دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

6/10/2020

Report Homework 7

Advanced Programming

Mohammad Javad Amin (9523008)
AMIRKABIR UNIVERSITY OF TECHNOLOGY

به نام خدا

هدف این سری تمرین به تشخیص چهره است.

۱) تابع loadImage را تعریف می‌کنیم که ورودی آن آدرس محل عکس است. در این تابع عکس را خوانده و سپس آن را به ابعاد 30*30 در می‌آوریم و خروجی این تابع flatten عکس(900*1) و عکس 30*30 است. این تابع به صورت زیر پیاده‌سازی می‌شود.

```
def loadImage(filename):   #load image from adress
    img=cv2.imread(filename,0)
    face=cv2.resize(img,(30,30))    #resize to 30*30
    return face.reshape((900,1)),face
```

۲)تابع loadfaces : این تابع نام فایل را به عنوان ورودی گرفته و سپس آن را از حالت فشرده خارج میکند سپس با استفاده از تابع loadimage تمام عکس را خوانده و نرمالیزه کرده و در ارایه به عنوان خروجی برمی‌گرداند . این تابع به صورت زیر پیاده‌سازی می‌شود.

```
def loadFaces(filename):
    list_faces=[]     #list all faces
    zf=ZipFile(filename+'.zip','r')
    zf.extractall()    #extract zip file
    for root, dirs, files in os.walk(filename):
        for file in files:
            #append the file name to the list
            X=loadImage(os.path.join(root,file)) #load each image
            list_faces.append(X[0]/np.max(X[0]))       #normalize each image
    array_faces=np.array(list_faces).reshape((400,900))
    return array_faces.transpose()  #
```

۳)findfaces : ورودی این تابع cov و تعداد eigenfaces ها استو در این تابع ابتدا eigen vector و eigenvalue را حساب کرده و سپس eigenvalue را مرتب می‌کنیم و همچنین جای eigen vector را متناسب با eigenvalue ها مرتب می‌کنیم. سپس به تعداد خواسته شده eigen vector و eigenvalue را در خروجی برمی‌گردانیم. این تابع به صورت زیر پیاده‌سازی می‌شود.

```
def findEigenFaces(cov,num_faces):
    w,v=np.linalg.eig(cov)   #find eigenvectors and eigenvalue
    #ignore imaginary part
    v=np.real(v)
    w=np.real(w)

    #sort eigenvlaue
    idx=w.argsort()[::-1]
    w=w[idx]
    v=v[:,idx]

    return v[:,:num_faces],w[:num_faces]
```

۴)showEigenFaces: در این تابع ما eigen face هایی که از تابع findfaces بدست آمده را ازکوچک به بزرگ چاپ می‌کنیم و آن را به صورت عکس ذخیره می‌کنیم. این تابع به صورت زیر پیاده‌سازی می‌شود.

```
def showEigenFaces(efaces, size):
    rows=size[0]  #row
    columns=size[1]  #columns
    fig, axs = plt.subplots(rows, columns)  #plot
    count=0
    efaces=efaces[:,::-1] #reverse efacese

    for i in range(rows):
        for j in range(columns):   #draw figure
            img=efaces[:,count].reshape((30,30))
            axs[i][j].imshow(img,cmap='gray')
            axs[i,j].set_title(f'figure {count}',fontsize=10)
            axs[i,j].set_xticks([]),axs[i,j].set_yticks([])
            count+=1
    plt.tight_layout()
    plt.savefig('FaceRecognition.png')  #save figure
```

۵)converFaces : در این تابع ما عکس ورودی را روی eignvectorها project می‌کنیم. این تابع به صورت زیر پیاده‌سازی می‌شود.

```
def convertFace(X, eigenfaces):    #project faeces
    return np.matmul(np.transpose(eigenfaces), X)
```

۶)createDataset : این تابع اسم فایل حاوی عکس ها را به عنوان ورودی می‌گیردو سپس به درون هر فایل رفته هر عکس را با استفاده از تابع loadImage خوانده سپس با تابع converFaces آن را project می‌دهد خروجی را همراه با اسم folder به صورت tuple در یک list ذخیره می‌کند و به عنوان خروجی برمی‌گرداند. این تابع به صورت زیر پیاده‌سازی می‌شود.

```
def createDataset(filename,efaces):  #create dataset
    if os.path.isdir(filename) is False:
        zf=ZipFile(filename+'.zip','r')
        zf.extractall()   #extract zip file

    filelist = []  #list of file
    for root, dirs, files in os.walk(filename):
         for file in files:
             #append the file name to the list
             filelist.append(os.path.join(root,file))  #adress of image
    sub_file = os.listdir(filename)   #list of sub file(name each person)
    list_data=[]  #dataset
    num=int(len(filelist)/len(sub_file))  #number of image each person
    for i in range(len(sub_file)):
        for j in range(num):
            data=loadImage(filelist[i*num+j])  #load image
            list_data.append((convertFace(data[0]/np.max(data[0]), efaces),sub_file[i]))#add
    return list_data
```

۷)KNN : در این تابع ما یک عکس جدید به عنوان ورودی می‌گیریم ابتدا برای یافتن صاحب عکس آن را project کرده و سپس فاصله اقلیدسی این نمونه با نمونه های موجود در dataset را حساب می‌کنیم سپس آن را مرتب میکنیم و K تای کمتر آن را جدا کرده و در یک listذخیره کرده سپس باید در listاسم کسی که بیشترین تکرار را دارد را برگردانیم(صاحب عکس). این تابع به صورت زیر پیاده‌سازی می‌شود.

```
def kNN(dataset, input_face_vec, eigenfaces, k) :

    in_face_project=convertFace(input_face_vec, eigenfaces) #project new face
    list_find=[]
    for i in dataset:  #claculate Euclidean distances
        d=distance.euclidean(i[0],in_face_project)
        list_find.append((d,i[1]))  #save distance and name
    list_find.sort()   #sort distance
    list_find=list_find[:k]

    count=Counter(elem[1] for elem in list_find)  #count each name
    name=count.most_common(1)[0][0]   #most repeated name
    return name,list_find
```

Math Game

۱)generateArray : این تابع ۱۰۰ عدد بین ۱۰۰ تا ۹۹۹ به صورت رندوم تولیدکرده که المان ها مضرب ۹ آن دارای مقدار ها یکسان است. این تابع به صورت زیر پیاده‌سازی می‌شود.

```
from numpy import random
def generateArray():
    x=random.randint(100,999, size=(100))  #generate 100 random number
    x[8::9]=x[8]
    return x.reshape((10,10))
```

پایان