IrisRecognition

April 5, 2018

1 Readme

1.0.1 Codes from all scripts are combined and explained below, from localization to evaluation. Limitation and Improvment are talked at the end of this file

2 Import packages

```
In [1]: import IrisLocalization
        import IrisNormalization
        import FeatureExtraction
        import ImageEnhancement
        import IrisMatching
        import os
        from skimage import io
        import numpy as np
        from scipy.spatial import distance
        from sklearn.decomposition import PCA
        from sklearn.preprocessing import StandardScaler
        import pandas as pd
        import matplotlib.pyplot as plt
        from itertools import combinations,product
        %matplotlib inline
        import PerformanceEvaluation
```

3 Irislocalization.py

```
In [2]: def edge_detection(image):
            sobel_x = np.array([[-1,0,1],[-2,0,2],[-1,0,1]])
            sobel_y = np.array([[-1,-2,-1],[0,0,0],[1,2,1]])
            gaussian = 1/159 * np.array([[2,4,5,4,2],[4,9,12,9,4],[5,12,15,12,5],[4
            image = signal.convolve2d(image, gaussian)
            edge_x = signal.convolve2d(image, sobel_x)
            edge_y = signal.convolve2d(image, sobel_y)
            edge_image = np.sqrt(np.square(edge_x) + np.square(edge_y)).astype(int)
            return edge_image
```

The function above take a raw image and output an edge image using sobel kernal convolution. sobel_x,sobel_y are sobel kernels in x and y direction. gaussian is a gaussian kernal for smoothing and denoise. edge_x, edge_y are edge images in x and y directions, and edge_image is the final result

```
In [3]: def threshold(image, num):
            image_bw = image.copy()
            image_bw[image_bw <= num] = 0
            image_bw[image_bw > num] = 255
            return image_bw
```

The function above use a threshold to make a binary image. number below the threshold variable "num" becomes 0(black) and above threshold become 255(white)

```
In [4]: def find_circles(edge_img, min_radius, max_radius):
            hough_radii = np.arange(min_radius, max_radius, 2)
            hough_res = hough_circle(edge_img, hough_radii)
            accums, cx, cy, radii = hough_circle_peaks(hough_res, hough_radii,
                                                       total_num_peaks=1)
            return list(zip(cy, cx, radii))
```

The function above takes an edge image and the range of radius to find the best circle voted by Hough transformation. It takes the hough_circle() function from skimage package, which do hough transformation of an edged image and returns the radius and location of the center. "cx", "cy" are the result circle center, "radii" is the corresponding radius. "min_radius", "max_radius" are the range of radius we vote for a circle, based on our prior knowledge.

```
In [5]: def canny_edge_detection(img, sigma):
            gaussian = 1/159 * np.array([[2,4,5,4,2],[4,9,12,9,4],[5,12,15,12,5],[4
            img = signal.convolve2d(img, gaussian)
            edge_image = canny(img, sigma)
            return edge_image
```

The function above use canny() function in skimage function to do canny edge detection, and return an edge image. This function works better than the edge_detecion() function above because it return thinner edge with less noise. Also, before using the canny function I use a gaussian kernal to smooth the image.

```
In [6]: def draw_circle(image, circle):
            for center_y, center_x, radius in circle:
                circy, circx = circle_perimeter(center_y, center_x, radius)
                image[circy, circx] = 0
            io.imshow(image, cmap='gray')
```

The function above is just a helper function to help visualize the effect of iris localization. input the raw image and iris(pupil) information(center, radius) it will circle the iris(pupil).

```
In [7]: def iris_localization(image):
            def find_pupil(image):
```

```
        image = canny_edge_detection(threshold(image, 70), 10)
        circle = find_circles(image, 30, 50)
        return circle
    def find_iris(image, inner_circle):
        cy, cx, r = inner_circle[0]
        image2 = canny_edge_detection(image, 10)
        circle = find_circles(image2, 102, 120)
        return circle
    inner_circle = find_pupil(image)
    outer_circle = find_iris(image,inner_circle)
    if (outer_circle[0][0] - inner_circle[0][0])**2 + (outer_circle[0][1] -
        outer_circle = [(inner_circle[0][0], inner_circle[0][1] ,outer_circ
    return inner_circle, outer_circle
```

The function above detect pupil first by applying threshold to the original image, then detect iris circle using the canny edge detection. "inner-circle" localize the pupil, "outer_circle" localize the iris. Also, if the center of iris is too far away from pupil iris(greater than 10, after tuning this parameter with cross-validation), I just re-location the circle with the same radius but shift the iris center to pupil center. This technique improves about 10% CRR of my evalution.

4 IrisNormalization.py

```
In [8]: def iris_normalization(image):
            pupil, iris = IrisLocalization.iris_localization(image)
            iris_x, iris_y, iris_r = iris[0]
            pupil_x, pupil_y, pupil_r = pupil[0]
            M = 64
            N = 512
            new = [[0 for i in range(N)] for j in range(M)]
            for i in range(M):
                for j in range(N):
                    theta = 2*np.pi*j/N
                    x = pupil_r*np.cos(theta) + (iris_r*np.cos(theta) - pupil_r*np.
                    y = pupil_r*np.sin(theta) + (iris_r*np.sin(theta) - pupil_r*np.
                    new[i][j] = image[min(279,int(y) + pupil_x)][min(319,int(x) + p
            return np.array(new)
```

The function above perform iris normalization. First it run the iris_localization() function to get the pupil and iris. "iris_x", "iris_y", "iris_r" represent the iris center and radius, "pupil_x", "pupil_y", "pupil_r" represent the pupil center and radius. M, N are height and width of the result image; and the nested for loop just apply the formula in Ma's paper(x, y represent the location in the new image and theta is the angle of each slice); and the image after normalization is stored in "new" variables.

5 ImageEnhancement.py

```
In [9]: def background(normalized_image, delta):
            M,N = normalized_image.shape
```

```
    background = np.array([[0 for i in range(int(N/delta))] for j in range
    for i in range(int(N/delta)):
        for j in range(int(M/delta)):
            background[j][i] = np.mean(normalized_image[j*delta:j*delta+del
    background = misc.imresize(background,(64,512),'bicubic')
    return background
```

The above function compute the background followed the same precedure in Ma's paper. The nested for loop iterate every 16*16 blocks(setting delta = 16) and compute the mean, then using the imresize() function in scipy.misc to get the original size background by bicubic interpolation.

```
In [10]: def background_subtraction(normalized_image,background):
             return cv2.subtract(normalized_image, background)
```

The above function just return the normalized image subtracted by background using the cv2.subtract() function in opencv2 package.

```
In [11]: def enhancement(image, delta):
             M,N = image.shape
             enhanced = image.copy()
             for i in range(int(N/delta)):
                 for j in range(int(M/delta)):
                     local = enhanced[j*delta:j*delta+delta,i*delta:i*delta+delta]
                     local = exposure.equalize_hist(local)
                     enhanced[j*delta:j*delta+delta,i*delta:i*delta+delta] = local*
             return enhanced
```

The function above perform image enhancement for every 3232 *block by setting delta=32(followed Ma's paper). The nested for loop perform histogram equalization for each 3232* block using exposure.equalize_hist() in skimage package

6 FeatureExtraction.py

```
In [12]: def gobar_filter(x,y,sigma_x,sigma_y):
             G = (1/(2*np.pi*sigma_x*sigma_y)) * np.exp(-0.5*(x**2/sigma_x**2 + y**
             return G
```

The function above return a gobar filter followed the equation in Ma's paper. "sigma_x","sigma_y" are (3,1.5) for the first filter and (4.5,1.5) for the second filter.

```
In [13]: def M1(x,y,f):
             return np.cos(2*np.pi*f*(np.sqrt(x**2 + y**2)))
```

The function above compute the sinusoidal part to get the desiged filter in Ma's paper

```
In [14]: def spatial_filter(enhanced, sigma_x,sigma_y,height,width):
             kernel = np.array([[0.0 for i in range(9)] for j in range(9)])
             for y in range(0 - height,height+1):
                 for x in range(0 - width,width+1):
                     kernel[height+y,width+x] = gobar_filter(x,y,sigma_x,sigma_y) *
             return signal.convolve2d(enhanced,kernel,'same')
```

The function above returns a *99 filter insteat of 88* because it gives better performance the the evaluation step, then perform convolution with the enhanced image "enhanced". the nested for loop compute each element of the designed filter, then store it in "kernal" variable. "height", "width" are 4 for this 9*9 filter.

```
In [15]: def feature_extraction(delta, roi):
             h,w = roi.shape
             features = []
             for i in range(int(w/delta)):
                 for j in range(int(h/delta)):
                     mean = np.mean(roi[j*delta:j*delta+delta, i*delta:i*delta+delt
                     std = np.std(roi[j*delta:j*delta+delta, i*delta:i*delta+delta]
                     features+=[mean,std]
             return features
```

The function above perform feature extration from "roi", which is the upper 48*512 image. The nested for loop here compute the mean and standard devation for every 88* block(set delta = 8).

7 IrisMatching.py

```
In [16]: def nearest_neighbor(features_train, features_test, labels_train, n):
             neighbors = []
             scaler = StandardScaler()
             features_train = scaler.fit_transform(features_train)
             features_test = scaler.fit_transform(features_test)
             for i in range(len(features_test)):
                 neighbor = 0
                 dis = np.inf
                 for j in range(len(features_train)):
                     d = distance.minkowski(features_test[i],features_train[j],p =
                     if d < dis:
                         dis = d
                         neighbor = j
                 neighbors.append(labels_train[neighbor])
             return neighbors
```

The function above search the nearest neighbor based on minkowski distance after using the StandardScaler() in sklearn. "neighbor" keep track of the current nearest neighbor, and "dis" stores the current shortest distance. if a closer neighbor is found, then update both "neighbor" and "dis" variables. Then append each nearest neighbor's class into a list called "neighbors".

```
In [17]: def nearest_neighbor_cosine(features_train, features_test, labels_train):
             neighbors = []
             scaler = StandardScaler()
             features_train = scaler.fit_transform(features_train)
             features_test = scaler.fit_transform(features_test)
             for i in range(len(features_test)):
                 neighbor = 0
```

```
            dis = np.inf
            for j in range(len(features_train)):
                d = distance.cosine(features_test[i],features_train[j])
                if d < dis:
                    dis = d
                    neighbor = j
            neighbors.append(labels_train[neighbor])
        return neighbors
```

The function above perform the same nearest neighbor search, but change the distance metric to cosine distance in scipy.spatial package.

```
In [18]: def dimentionality_reduction(features_train, features_test,components = 32
         scaler = StandardScaler()
         features_train = scaler.fit_transform(features_train)
         features_test = scaler.fit_transform(features_test)
         pca = PCA(n_components = components)
         f_train = pca.fit_transform(features_train)
         f_test = pca.transform(features_test)
         f_train = scaler.fit_transform(f_train)
         f_test = scaler.fit_transform(f_test)
         return f_train, f_test
```

The function above perform PCA to reduce the number of features instead of LDA, because it gives better result in the evaluation step. PCA() is a class in sklearn.decomposition package. Also before and after PCA, doing nomalization by StandardScaler() class in sklearn package will also increase CRR.

8 PerformanceEvaluation.py

```
In [19]: def compute_CRR(features_train, features_test, labels_train,labels_test, m
         if metric == 'L1':
             prediction = IrisMatching.nearest_neighbor(features_train, feature
         elif metric == 'L2':
             prediction = IrisMatching.nearest_neighbor(features_train, feature
         else:
             prediction = IrisMatching.nearest_neighbor_cosine(features_train,
         return (list(np.array(prediction) - np.array(labels_test)).count(0))/1
```

The function above compute the CRR for three different mertic “L1”,“L2” and “cosine”

```
In [20]: def verification(x,y,threshold):
         if distance.cosine(x,y) < threshold:
             return 1
         else:
             return 0
```

The code above check whether the cosine distance is smaller than the given threshold. (1 is smaller, 0 if not)

```
In [21]: def compute_FMR(predicted_label, matched_pair, labels):
             return 1 - (list(np.array(predicted_label[:len(matched_pair)]) - np.ar
         def compute_FNMR(predicted_label, matched_pair, labels):
             return 1 - (list(np.array(predicted_label[:len(matched_pair)]) - np.ar
```

The function above compute the FMR and FNMR

9 Main Function

```
In [22]: base_dir = os.getcwd() + '/CASIA Iris Image Database (version 1.0)'
         def read_data(base_dir):
             train_image = []
             test_image = []
             for filename in os.listdir(base_dir)[1:]:
                 for file in os.listdir(base_dir + '/' + filename):
                     if str(file) == '1':
                         for image in os.listdir(base_dir + '/' + filename + '/' +
                             try:
                                 img = io.imread(base_dir + '/' + filename + '/' +
                                 train_image.append(img)
                             except:
                                 pass
                     elif str(file) == '2':
                         for image in os.listdir(base_dir + '/' + filename + '/' +
                             try:
                                 img = io.imread(base_dir + '/' + filename + '/' +
                                 test_image.append(img)
                             except:
                                 pass
                     else:
                         pass
             return train_image, test_image
```

The function above read all train and test image, stored in two lists "train_image", "test_image"

```
In [23]: # load images
         train_image, test_image = read_data(base_dir)
```

```
In [24]: def image_process(image_list):
             features = []
             for image in image_list:
                 normalized_image = IrisNormalization.iris_normalization(image)
                 enhanced = ImageEnhancement.enhancement(normalized_image, 32)
                 sigma_x1,sigma_y1 = 3.0,1.5
                 sigma_x2,sigma_y2 = 4.5,1.5
                 roi_1 = FeatureExtraction.spatial_filter(enhanced,sigma_x1,sigma_y
                 roi_2 = FeatureExtraction.spatial_filter(enhanced,sigma_x2,sigma_y
```

```
            features.append(FeatureExtraction.feature_extraction(8, roi_1) + F
        return np.array(features)
```

The function above pre-process each image(localization, normalization, feature extration) to get the feature vectors

```
In [25]: # pre_process images in train and test dataset
         features_train = image_process(train_image)
         features_test = image_process(test_image)
```

```
In [26]: #get corresponding labels for train and test set
         labels_train = []
         for i in range(int(len(features_train)/3)):
             for j in range(3):
                 labels_train.append(i + 1)
         labels_test = []
         for i in range(int(len(features_test)/4)):
             for j in range(4):
                 labels_test.append(i + 1)
         labels_train = np.array(labels_train)
         labels_test = np.array(labels_test)
```

10 compute the CRR for L1,L2,cosine distance using all features:

```
In [27]: L1_full = PerformanceEvaluation.compute_CRR(features_train, features_test,
         L2_full = PerformanceEvaluation.compute_CRR(features_train, features_test,
         cosine_full = PerformanceEvaluation.compute_CRR(features_train, features_t
```

10.0.1 compute the CRR for L1,L2,cosine distance using reduced number of features(180 features gives the best accuracy):

```
In [28]: f_train, f_test = IrisMatching.dimentionality_reduction(features_train, fe
```

```
In [29]: L1_reduced = PerformanceEvaluation.compute_CRR(f_train, f_test, labels_tra
         L2_reduced = PerformanceEvaluation.compute_CRR(f_train, f_test, labels_tra
         cosine_reduced = PerformanceEvaluation.compute_CRR(f_train, f_test, labels
```

11 Output the table T3

```
In [30]: CRR = {"L1 distance measure":[L1_full,L1_reduced],
                 "L2 distance measure":[L2_full,L2_reduced],
                 "cosine distance measure":[cosine_full,cosine_reduced]}
         CRR_table = pd.DataFrame.from_dict(CRR,orient='index')
         CRR_table.columns = ['Original Feature Set','Reduced Feature Set']
         CRR_table
```

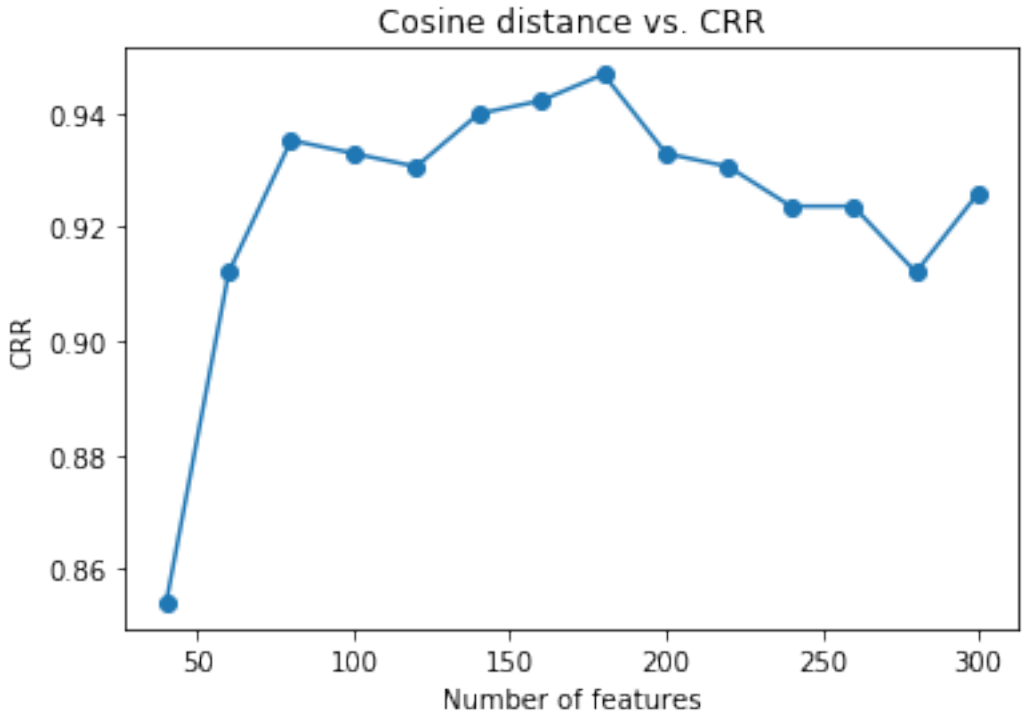
Out[30]:

	Original Feature Set	Reduced Feature Set
L1 distance measure	0.870370	0.870370
L2 distance measure	0.861111	0.875000
cosine distance measure	0.861111	0.946759

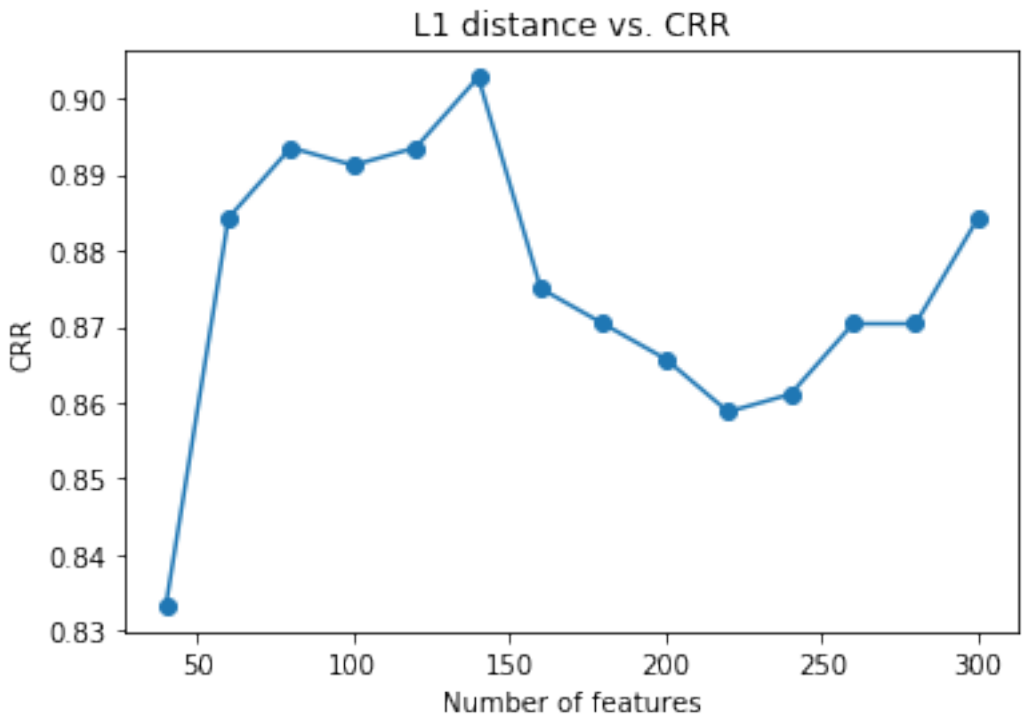
12 Output fig 10 with 3 distance metric:

```
In [31]: score_cosine = []
         score_L1 = []
         score_L2 = []
         for num in range(40, 320, 20):
             f_train, f_test = IrisMatching.dimentionality_reduction(features_train
             L1_reduced = PerformanceEvaluation.compute_CRR(f_train, f_test, labels
             L2_reduced = PerformanceEvaluation.compute_CRR(f_train, f_test, labels
             cosine_reduced = PerformanceEvaluation.compute_CRR(f_train, f_test, la
             score_cosine.append((num,cosine_reduced))
             score_L1.append((num,L1_reduced))
             score_L2.append((num,L2_reduced))
```

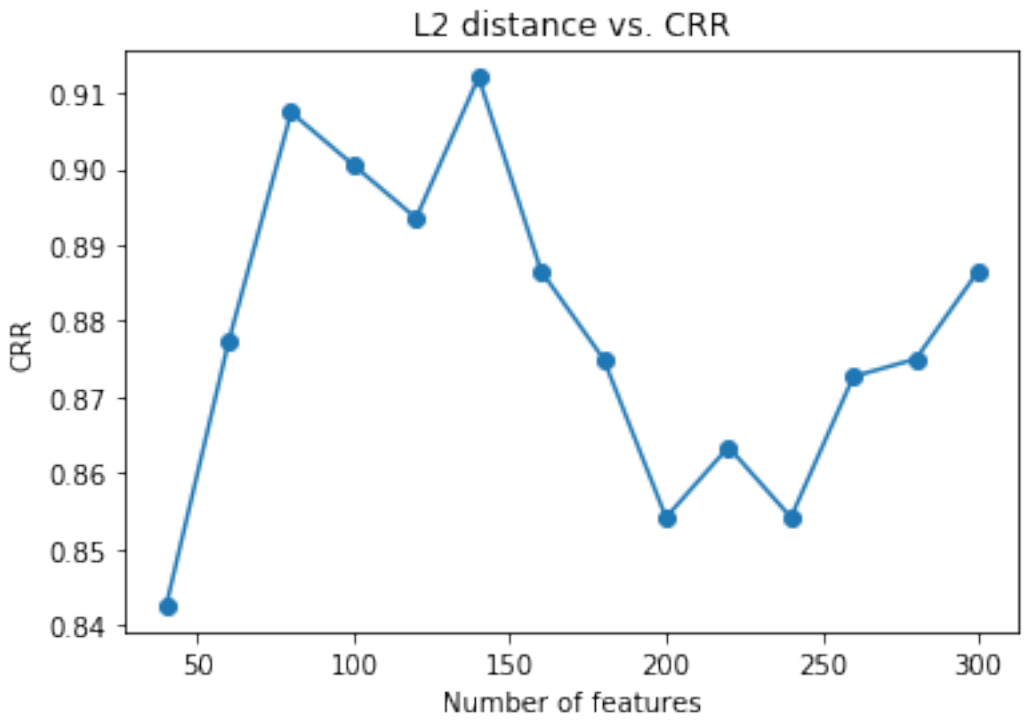
```
In [32]: plt.plot(*zip(*score_cosine), '-o')
         plt.xlabel("Number of features")
         plt.ylabel("CRR")
         plt.title("Cosine distance vs. CRR")
         plt.show()
```




```
In [33]: plt.plot(*zip(*score_L1), '-o')
         plt.xlabel("Number of features")
         plt.ylabel("CRR")
         plt.title("L1 distance vs. CRR")
         plt.show()
```




```
In [34]: plt.plot(*zip(*score_L2), '-o')
         plt.xlabel("Number of features")
         plt.ylabel("CRR")
         plt.title("L2 distance vs. CRR")
         plt.show()
```



13 Generate all possible combinations of train test pairs to do verification:

```
In [35]: f_train, f_test = IrisMatching.dimentionality_reduction(features_train, fe
         d_train = {}
         for k,v in zip(list(labels_train),list(f_train)):
             if k not in d_train:
                 d_train[k] = []
                 d_train[k].append(tuple(v))
             else:
                 d_train[k].append(tuple(v))

In [36]: d_test = {}
         for k,v in zip(list(labels_test),list(f_test)):
             if k not in d_test:
                 d_test[k] = []
                 d_test[k].append(tuple(v))
             else:
                 d_test[k].append(tuple(v))
```

generate those matched pairs, these pairs should be accepted as same iris

```
In [37]: matched_pair = []
         for k in d_train:
             matched_pair += list(product(d_train[k], d_test[k]))
```

generate those unmatched pairs, these pairs should not be accepted as same iris

```
In [38]: unmatched_pair = []
         for i in range(1,1 + len(d_train.keys())):
             for j in range(1, 1 + len(d_train.keys())):
                 if i != j:
                     unmatched_pair += list(product(d_train[i], d_test[j]))
```

Combine matched and unmatched pairs, and generate corresponding labels(1 or 0)

```
In [39]: pairs = matched_pair + unmatched_pair
         labels = [1 for x in matched_pair] + [0 for x in unmatched_pair]
```

Verification mode based on different thresholds:

```
In [40]: predicted_labels = []
         thresholds = [0.6,0.625,0.65,0.675,0.7,0.725,0.75,0.775,0.8]
         for threshold in thresholds:
             temp = []
             for x,y in pairs:
                 temp.append(PerformanceEvaluation.verification(x,y,threshold))
             predicted_labels.append(temp)
```

```
In [41]: FMR = []
         FNMR = []
         for predicted_label in predicted_labels:
             FMR.append(PerformanceEvaluation.compute_FMR(predicted_label, matched_
             FNMR.append(PerformanceEvaluation.compute_FNMR(predicted_label, matche
```

Output table 4:

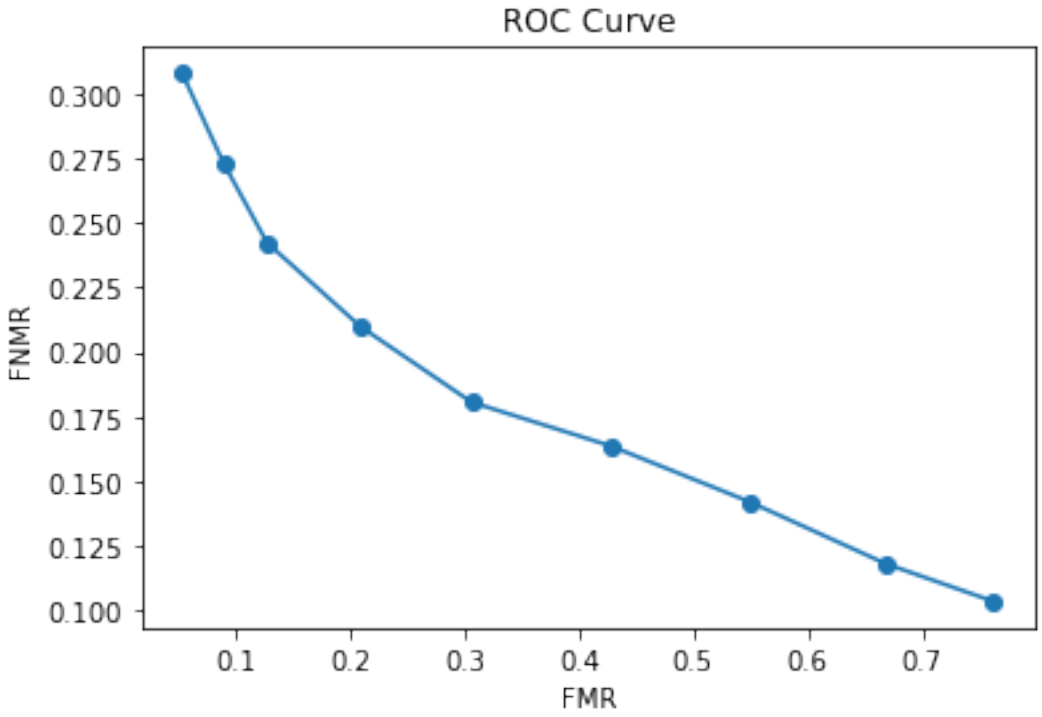
```
In [42]: table_verification = pd.DataFrame(np.array([thresholds,FMR,FNMR]).T, colum
         table_verification = table_verification.set_index('Threshold')
         table_verification
```

Out[42]:

Threshold	FMR	FNMR
0.600	0.053797	0.307870
0.625	0.089855	0.273148
0.650	0.127886	0.242284
0.675	0.209266	0.209877
0.700	0.306336	0.180556
0.725	0.427968	0.163580
0.750	0.549067	0.141975
0.775	0.667636	0.118056
0.800	0.762080	0.103395

Plot fig 13:

```
In [43]: plt.plot(FMR,FNMR,'-o')
         plt.xlabel("FMR")
         plt.ylabel("FNMR")
         plt.title("ROC Curve")
         plt.show()
```



14 Limitation and Improvement:

1. Not consider the rotation effect when computing the nearest neighbor, result in a little bit high FMR and FNMR when doing verification, because a small rotation will lead to large distance. To imptove, just add a little rotation with doing normalization, such as (-9,-6,-3,3,6,9) degree, mentioned by the paper.
2. Bacause out dataset is small, I use the full dataset to do verification instead of boostrap samples and compute confidence interval. If dataset grows, using all possibles pairs will decrease the speed when testing different threshold. So boostraping to compute confidence interval is a better idea.
3. Localization is not accurate because i didn't remove those noise edges. Removing them will lead to better localization result and improve the CRR.