called after recognize.py

import numpy as np

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 math

import os

def extract(img\_name, plot):

img = io.imread(img\_name)

if plot:

print img.shape

io.imshow(img)

plt.title('Original Image')

io.show()

if plot:

hist=exposure.histogram(img)

plt.bar(hist[1],hist[0])

plt.title('Histogram')

plt.show()

th=211

img\_binary= (img<th).astype(np.double)

#"""

if plot:

io.imshow(img\_binary)

plt.title('Binary Image')

io.show()

#"""

img\_label = label(img\_binary,background=0)

if plot:

io.imshow(img\_label)

plt.title('Labeled Image')

io.show()

print np.amax(img\_label)

regions = regionprops(img\_label)

if plot:

io.imshow(img\_binary)

ax=plt.gca()

plt.title('Bounding Boxes')

Features = []

for props in regions:

minr, minc, maxr, maxc = props.bbox

area=(maxr-minr)\*(maxc-minc)

if area<150 or maxr-minr<=9 or maxc-minc<=9 or area>10000:

continue

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,cr,cc)

nu=moments\_normalized(mu)

hu=moments\_hu(nu)

"""

white=0.0

for i in range(maxr-minr):

for j in range(maxc-minc):

if roi[i][j]==1:

white+=1

white=white/area

hu=np.append(hu,[white])

"""

Features.append(hu)

if plot:

ax.add\_patch(Rectangle((minc,minr), maxc - minc, maxr - minr, fill=False, edgecolor='red', linewidth=1 ))

if plot:

io.show()

return Features

def char\_value(char\_list, value):

for i in range(len(char\_list)):

if value<=char[i][1]:

return char[i][0]

return 0

def normalize(Features, mean, std):

std=std\*1.0

for i in range(len(Features)): #characters

for j in range (7): #features

Features[i][j]=(Features[i][j]-mean)/std

#ndb[i][1][j][7]=ndb[i][1][j][7]/100.0

Test\_Features=extract('test1.bmp',True)

with open('ndb.dat', 'rb') as rfp:

ndb=pickle.load(rfp)

normalize(Test\_Features,ndb[len(ndb)-1][0],ndb[len(ndb)-1][0])

#create 2D feature matrix

Features=[]

for i in range(len(ndb)-1): #characters

for j in range(len(ndb[i][1])): #regions

Features.append(ndb[i][1][j])

char=[]

total=0

for i in range(len(ndb)-1):

total+=len(ndb[i][1])

if i==0:

total=total-1

char\_index=[ndb[i][0],total]

char.append(char\_index)

D=cdist(Test\_Features,Features)

io.imshow(D)

plt.title('Distance Matrix')

io.show()

#find smallest index in each row of D without sort

D\_index=[]

for i in range(len(D)):

min=1000

index=-1

for j in range(len(D[i])):

if D[i][j]<min:

min=D[i][j]

index=j

D\_index.append(index)

Ytrue = []

Ypred = []

for i in range(len(D\_index)):

# Ytrue.append(char\_value(char,i))

Ypred.append(char\_value(char,D\_index[i]))

#print(D\_index[i]),

#print(Ypred[i]),

## RunMyOCRRecognition.py

#Does everything

#takes test file via console input

#Overwrites any current db.dat or ndb.dat

#Creates these files similar to the methods used in train and recognize with the training files a.bmp, d.bmp...

#Attempts ocr with training data on the test file

#Prints all plots including the final bounding box image with accompanying ocr results

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 math

#sort that puts in order from top to bottom, left to right

def sort(Features):

sortList = []

unsort=Features[:]

for i in range(len(unsort)):

minc=100000

minr=100000

index=-1

for j in range (len(unsort)):

if unsort[j][8]<minr and abs(unsort[j][8]-minr)>30:

minr=unsort[j][8]

minc=unsort[j][7]

index=j

else:

if abs(unsort[j][8]-minr)<10 and unsort[j][7]<minc:

minr=unsort[j][8]

minc=unsort[j][7]

index=j

sortList.append(unsort.pop(index))

return sortList

def train(symbol, img\_name, plot,db=None):

img = io.imread(img\_name)

if plot:

print img.shape

io.imshow(img)

plt.title('Original Image')

io.show()

if plot:

hist=exposure.histogram(img)

plt.bar(hist[1],hist[0])

plt.title('Histogram')

plt.show()

th=211

img\_binary= (img<th).astype(np.double)

#io.imsave(symbol+"\_binary.jpg", img\_binary, None, quality=100)

#img\_binary=io.imread('test a binary fix1.bmp')

#"""

if plot:

io.imshow(img\_binary)

plt.title('Binary Image')

io.show()

#"""

img\_label = label(img\_binary,background=0)

if plot:

io.imshow(img\_label)

plt.title('Labeled Image')

io.show()

print np.amax(img\_label)

regions = regionprops(img\_label)

if plot:

io.imshow(img\_binary)

ax=plt.gca()

plt.title('Bounding Boxes')

if not db:

db = []

Features = []

for props in regions:

minr, minc, maxr, maxc = props.bbox

area=(maxr-minr)\*(maxc-minc)

if area<150 or maxr-minr<=9 or maxc-minc<=9 or area>10000:

continue

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,cr,cc)

nu=moments\_normalized(mu)

hu=moments\_hu(nu)

"""

white=0.0

for i in range(maxr-minr):

for j in range(maxc-minc):

if roi[i][j]==1:

white+=1

white=white/area

hu=np.append(hu,[white])

"""

Features.append(hu)

if plot:

ax.add\_patch(Rectangle((minc,minr), maxc - minc, maxr - minr, fill=False, edgecolor='red', linewidth=1 ))

if plot:

plt.savefig(symbol+'\_component\_bb.png', bbox\_inches='tight', trasparent=False)

io.show()

#io.imsave(symbol+"\_component\_bb.jpg", img\_label, None, quality=100)

data=[symbol, Features]

db.append(data)

return db

def autotrain():

chars = ['d','m','n','o','p','q','r','u','w']

db=train('a','a.bmp',True)

for i in range(len(chars)):

db=train(chars[i],chars[i]+".bmp",True,db)

with open('db.dat', 'wb') as wfp:

pickle.dump(db,wfp)

def normalize(fname, db):

count=0

total=0

for i in range(len(db)): #characters

for j in range(len(db[i][1])): #regions

for k in range (7): #features

total+=db[i][1][j][k]

count+=1

mean=total/count

ndb=db[:]

total=0

for i in range(len(db)): #characters

for j in range(len(db[i][1])): #regions

for k in range (7): #features

ndb[i][1][j][k]= ndb[i][1][j][k]-mean

total+=math.pow(ndb[i][1][j][k],2)

std=math.sqrt(total/count)

for i in range(len(db)): #characters

for j in range(len(db[i][1])): #regions

for k in range (7): #features

ndb[i][1][j][k]= ndb[i][1][j][k]/std

#ndb[i][1][j][7]=ndb[i][1][j][7]/1.0

ndb.append([mean,std])

with open(fname, 'wb') as wfp:

pickle.dump(ndb,wfp)

def extract(img\_name, plot):

img = io.imread(img\_name)

if plot:

print img.shape

io.imshow(img)

plt.title('Original Image')

io.show()

if plot:

hist=exposure.histogram(img)

plt.bar(hist[1],hist[0])

plt.title('Histogram')

plt.show()

th=211

img\_binary= (img<th).astype(np.double)

#io.imsave(symbol+"\_binary.jpg", img\_binary, None, quality=100)

#"""

if plot:

io.imshow(img\_binary)

plt.title('Binary Image')

io.show()

#"""

img\_label = label(img\_binary,background=0)

if plot:

io.imshow(img\_binary)

plt.title('Labeled Image')

io.show()

print np.amax(img\_label)

regions = regionprops(img\_label)

if plot:

io.imshow(img\_binary)

ax=plt.gca()

plt.title('Bounding Boxes')

Features = []

for props in regions:

minr, minc, maxr, maxc = props.bbox

area=(maxr-minr)\*(maxc-minc)

if area<150 or maxr-minr<=9 or maxc-minc<=9 or area>10000:

continue

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,cr,cc)

nu=moments\_normalized(mu)

hu=moments\_hu(nu)

"""

white=0.0

for i in range(maxr-minr):

for j in range(maxc-minc):

if roi[i][j]==1:

white+=1

white=white/area

hu=np.append(hu,[white])

"""

hu=np.append(hu,[cc+minc,cr+minr])

Features.append(hu)

if plot:

ax.add\_patch(Rectangle((minc,minr), maxc - minc, maxr - minr, fill=False, edgecolor='red', linewidth=1 ))

if plot:

#io.imsave("test\_binary.jpg", img\_label, None, quality=100)

#plt.savefig('test\_component\_bb.png', bbox\_inches='tight', trasparent=False)

io.show()

return Features

def char\_value(char\_list, value):

for i in range(len(char\_list)):

if value<=char\_list[i][1]:

return char\_list[i][0]

return 0

def normalizeGiven(Features, mean, std):

std=std\*1.0

for i in range(len(Features)): #characters

for j in range (7): #features

Features[i][j]=(Features[i][j]-mean)/std

#Features[i][7]=Features[i][7]/1.0

def ocr(fname):

#train and normalize with training files a.bmp ...

autotrain()

with open('db.dat', 'rb') as rfp:

db=pickle.load(rfp)

normalize('ndb.dat',db)

#get features from given file

CTest\_Features=extract(fname,True)

CTest\_Features=sort(CTest\_Features)

with open('ndb.dat', 'rb') as rfp:

ndb=pickle.load(rfp)

Features=[]

for i in range(len(ndb)-1): #characters

for j in range(len(ndb[i][1])): #regions

Features.append(ndb[i][1][j])

char=[]

total=0

for i in range(len(ndb)-1):

total+=len(ndb[i][1])

if i==0:

total=total-1

char\_index=[ndb[i][0],total]

char.append(char\_index)

Test\_Features=[]

for i in range(len(CTest\_Features)):

Test\_Features.append(CTest\_Features[i][0:7])

normalizeGiven(Test\_Features,ndb[len(ndb)-1][0],ndb[len(ndb)-1][0])

D=cdist(Test\_Features,Features)

io.imshow(D)

plt.title('Distance Matrix')

#plt.savefig('Distance\_Matrix.png', bbox\_inches='tight', trasparent=False)

io.show()

#find 5 smallest indeces in each row of D without sort and store most common

D\_index=[]

for i in range(len(D)):

min=1000

index=-1

for j in range(len(D[i])):

if D[i][j]<min:

min=D[i][j]

index=j

D\_index.append(index)

"""

for i in range(len(D)):

min=[1000,1000,1000,1000,1000]

index=[-1,-1,-1,-1,-1]

for j in range(len(D[i])):

if D[i][j]<min[0]:

min[4]=min[3]

min[3]=min[2]

min[2]=min[1]

min[1]=min[0]

min[0]=D[i][j]

index[4]=index[3]

index[3]=index[2]

index[2]=index[1]

index[1]=index[0]

index[0]=j

else:

if D[i][j]<min[1]:

min[4]=min[3]

min[3]=min[2]

min[2]=min[1]

min[1]=D[i][j]

index[4]=index[3]

index[3]=index[2]

index[2]=index[1]

index[1]=j

else:

if D[i][j]<min[2]:

min[4]=min[3]

min[3]=min[2]

min[2]=D[i][j]

index[4]=index[3]

index[3]=index[2]

index[2]=j

else:

if D[i][j]<min[3]:

min[4]=min[3]

min[3]=D[i][j]

index[4]=index[3]

index[3]=j

else:

if D[i][j]<min[4]:

min[4]=D[i][j]

index[4]=j

#end inner loop

for k in range(5):

index[k]=char\_value(char,index[k])

#find mode (weight on distance, no sort)

freq = []

for k in range(5):

if index[k] not in freq:

count = 0

for j in range(5):

if index[j]==index[k]:

count+=1

freq.append([index[k],count])

#find max frequency

max=0

for k in range(len(freq)):

if freq[k][1]>max:

max=freq[k][1]

label=freq[k][0]

D\_index.append(label)

#print(D\_index[i]),

#"""

pkl\_file=open('test1\_gt.pkl.txt', 'rb')

mydict=pickle.load(pkl\_file)

pkl\_file.close()

classes = mydict['classes']

locations = mydict['locations']

#char\_value(D\_index[i]) returns detected character such that i is the training label Features[i]

#CTest\_Features[i][7] contains 2\*1 list with [meanc, meanr] representing center of the same label Features[i]

total=0

Feature\_Center = []

for i in range(len(CTest\_Features)):

Feature\_Center.append(CTest\_Features[i][7:9])

D=cdist(locations,Feature\_Center)

Results=[]

for i in range (len(locations)):

for c in range (len(locations[i])):

min=10000

index=-1

for j in range(len(D\_index)):

if D[i][j]<min:

min=D[i][j]

index=j

Results.append([classes[i],char\_value(char,D\_index[index])])

"""

print("Expected "+classes[i]),

print("Detected"),

print(char\_value(char,D\_index[index])),

print(D\_index[index])

"""

return Results

fname=raw\_input('Input testing file:')

Results=ocr(fname)

img = io.imread(fname)

th=211

img\_binary= (img<th).astype(np.double)

img\_label = label(img\_binary,background=0)

io.imshow(img\_binary)

ax=plt.gca()

plt.title('Bounding Boxes')

regions = regionprops(img\_label)

for props in regions:

minr, minc, maxr, maxc = props.bbox

ax.add\_patch(Rectangle((minc,minr), maxc - minc, maxr - minr, fill=False, edgecolor='red', linewidth=1 ))

io.show()

print("From top left to bottom right")

for i in range(len(Results)):

print("Expected "),

print(Results[i][0]),

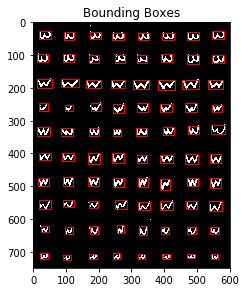
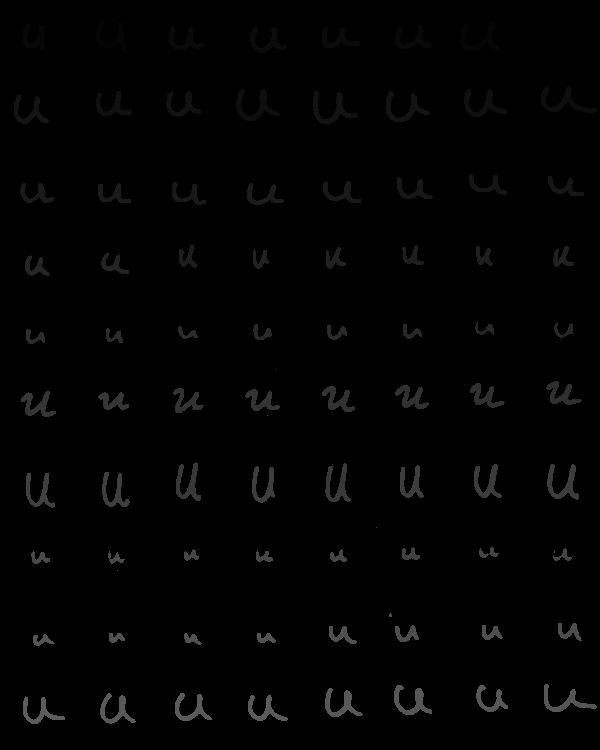
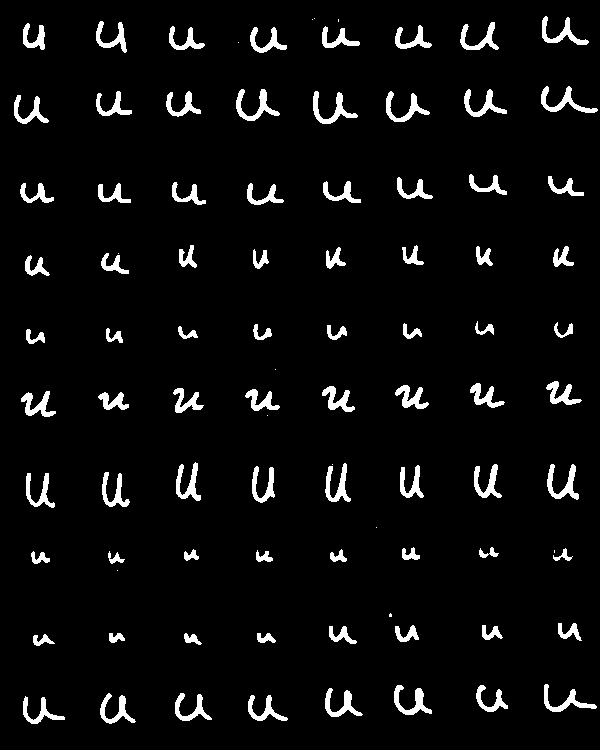
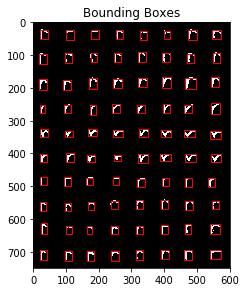
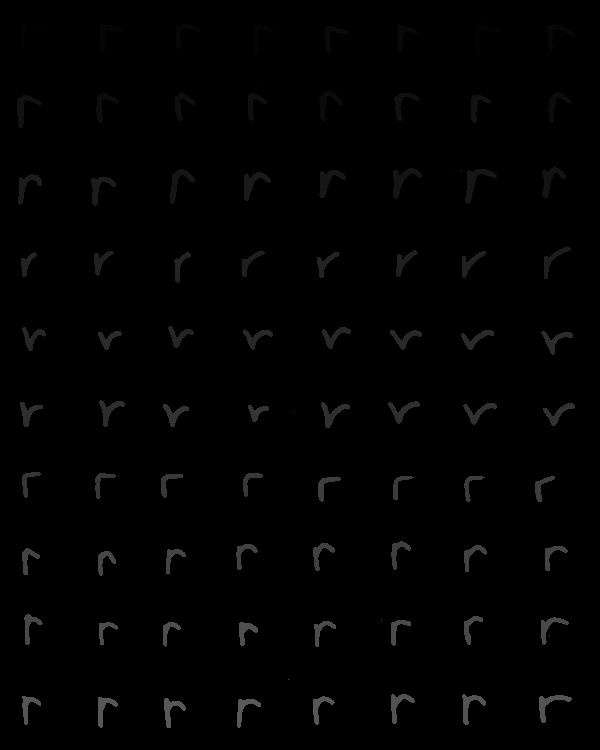
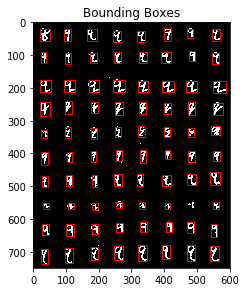
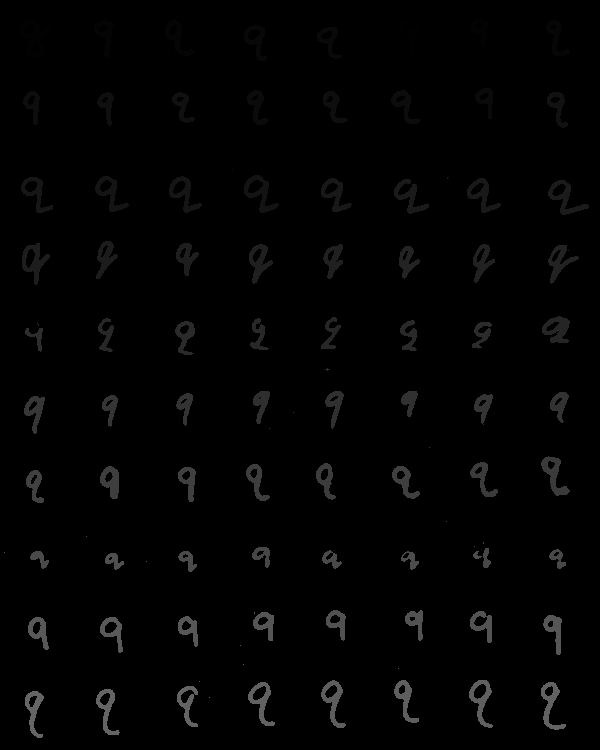
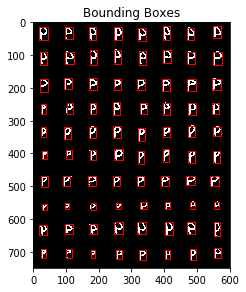
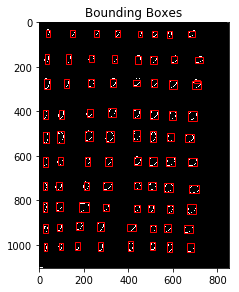
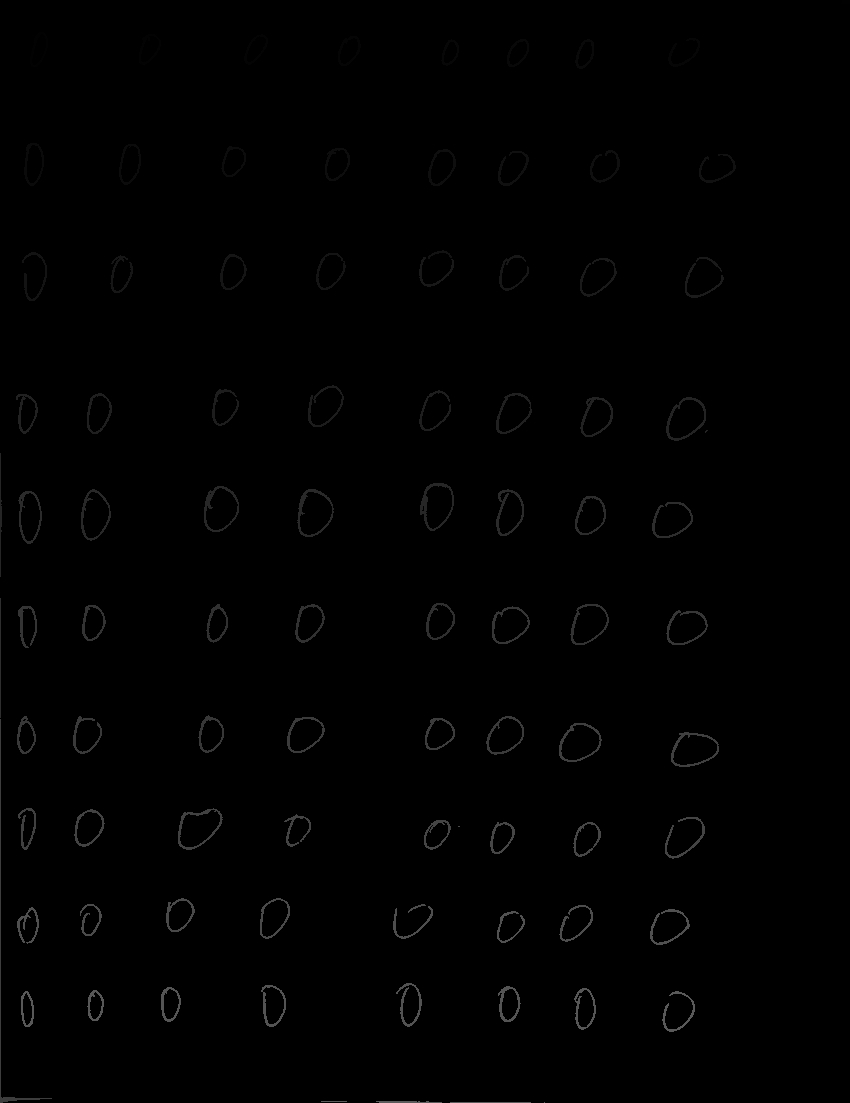
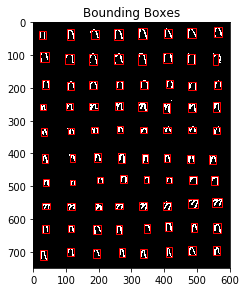
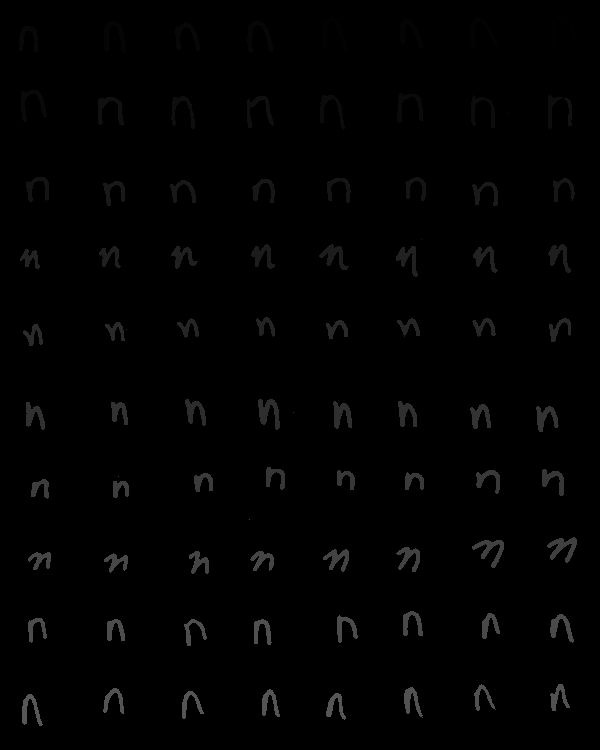
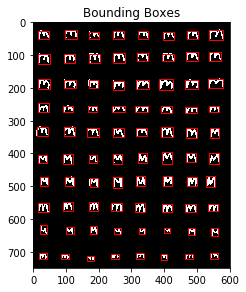
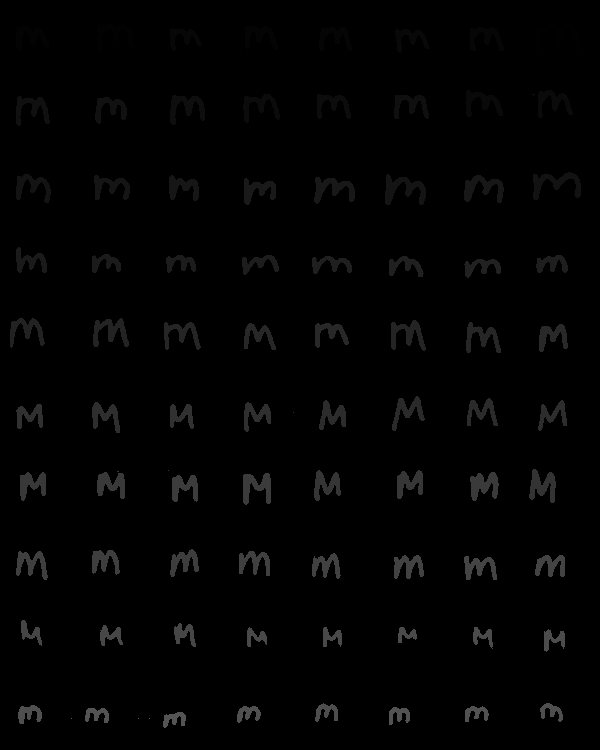
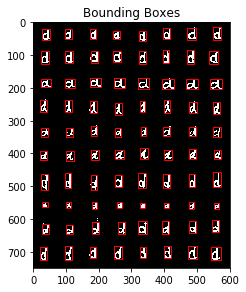
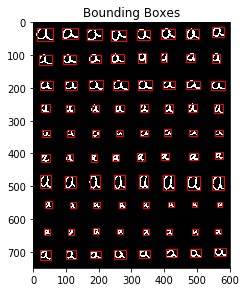
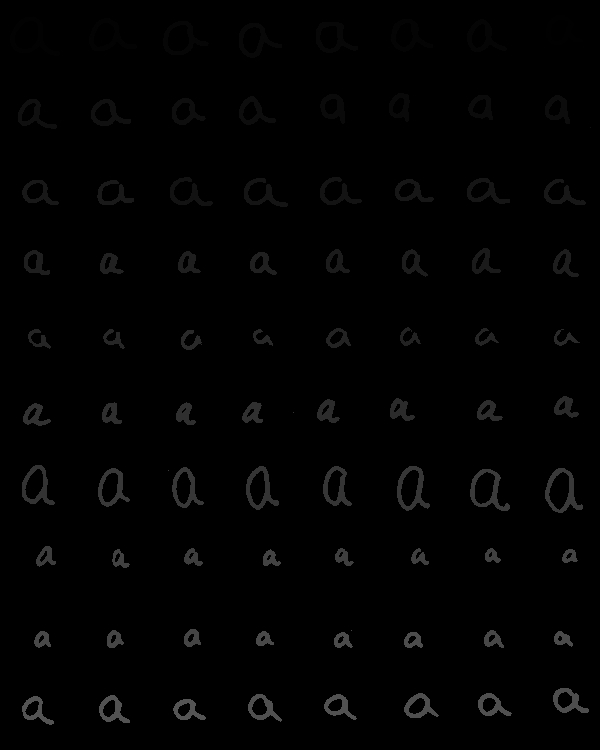
print("Detected"),

print(Results[i][1])

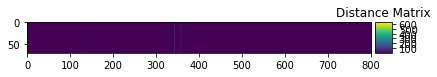
# Images

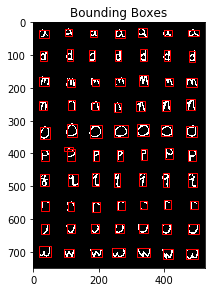
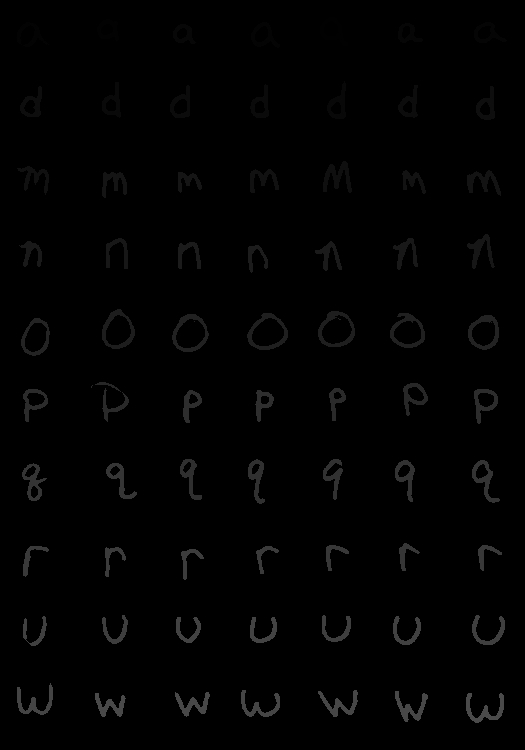
## Training

Plots with bounding boxes not full quality. Preceding connected component images show test images at full size.



## Recognition





Enhancements did not alter recognition

# Results

## Values

The static binary image threshold picked was 211.

71 components were obtained from the test image

Base recognition rate: 15.7%

Recognition rate picking out of 5 closest

## Features/experiments

Two enhancements were tested but not implemented in the final code.

**Fill percentage:**

An 8th feature was added to the database during extraction. This was the percentage of pixels in the ROI that were foreground. This was divided by several scalars during normalization but no matter what the result heavily skewed to certain values (especially ‘o’).

**5 nearest neighbors:**

Mode of the 5 least distant feature vectors considered instead. Results were similar to the base recognition. This implies that there is a greater flaw during recognition/normalization.

## Other Features

**Sort:**

Features from the test image are not stored in a useful order during extraction. This function sorted them with greater precedence to which “row” it is from before sorting by column. Doing this allows for simple comparison between the groundtruth locations matrix, the classes matrix, and the features found from test.

**autotrain:**

Runs an altered version of train.py that calls each training file automatically. If more training files were used the symbol must be added to the list parameter chars that is given to autotrain