Homework 1
Computer Vision

Xiaoli He

March 2, 2017

Table 1: Recognition rates for all exploration

Enhancement	recognition rate
basic	[0.2,0.5]
thresholding(Otsu's)	[0.31, 0.45]
equating contrast	[0.07,0.25]
median filter	[0.31,0.45]
closing, skeletonize and dilation	[0.38,0.36]
KNN (k=3)	[0.17,0.55]
combined	[0.43,0.46]

1 Results without any improvement

We tried on different thresholds for binarizing image and bounding box. In the end, we used 200 for binarizing image and 10 for the bounding box.
The recognition rates for test sets are summarized in Table1

1.1 Training sets

The accuracy for training data is 0.473643. You can see from the confusion matrix in Figure 2 that the recognition accuracy is not high.
Figure 1 below shows the connected component image with bounding boxes.

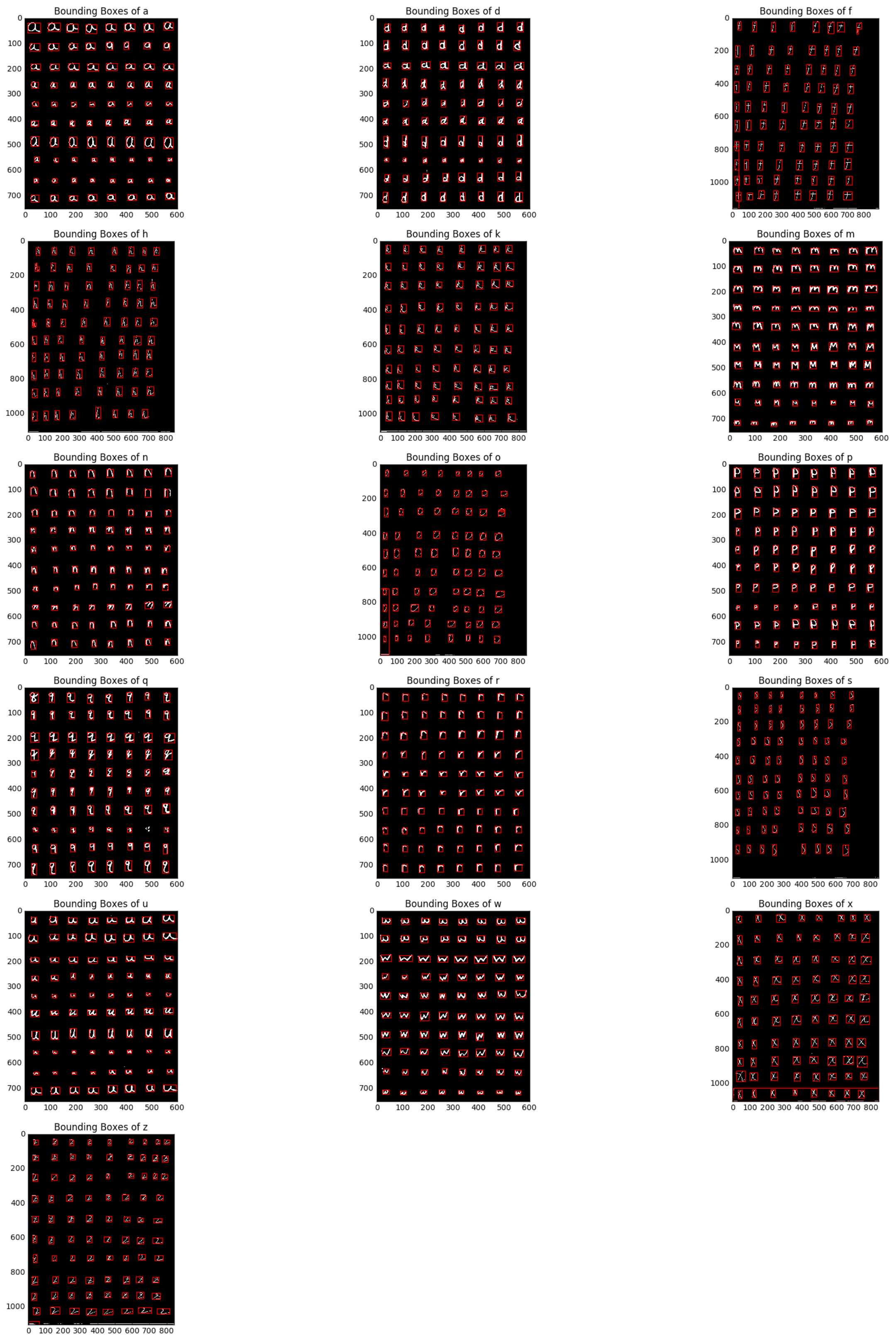


Figure 1: Training data: connected component image with bounding boxes

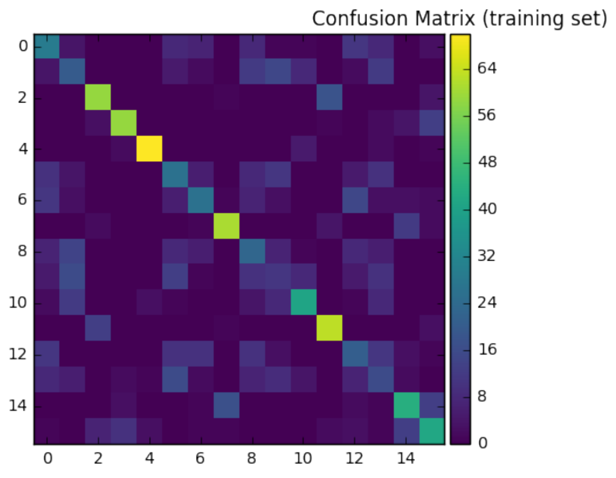


Figure 2: Training data: confusion matrix

1.2 Recognition on test sets

The accuracy for the given test1 and test2 are 0.2, 0.5.The number of components gained on test1 and test2 are 70 and 80.
Figure 3 below show the connected component image with bounding boxes.Due to some outliers, the Distance matrix only show range of 3 STD from the mean.

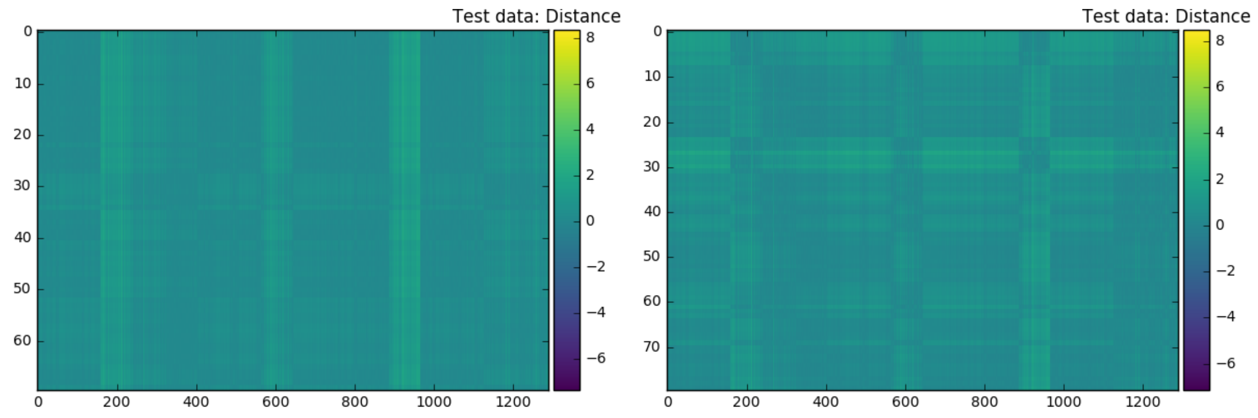
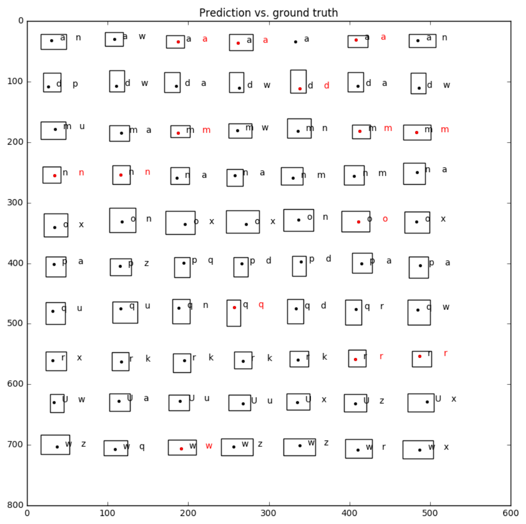


Figure 3: Test sets: connected component image with bounding boxes

Figure 4 shows predicted labels versus true labels. The dots are the true centroids for all characters. The square around them are the predicted bounding boxes. The two labels at each location are the ground truth and the prediction. If the prediction is correct, the dot and the predicted label is in red.



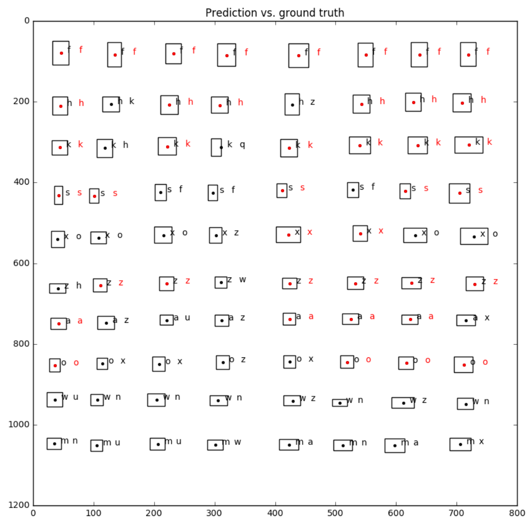


Figure 4: Test sets: prediction vs. ground truth

2 Improvements

We explored several methods to increase accuracy. Unfortunately, non of them have great improvements for both tests. Here we lists all the successful and unsuccessful attempts.

2.1 Automating thresholding(inconsistent effect)

We found that the fixed threshold for binarizing image works good for some examples but not the others. Therefore, we proposed that it makes more sense to automate thresholding.
We've tried the following methods in the *skimage* toolbox: adaptive method, ISODATA, Li's Minimum Cross Entropy, OTSU, Yen. However, the effects are different for test1 and test2: the recognition rates are [0.3142857142857143, 0.45]. It increases performance on test1, but not test2.
Figure 5 shows the recognition results by using Otsu's method.

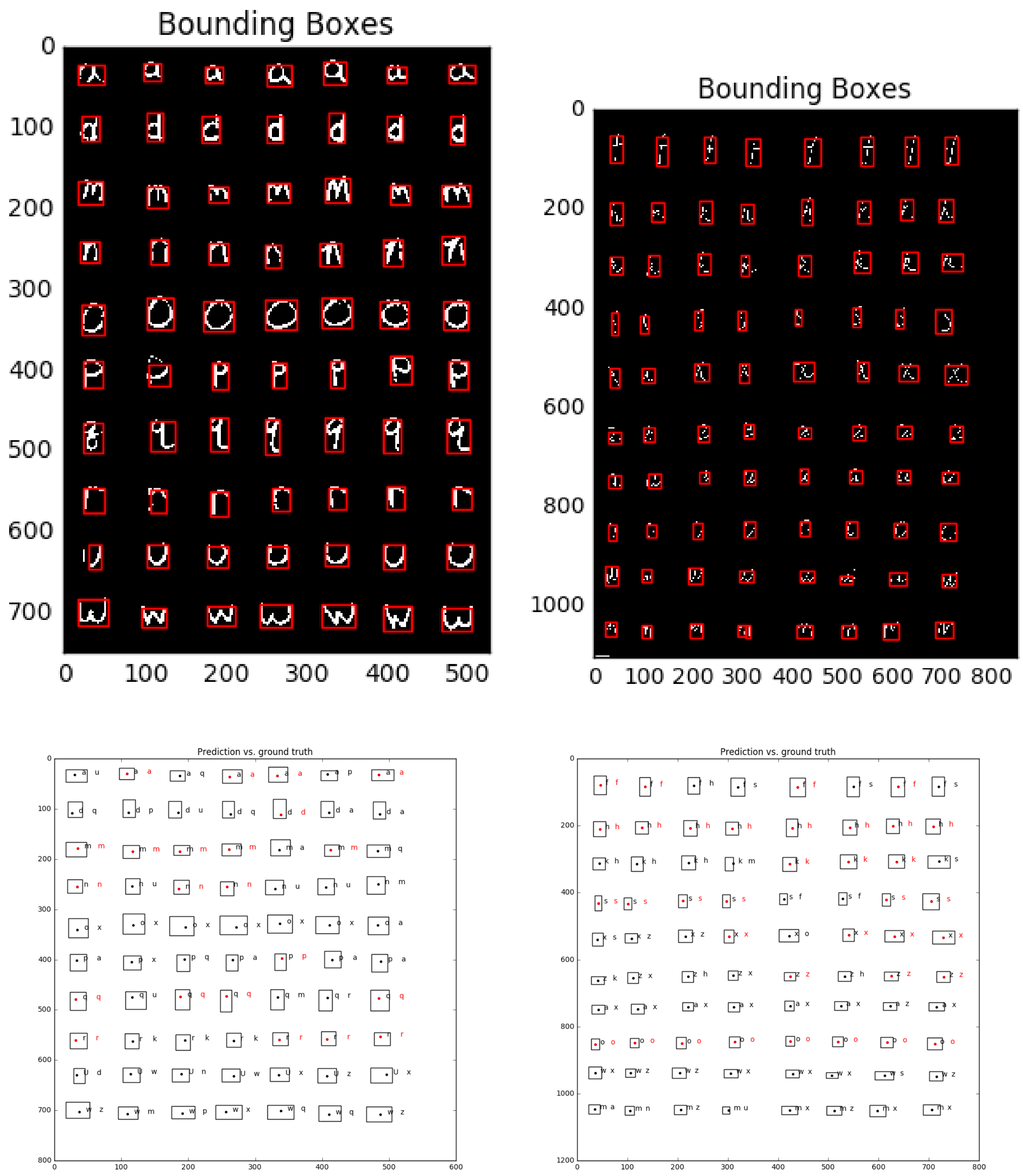


Figure 5: Test sets (using automated thresholding): connected component image with bounding boxes and its prediction vs. ground truth

2.2 Filters before thresholding(inconsistent effect)

We observed that for some examples, the binary image is more noisy (in terms of containing more unconnected edges), therefore, we tried two methods to enhance the image before thresholding. Because after these operations, the image range (was 0-255) will be changed, so the improvement reported here is the sum of enhancement and automated thresholding (Otsu's method).

1. **equating contrast** We used *equalize_adapthist* in *skimage.exposure* to equate the contrast. But the effect is negative. The recognition rates are [0.07142857142857142, 0.25].

2. **median filter** We also tried *median* in *skimage.filters*,. But the effect is quite different for test1 and test2. The recognition rates are [0.3142857142857143, 0.45]. Figure 6 shows the corresponding performance.

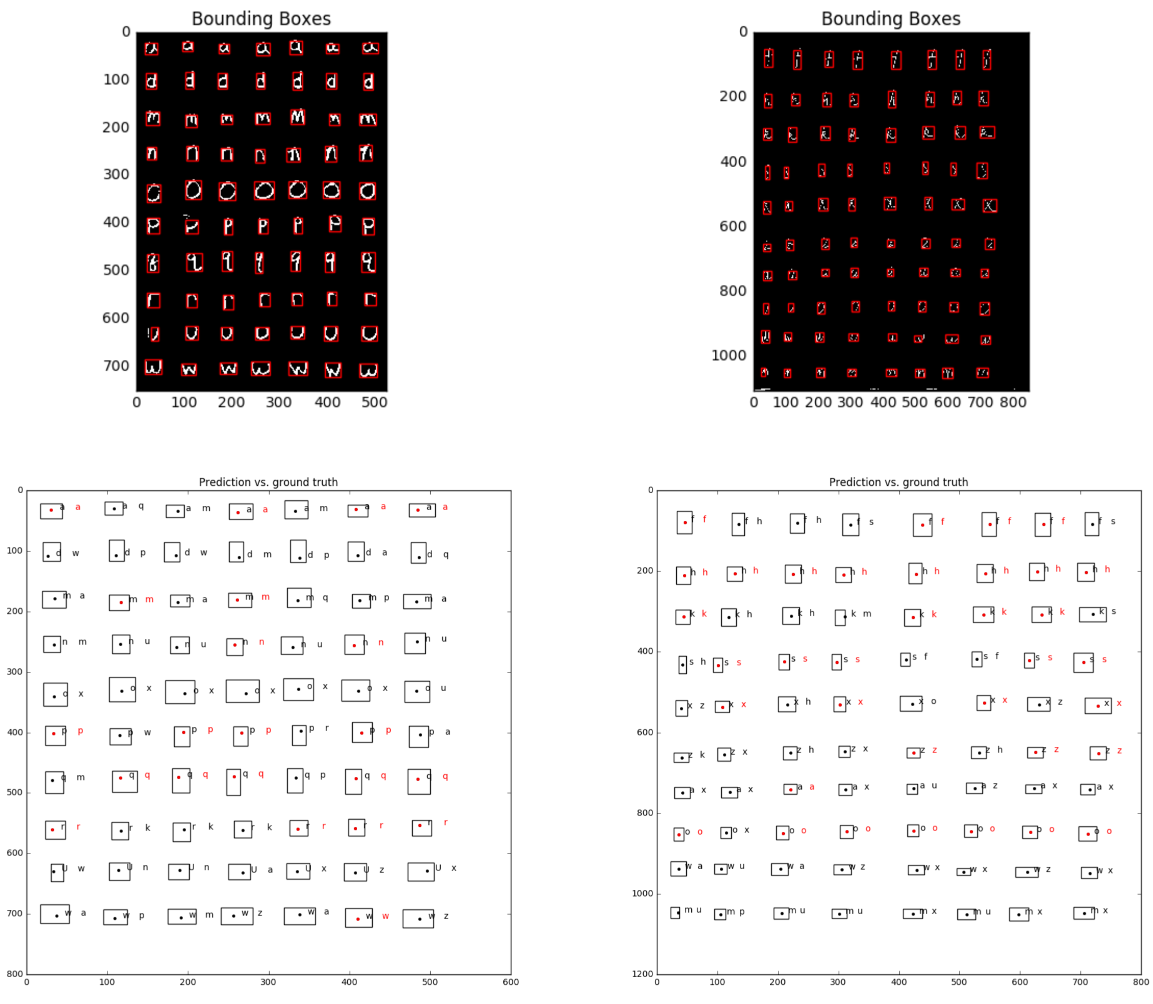


Figure 6: Test sets (using median filter before thresholding): connected component image with bounding boxes and its prediction vs. ground truth

2.3 Closing, skeletonize and Dilation on binary image(inconsistent effect)

We found that some examples are very thick, and others are too thin (so that it contains unconnected edges).To balance those two kinds of training examples, we first applied *closing* and *skeletonize* and then applied *binary_dilation* in module *morphology*. The recognition rates were [0.38571428571428573, 0.3625]. Though the rate was reduced for test2, from Figure 7, we can see that the letters in test2 are actually much more clear than before.

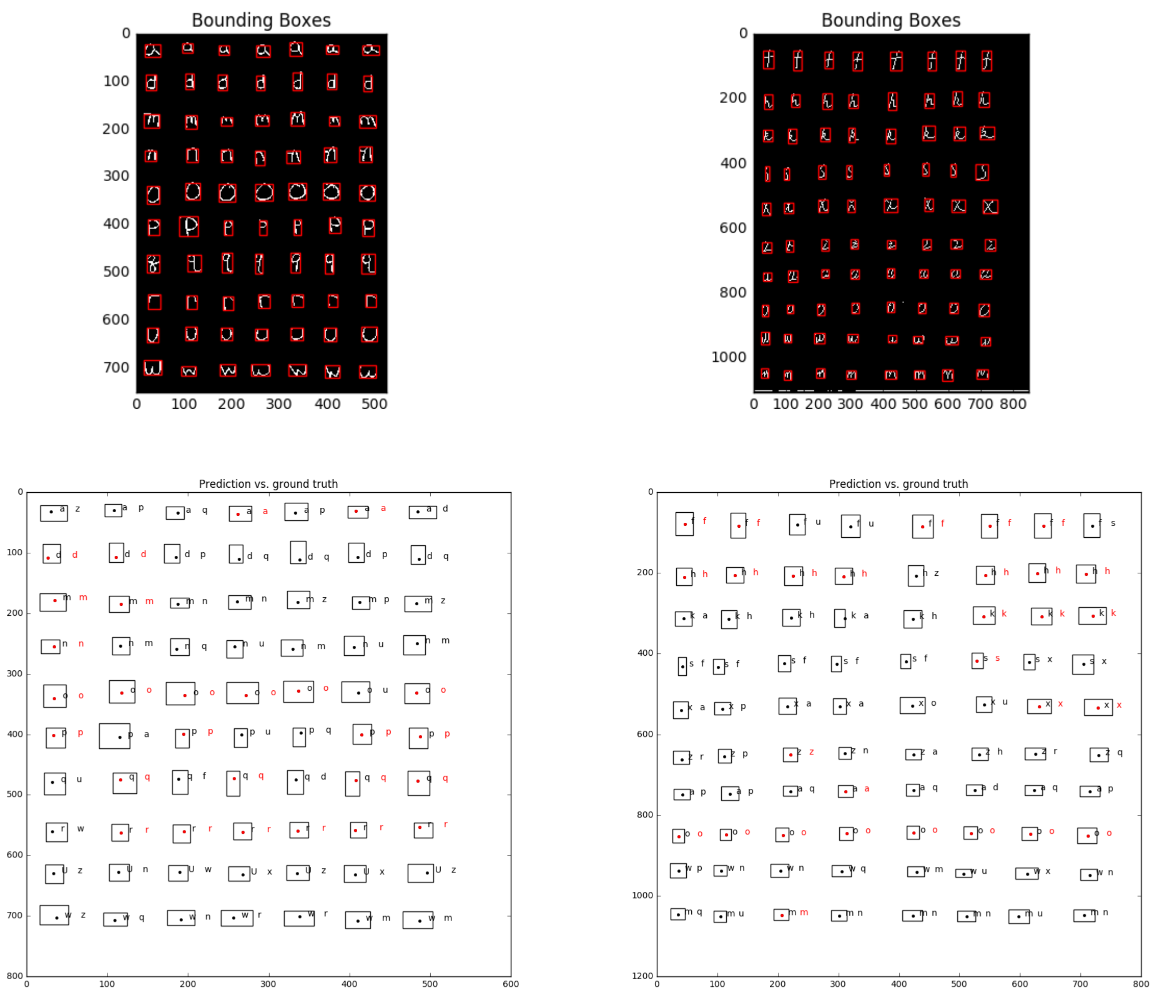


Figure 7: Test sets (using skeletonize and dilation before thresholding): connected component image with bounding boxes and its prediction vs. ground truth

2.4 k nearest neighbors classifier(inconsistent effect)

We found that in the training set, many letters are so similar to another (such as 'a' and 'd'). Therefore, instead of choosing the closest label as prediction, we tried to choose k = 3 nearest neighbors, and chose the mode among those candidates. Still, we used threshold = 200 for image binarization and 10 for bounding box. The recognition rates are [0.17142857142857143, 0.55]. It increases a little bit for test2, but not test1. Figure 8 show its results.

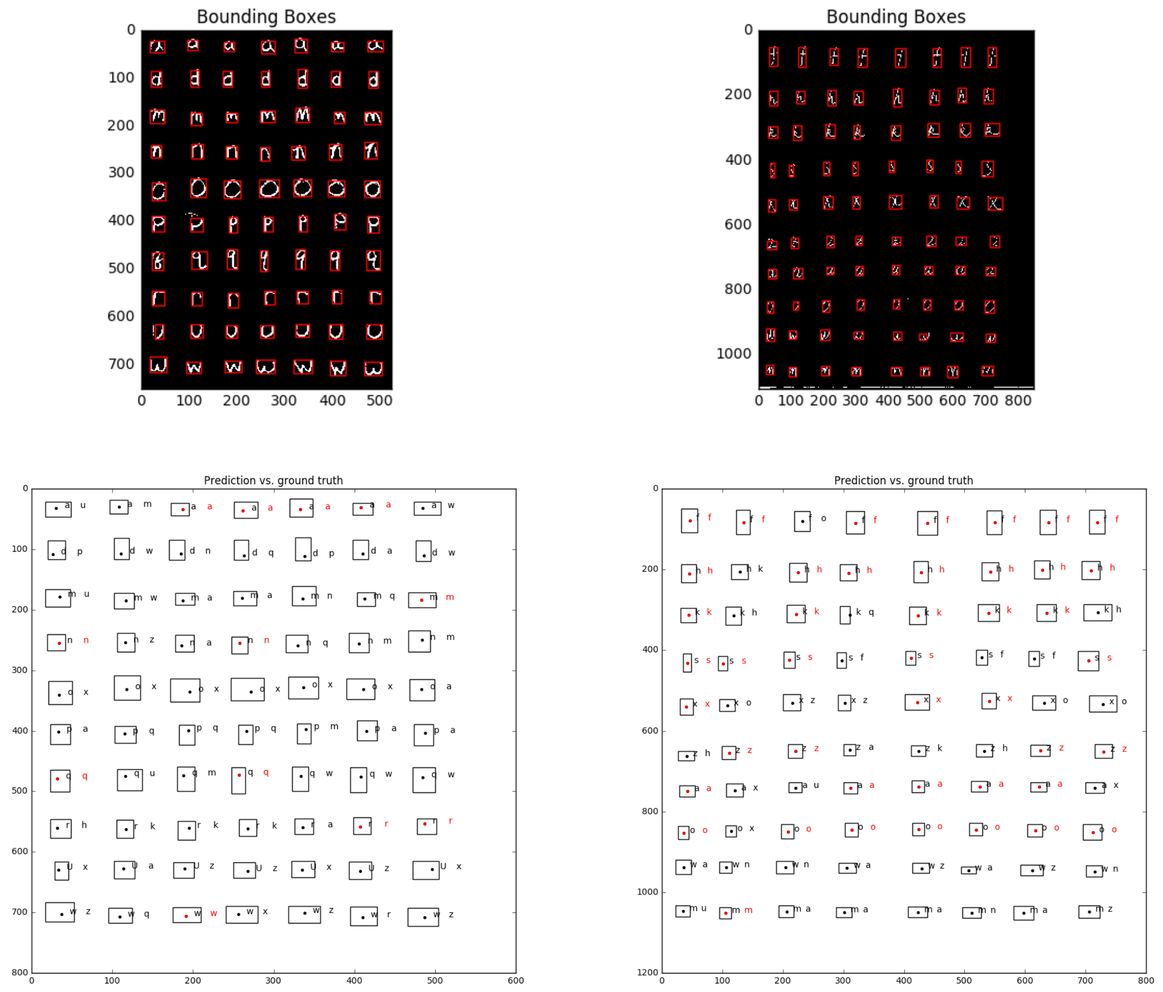


Figure 8: Test sets (using KNN classifier): connected component image with bounding boxes and its prediction vs. ground truth

2.5 All improvements combined

We combined median filter, Otsu's method thresholding, closing,skeletonize and dilation, and KNN classifiers together, and the recognition rates are: [0.42857142857142855, 0.4625].
In summary, the enhancement increases performance of test1 by almost 23%, but reduces performance of test2 by 4%.
Figure 9 shows corresponding results.

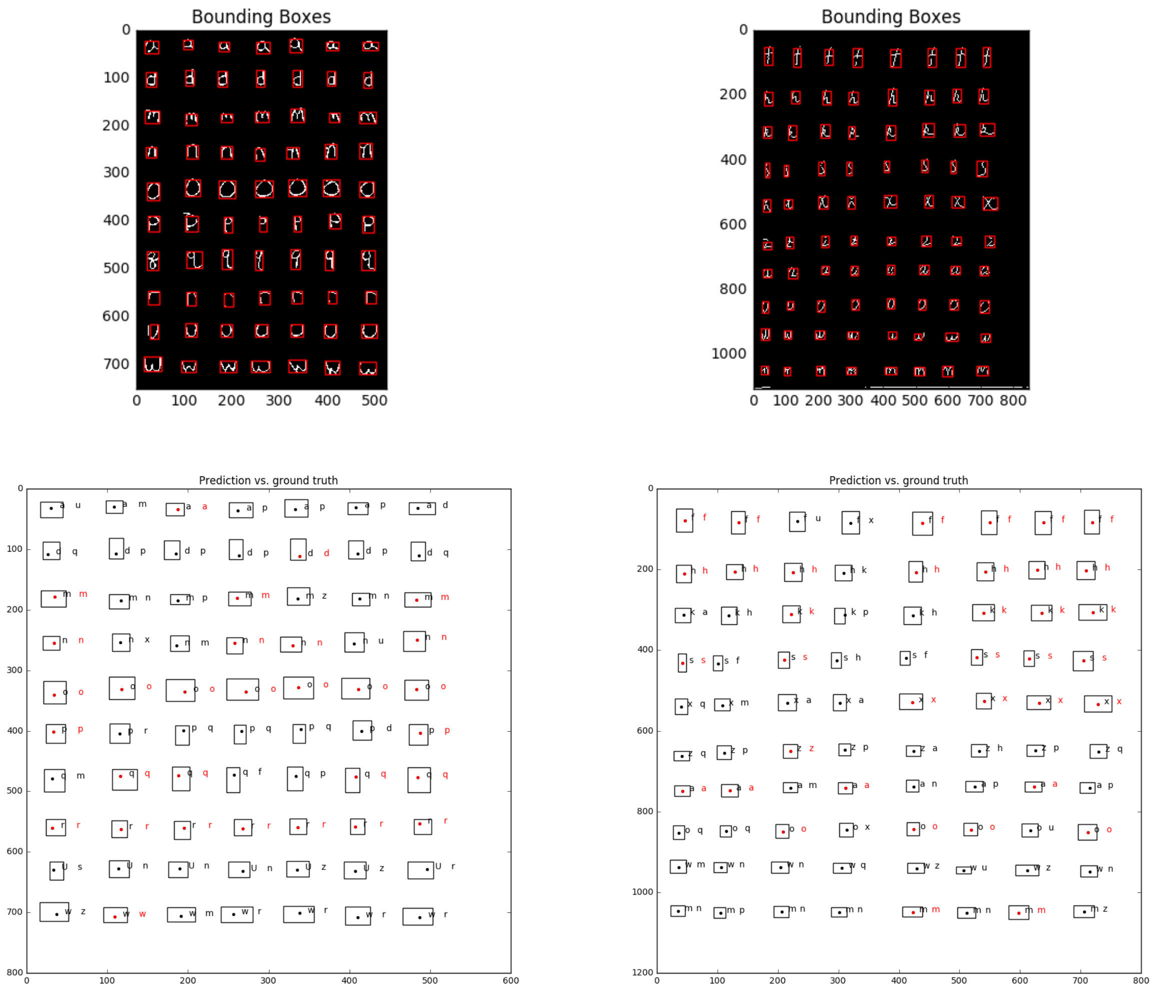


Figure 9: Test sets (using combined enhancement): connected component image with bounding boxes and its prediction vs. ground truth

Code files

Note: to run the codes, you might need to change the directory of:

1. train.py: (line 14, training image)
2. *test_xh.py*: (line 14, test image),(line 130,ground truth)
3. RunMyOCRRecogniton.py: (line 13: add current path to sys.path)

RunMyOCRRecognition:

```
def main_func(test_image, test_image_gt, display_idx = 1, improve_idx = 1, num_k = 15 ):
    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_norm
    from skimage import io, exposure
    import matplotlib.pyplot as plt
    from matplotlib.patches import Rectangle
    from matplotlib import path
    import pickle
    import os
    import sys
    sys.path.append("/Users/xiaolihe/Documents/Computer-Vision-534/hw1")
    import train
    import test_xh
    """
␣␣␣␣params:
␣␣␣␣␣␣␣␣test_image:␣list␣of␣file␣name␣of␣test␣images
␣␣␣␣␣␣␣␣test_image_gt:␣list␣of␣ground␣truth␣of␣the␣test␣images
␣␣␣␣␣␣␣␣display_idx:␣whether␣to␣display␣all␣figures
␣␣␣␣␣␣␣␣improve_idx:␣whether␣to␣use␣all␣the␣improvements
␣␣␣␣return:
␣␣␣␣␣␣␣␣Ypred_test1:␣predicted␣labels
␣␣␣␣␣␣␣␣features_test1:
␣␣␣␣␣␣␣␣regions1:
␣␣␣␣␣␣␣␣accuracy:␣accuracy␣of␣predictions
␣␣␣␣"""
    characters = ['a','d','f','h','k','m','n','o','p','q','r','s','u','w','x','z']
    Features = []
    class_labels = []
    threshold_binary = 200 ## before any improvement
    threshold_size = 10
    print threshold_size
     ## for knn
    # display_idx = 1
    # improve_idx = 1
    for c in characters:
        tmp_feature, tmp_class_label = train.feature_database(c, threshold_binary, threshold_s
        [Features.append(tf) for tf in tmp_feature]
        [class_labels.append(tf) for tf in tmp_class_label]
```

```
# ### Normalization
# get features (exclude labels)
mean_features =  np.asarray(np.mean(Features,0))
std_features = np.asarray(np.std(Features,0))
# print mean_features
# print std_features
normed_features = [(x-mean_features)/std_features for x in Features]


### Recognition on Training Data
accuracy_train = 0
D = cdist(normed_features, normed_features)
io.imshow(D,vmin = D.mean()-3*D.std(),vmax = D.mean()+3*D.std())
plt.title('Distance␣Matrix(training␣set)')
io.show()
D_index = np.argsort(D, axis=1)
if improve_idx == 1:
    if num_k == 1:
        num_k = 2
    Ypred_train = [train.cal_knn(class_labels,num_k,x) for x in D_index]
else:
    Ypred_train = [class_labels[x[1]] for x in D_index]
accuracy_train = np.mean([Ypred_train[x] == class_labels[D_index[x][0]] for x in range
print 'The␣accuracy␣for␣training␣data␣is␣%f' %accuracy_train
## confusion matrix
confM = confusion_matrix(class_labels,Ypred_train)
io.imshow(confM)
plt.title('Confusion␣Matrix␣(training␣set)')
io.show()


# ### Testing (Recognition)
Ypred_test = []
features_test = []
regions = []
accuracy = []
for timage,timage_gt in zip(test_image,test_image_gt):
    tmp1,tmp2,tmp3 = test_xh.Recognition(timage,normed_features, mean_features,std_feat
    Ypred_test.append(tmp1)
    features_test.append(tmp2)
    regions.append(tmp3)
    ### Evaluate performance ( recognition rate)
    accuracy.append(test_xh.evaluate_ORC(timage_gt,tmp3,tmp1))
return Ypred_test, features_test, regions, accuracy
```

train:

```
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_normalize
from skimage import io, exposure, filters, morphology
import matplotlib.pyplot as plt
import os
from matplotlib.patches import Rectangle
import pickle

def feature_database(char,thr_binary,threshold_size,disp_idx,improve_idx):
    Features = []
    class_label = []
    img = io.imread(os.getcwd()+'/Documents/Computer-Vision-534/hw1/H1-16images/'+char+'.bm
    hist = exposure.histogram(img)

    if improve_idx:
        ### equated contrast
        # img = exposure.equalize_adapthist(img)

        # smoothing
        # img = filters.gaussian(img, sigma = 1)
        # img = filters.laplace(img)
        img = filters.median(img,selem = morphology.disk(1))


        # automate threshold
        # thr_binary = filters.threshold_isodata(img)
        # thr_binary = filters.threshold_li(img)
        thr_binary = filters.threshold_otsu(img)
        # thr_binary = filters.threshold_yen(img)
        # print thr_binary
        # thr_binary = 200
        img_binary = (img < thr_binary).astype(np.double)
        # img_binary = np.logical_not(filters.threshold_adaptive(img,151,method='gaussian'

        ### dilation and erosion
        img_binary = morphology.binary_closing(img_binary).astype(np.double)
        img_binary = morphology.skeletonize(img_binary).astype(np.double)
        img_binary = morphology.binary_dilation(img_binary).astype(np.double)
    else:
        img_binary = (img < thr_binary).astype(np.double)

    img_label = label(img_binary, neighbors = 8, background=0)      ### labeling
    regions = regionprops(img_label,cache = True)
    ### bounding boxes and Store features in Features
    if disp_idx:
        io.imshow(img)
        plt.title('Original_Image')
        io.show()
        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()
    io.imshow(img_binary)
    ax = plt.gca()
    for props in regions:
        minr, minc, maxr, maxc = props.bbox
        if maxc - minc >= threshold_size and maxr - minr >= threshold_size:
            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)
            Features.append(hu)
            class_label.append(char)
            # if disp_idx:
            ax.add_patch(Rectangle((minc, minr), maxc - minc, maxr - minr, fill=False, edg
    print 'Number␣of␣labeling␣components:␣%d' %len(Features)
    # if disp_idx:
    ax.set_title('Bounding␣Boxes␣of␣%s' %char)
    # save it to files
    plt.savefig('training_images_%s_improve%d.png' %(char,improve_idx))
    io.show()
    return Features,class_label

def cal_knn(class_labels,num_k,x):
    res = [class_labels[x[i]] for i in range(1,num_k)]
    return max(set(res),key = res.count)
```

test_xh :

```
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_normalize
from skimage import io, exposure, filters, morphology
import matplotlib.pyplot as plt
import os
from matplotlib.patches import Rectangle
import pickle


def cal_knn(class_labels,num_k,x):
    res = [class_labels[x[i]] for i in range(num_k)]
    return max(set(res),key = res.count)

def Recognition(filename,features_train, features_mean,features_std,class_labels,thr_binary
    features_test = []
    ## read image and binarization
    img = io.imread(os.getcwd()+'/Documents/Computer-Vision-534/hw1/H1-16images/' + filenam
    hist = exposure.histogram(img)

    if improve_idx:

        ### equated contrast
        # img = exposure.equalize_adapthist(img)

        # smoothing
        # img = filters.gaussian(img, sigma = 1)
        # img = filters.laplace(img)
        img = filters.median(img,selem = morphology.disk(1))


        # automate threshold
        # thr_binary = filters.threshold_isodata(img)
        # thr_binary = filters.threshold_li(img)
        thr_binary = filters.threshold_otsu(img)
        # thr_binary = filters.threshold_yen(img)
        # thr_binary = 200
        img_binary = (img < thr_binary).astype(np.double)
        # img_binary = np.logical_not(filters.threshold_adaptive(img,151,method='gaussian'

        ### dilation and erosion
        img_binary = morphology.binary_closing(img_binary).astype(np.double)
        img_binary = morphology.skeletonize(img_binary).astype(np.double)
        img_binary = morphology.binary_dilation(img_binary).astype(np.double)

    else:
        img_binary = (img < thr_binary).astype(np.double)


    ## Extracting Characters and Their Features
    img_label = label(img_binary, neighbors = 8, background=0)      ### labeling
```

```
### bounding boxes and Store features in Features
if disp_idx:
    io.imshow(img)
    io.show()
    plt.title('Original Image')
    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()
io.imshow(img_binary)
ax = plt.gca()
regions = regionprops(img_label,cache = True)
regions_return = []
for props in regions:
    minr, minc, maxr, maxc = props.bbox
    if maxc - minc >= threshold_size and maxr - minr >= threshold_size:
        regions_return.append([minr, minc, maxr, maxc])
        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)
        # normalization
        normed_hu = (hu - features_mean)/features_std
        features_test.append(normed_hu)
        # if disp_idx:
        ax.add_patch(Rectangle((minc, minr), maxc - minc, maxr - minr, fill=False, edg
print 'Number of labeling components: %d' %len(features_test)
# if disp_idx:
ax.set_title('Bounding Boxes')
plt.savefig('%s_Bounding_Boxes_improve%d' %(filename,improve_idx))
io.show()
## calculating a distance matrix
D = cdist(features_test,features_train)
# print np.shape(features_test),np.shape(features_train)
# if disp_idx:
io.imshow(D,aspect='auto', vmin = D.mean()-3*D.std(),vmax = D.mean()+3*D.std())
plt.title('Test data: Distance Matrix')
plt.savefig('%s_Distance_Matrix_improve%d' %(filename,improve_idx))
io.show()
D_index = np.argsort(D, axis=1)
if improve_idx:
    # Ypred = [train.class_labels[x[0]] for x in D_index]
    Ypred = [cal_knn(class_labels,num_k,x) for x in D_index]
    # train.cal_knn(class_labels,num_k,x)
else:
    Ypred = [class_labels[x[0]] for x in D_index]
    # print np.shape(D)
    return Ypred, features_test ,regions_return



def evaluate_ORC(gtruth ,regions ,Ypred_test ):
    import matplotlib.patches as patches
    # load groundtruth
    pkl_file = open(os.getcwd() + '/Documents/Computer-Vision-534/hw1/' + gtruth + '.pkl',
    mydict = pickle.load(pkl_file)
    pkl_file.close()
    classes = mydict['classes']  # N*1
    locations = mydict['locations'] # N*2
    # print classes,locations
    # function: in polygon
    def inpolygon(polygons ,p):
        for i in range(len(polygons)):
            minr, minc, maxr, maxc = polygons[i]
            if (minr-p[1])*(maxr-p[1]) <= 0 and (minc-p[0])*(maxc-p[0]) <= 0:
                return i
        return -1
    # cal recognition rate
    correct_ct = 0
    ## visualize
    fig = plt.figure(figsize = [10,10])
    ax = fig.add_subplot(111)
    for i in range(len(locations)):
        rc_gt = locations[i]
        idx = inpolygon(regions ,rc_gt)
        ## visualize
        ax.plot(rc_gt[0],rc_gt[1],'k.')
        ax.annotate(classes[i],xy = tuple(locations[i]+np.array([10,0])))
        if idx >= 0:
            ## visualize
            ax.add_patch(patches.Rectangle((regions[idx][1],regions[idx][0]),regions[idx][3
                                            regions[idx][2]-regions[idx][0], fill=False))
            if Ypred_test[idx]  == classes[i]:
                ax.plot(rc_gt[0],rc_gt[1],'r.')
                ax.annotate(Ypred_test[idx],xy = tuple(locations[i]+np.array([30,0])),colo
                correct_ct += 1
            else:
                ax.annotate(Ypred_test[idx],xy = tuple(locations[i]+np.array([30,0])),colo
    plt.gca().invert_yaxis()
    ax.set_title('Prediction␣vs.␣ground␣truth')
    # save it to files
    plt.savefig('%s_Prediction_vs_ground_truth_improve.png' %gtruth)
    plt.show()
    return float(correct_ct)/len(locations)
```