SIL775 Biometric Security
Assignment 2 (Iris Recognition System)

Develop a Compressed Sensing based Iris Recognition system using the following:

Methodology:

1. Pre-processing

Pre-processing Comprises Iris Detection using :

Canny edge algorithm :

To do the canny edge detection the openCv library is been used(cv2.Canny()).

Enhancement using Gamma Correction :

Gamma-correction is been performed to enhance the image , the code to do the following is given below.

```
def gamma_correcetion(img_original,gamma = 1.4):
    lookUpTable = np.empty((1,256), np.uint8)
    for i in range(256):
        lookUpTable[0,i] = np.clip(pow(i / 255.0, gamma) * 255.0, 0, 255)
    res = cv2.LUT(img_original, lookUpTable)
    return res
```

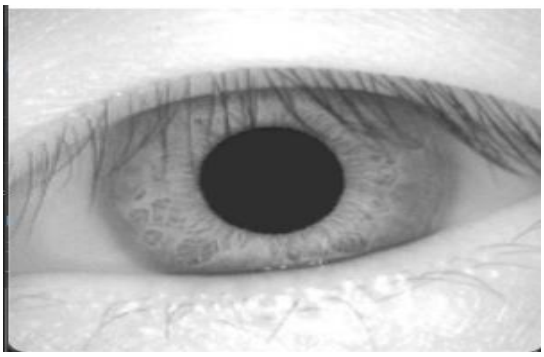
✓ 0.6s

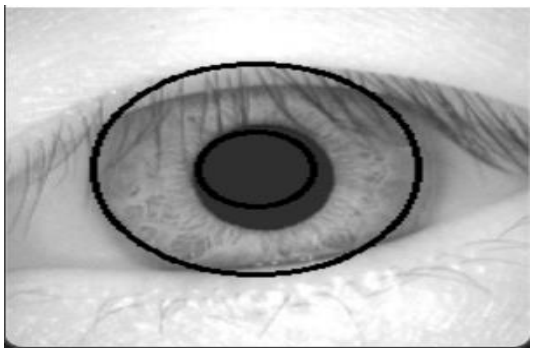
Iris localization using Hough Transform :

To do the iris localization again openCv library is used(cv2.HoughCircles()) and corresponding Inner and the outer circles were drawn into the image.

```
def Localization(images):

    boundary=[]
    centers=[]

    for img in images:

        blur = cv2.GaussianBlur(img,(5,5),0)

        maskimage = cv2.inRange(blur, 0, 70)
        output = cv2.bitwise_and(blur, maskimage)

        edged = cv2.Canny(output, 100, 220)

        edged_gamma = gamma_correcetion(edged)

        circles = cv2.HoughCircles(edged_gamma, cv2.HOUGH_GRADIENT, 10, 100)
        circles = np.uint16(np.around(circles))

        cv2.circle(img,(circles[0][0][0],circles[0][0][1]),circles[0][0][2],(0,255,0),2)
        cv2.circle(img,(circles[0][0][0],circles[0][0][1]),circles[0][0][2]+40,(0,255,0),2)

        boundary.append(img)
        centers.append([circles[0][0][0],circles[0][0][1],circles[0][0][2]])

    return boundary,centers
```

✓ 0.8s





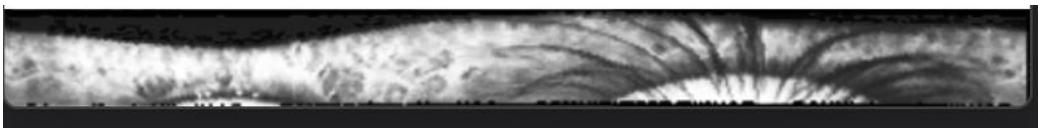
Normalization :

Normalization is performed to the image obtained after canny edge detection and gamma correction to obtain the template of the image.

```python
def Normalization(boundary,centers):

    img_lst = [i for i in boundary]
    iris_radius,totalsamples = 40,360
    pival = 2*np.pi
    normalized=[]
    i=0

    for img in img_lst:
        center_x,center_y,radius_pupil = centers[i][0],centers[i][1],int(centers[i][2])
        polar,samples = np.zeros((iris_radius, totalsamples)),np.linspace(0,pival, totalsamples)[:-1]
        for r in range(iris_radius):
            for angle in samples:
                x,y = int((r+radius_pupil)*np.cos(angle)+center_x),int((r+radius_pupil)*np.sin(angle)+center_y)
                try:
                    polar[r][int((angle*totalsamples)/(pival))] = img[y][x]
                except IndexError:
                    pass
                continue
        normalized.append(cv2.resize(polar,(512,48)))
        i+=1
    return normalized
```

✓ 0.8s

Template :



2. Feature Extraction

A function gabor is been defined which is used to in calculate the spatial filter .

It runs over the normalized image by making an 8x8 block to the features.

To omit the effect of eyelashes the block runs over the 48*512 region instead of the whole image .

```
def FeatureExtraction(enhanced):

    feature_vector=[]
    filter1 = np.zeros((8,8))
    filter2 = filter1

    for i in range(8):
        for j in range(8):
            filter1[i,j] = gabor_filter((-4+j),(-4+i),3,1.5,0.67)

    for i in range(8):
        for j in range(8):
            filter2[i,j] = gabor_filter((-4+j),(-4+i),4,1.5,0.67)

    for i in range(len(enhanced)):
        img = enhanced[i]
        img_roi = img[:48,:]
        feature_vector.append(get_vec(scipy.signal.convolve2d(img_roi,filter1,mode='same'),scipy.signal.convolve2d(img_roi,filter2,mode='same')))

    return feature_vector
```

A function is defined to create two channels of filters and these filters were convolved in the region of interest to get the two filtered images.

```
def gabor_filter(x, y, dx, dy, f):
    val = np.cos(2*np.pi*f*math.sqrt(x **2 + y**2))
    mul_a = np.exp(-0.5*(x**2 / dx**2 + y**2 / dy**2))
    mul_b = (1/(2*math.pi*dx*dy))
    gb = mul_b * mul_a * val
    return gb

def get_std(grid):
    absolute = np.absolute(grid)
    mean = np.mean(absolute)
    std = np.mean(np.absolute(absolute-mean))
    return mean,std

def get_vec(convolvedtrain1,convolvedtrain2):

    feature_vec=[]

    for i in range(6):
            for j in range(64):

                start_height,end_height = i*8,i*8+8
                start_wid,end_wid = j*8,j*8+8

                grid1,grid2 = convolvedtrain1[start_height:end_height, start_wid:end_wid],convolvedtrain2[start_height:end_height, start_wid:end_wid]

                mean1,std1 = get_std(grid1)
                mean2,std2 = get_std(grid2)

                feature_vec.append(mean1)
                feature_vec.append(std1)
                feature_vec.append(mean2)
                feature_vec.append(std2)

    return feature_vec
```

The filtered images are then used to calculate mean and standard-deviation grid by grid with a grid of dimension 8x8.

The calculated values are appended to form the feature vector of size 1536 (6x64x4).

For every image the feature vectors were appended into the feature_vec.

3. Formation of Dictionary

In our case 10 classes were taken with 3 images per person

So in total 30 images were loaded initially from the file and a dictionary is been created of the features vectors of these images.

The dimension of the dictionary is 1536x30

```
Dk = []
for num in range(1,11):
    images  = []
    for img in glob.glob('D:/IITD_M.tech/Sem_2/Biometric_security/Assignment_2/IrisRecognition-master/CASIA1_train/' + str(num) + '/*.jpg'):
        n= cv2.imread(img)
        n = cv2.cvtColor(n, cv2.COLOR_BGR2GRAY)
        n = cv2.resize(n, (200, 200))
        images.append(n)

    # print(len(images_train))
    # cv2.imshow('normalized_image',images_train[1])
    # cv2.waitKey(0)

    boundary,centers = Localization(images)
    normalized = Normalization(boundary,centers)

    # cv2.imshow('normalized_image',normalized_train[0])
    # cv2.waitKey(0)

    enhanced=[]
    for res in normalized:
        res = res.astype(np.uint8)
        im=cv2.equalizeHist(res)
        enhanced.append(im)

    feature_vector = FeatureExtraction(enhanced)
    Dk.append(feature_vector)

arr = []
for i in Dk:
    for j in i:
        arr.append(j)

D = np.array(arr)
D = np.transpose(D)
```

```
1  print(D.shape)
✓ 0.7s
(1536, 30)
```

4. Finding Sparse vector using basis pursuits

The query image is loaded and corresponding feature vector is obtained.

The parameters D, y and alpha are dictionary, feature_vector of the query image and the spare vector respectively.

To solve the convex optimization problem(Second order cone programming) the CVXPY library is been used in which the constraints and the objective function is defined and loaded as required.

```
m = 1536
n = 30
epsilon = 0.1
y = np.array(feature_vector_test)

alpha = cp.Variable(n)
soc_objective = cp.Minimize(cp.norm(alpha,1))
soc_constraints = [cp.norm((D @ alpha - y),2)<=epsilon]

soc_prob = cp.Problem(soc_objective,soc_constraints)
result = soc_prob.solve()

values = alpha.value
values_m = values.argmin()
```

5. Matching

At last the matching is performed by first getting the sparse vector alpha and then a characteristic function is defined which will output a delta vector having one at the locations where the dimensions of alpha will be greater than a threshold.

Using that delta a residual error vector is obtained and the class of the image corresponds to the index of lowest residual error in the vector.

```
def resudl_error(y,D,values_m):

    residual_error = []

    for i in range(1,11):
        delta = char_function(i)
        # print(delta)
        a = np.linalg.norm((y - D.dot(delta)))
        residual_error.append(a)

    residual_error = np.array(residual_error)
    print(residual_error)

    c = math.floor(values_m/3)+1
    min_er = residual_error.argmin() + 1

    return residual_error,min_er,c
```

Results :

The query image is been loaded to check for the class of the image.

The following residual error vector is been obtained along with the class label of the image.

The image belongs to the class having the minimum residual error.

```
[125.20792253 122.84674057 120.20304239 123.44683683 120.67157387
 121.65415054 108.8473496  116.23086876 125.37356834 119.19445376]
The image belongs to the class : 7
```