基于人脸 T 形分布 Gabor 变换的人脸识别算法开发文档

一、主要内容

PCA 方法是人脸识别中最基本也是最常见的方法 ,目前很多的人脸识别算法都是对这种方法的改进或与其他方法的综合．在人脸识别过程中，这种方法考虑的是样本整体特征，通过抽取人脸的主要成分，构建特征脸空间，对比人脸图像进行识别．由于其降维和特征提取方面的有效性，在人脸识别领域得到广泛应用。

受采集人脸图像时环境光照等因素影响，传统的代数方法很难达到较高识别率．Gabor 小波变换呈现为频率和方向的多尺度变换，并且与人类视觉感受野剖面非常相似，因此 Gabor 滤波器被用于提取图像局部纹理特征或对整体图像进行卷积得到 Gabor 滤波后的图像．将 Gabor 统计纹理特征用于人脸识别，可以很好地消除外界环境对识别正确率的影响，具有更强的鲁棒性。

本文将 Gabor 小波和 PCA 方法结合起来，对人脸图像进行 Gabor 小波卷积，提取基于人脸 T 形分布 Gabor 变换后特征，然后通过 PCA 降维, 得到特征向量，最后用夹角余弦计算相似度，通过实验得到较好的识别效果,以阈值 0.6 为准，判断是否属于同一人，大于 0.6 则为同一人，小于 0.6 则不是同一人。

二、Garbor 变换特征提取算法步骤

1、读入图像，将图像 image 归一化大小（112*92）。
2、将归一化后的图像灰度化，gamma 校正，DoG 滤波，直方图均衡化等一系列预处理。
3、定义了一个 5 尺度 8 方向的 Gabor 变换。
4、预处理后的图像与 Gabor 滤波器进行卷积，得到卷积后的 Gabor 特征。
5、选取 T 形分布的 Gabor 特征，然后将 Gabor 特征构成行向量。
6、将行向量进行 PCA 降维，得到特征向量。
7、通过余弦夹角计算两个特征向量的相似度。

三、建模过程

3.1、读入图像，将图像归一化

```
filename1="E:/taqu/279.jpg"
filename2="E:/taqu/281.jpg"
img11=cv.imread(filename1)
img22=cv.imread(filename2)
img11=cv.resize(img11, (92,112), interpolation=cv.INTER_CUBIC)
img22=cv.resize(img22, (92,112), interpolation=cv.INTER_CUBIC)
```

3.2、将归一化后的图像进行预处理

3.2.1、将归一化后的图像进行灰度化处理

```
src = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
```

3.2.2、将灰度化后的图像进行 gamma 校正

所谓伽玛校正就是对图像的伽玛曲线进行编辑，以对图像进行非线性色调编辑的方法，检出图像信号中的深色部分和浅色部分，并使两者比例增大，从而提高图像对比度效果。

Gamma 校正公式如下：

$$I(x,y) = I(x,y)^{\frac{1}{\text{gamma}}}$$

gamma 校正原理：

假设图像中有一个像素，值是 200 ，那么对这个像素进行校正必须执行如下步骤：

1、归一化 :将像素值转换为 0 ~1 之间的实数。算法如下 :(i + 0. 5)/256 这里包含 1 个除法和 1 个加法操作。对于像素 A 而言，其对应的归一化值为 0. 783203 。

2、 预补偿 ：根据公式, 求出像素归一化后的 数据以 1 /gamma 为指数的对应值。这一步包含一个求指数运算。若 gamma 值为 2. 2 , 则 1 /gamma 为 0. 454545 , 对归一化后的 A 值进行预补偿的结果就是 0. 783203 ^0. 454545 = 0. 894872 。

3、反归一化 :将经过预补偿的实数值反变换为 0 ~ 255 之间的整数值。具体算法为 :f*256 - 0. 5 此步骤包含一个乘法和一个减法运算。续前例 ，将 A 的预补偿结果 0. 894872 代入上式 , 得到 A 预补偿后对应的像素值为 228 , 这个 228 就是最后送入显示器的数据。

如上所述如果直接按公式编程的话，假设图像的分辨率为 800*600 ，对它进行 gamma 校正，需要执行 48 万个浮点数乘法、除法和指数运算。效率太低，根本达不到实时的效果。

针对上述情况，提出了一种快速算法，如果能够确知图像的像素取值范围 , 例如 ，0 ~ 255 之间的整数 , 则图像中任何一个像素值只能 是 0 到 255 这 256 个整数中的某一个 ; 在 gamma 值已知的情况下 ,0 ~ 255 之间的任一整数 , 经过 “归一 化、预补偿、反归一化” 操作后 , 所对应的结果是唯一的 , 并且也落在 0 ~ 255 这个范围内。

如前例 , 已知 gamma 值为 2. 2 , 像素 A 的原始值是 200 , 就可求得经 gamma 校正后 A 对应的预补偿值为 228 。基于上述原理 , 我们只需为 0 ~ 255 之间的每个整数执行一次预补偿操作 ,将其对应的预补偿值存入一个预先建立的 gamma 校正查找表 , 就可以使用该表对任何像素值在 0 ~ 255 之间的图像进行 gamma 校正。

Gamma 校正实现代码如下：

```
############gamma 校正查找表
def BuildTable(gamma):
    table=[]
    for i in range(0,256):
        x1=(i+0.5)/256
        x2=1/gamma
        x3=np.power(x1,x2)
        x4=x3*256-0.5
        table.append(x4)
    return table
##############gamma 校正算法
def GammaCorrectiom(img1,gamma):
    mm=BuildTable(gamma)
    m, n = img1.shape
    for i in range(0, m):
        for j in range(0, n):
            img1[i][j] = mm[img1[i][j]]
    return img1
```

查看效果：

3.2.3、DoG 滤波

对于二维图像，由于被摄物体对光线的吸收与反射性能不同，常常会造成光照强度不均匀，Gamma 校正不能消除由人脸局部表面不平滑即表面结构引起的阴影区域的影响。为了消除其影响，可以将图像转换到频域，使用 DoG（高斯差分）滤波的方法来处理。DoG 函数相当于一个高斯函数的二次微分，可表示为中心区与周边区的两个高斯函数响应之差。通过二维傅里叶变换将图像转换到频域，使用具有平滑性能的高通滤波器过滤冗余信息，保存低频有效信息，然后对平滑化后的结果进行二次微分处理，从而进一步修正光照强度的不均匀。

Difference of Gaussian(DOG)是高斯函数的差分。我们已经知道可以通过将图像与高斯函数进行卷积得到一幅图像的低通滤波结果，即去噪过程，这里的 Gaussian 和高斯低通滤波器的高斯一样，是一个函数，即为正态分布函数。

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$$

那么 difference of Gaussian 即高斯函数差分是两幅高斯图像的差，具体到图像处理来讲，就是将两幅图像在不同参数下的高斯滤波结果相减，得到 DoG 图。

核心代码如下：

```
##############DoG 滤波###################
def DoG(img1,sig1,sig2):
    img2= cv.GaussianBlur(img1, (3, 3),sig1) - cv.GaussianBlur(img1, (3, 3), sig2)
    return img2
```

3.2.4、直方图均衡化

直方图是图像中像素强度分布的图形表达方式. 它统计了每一个强度值所具有的像素个数.直方图均衡化是通过拉伸像素强度分布范围来增强图像对比度的一种方法.

利用 OpenCV 函数 equalizeHist 对图像进行直方图均衡化，核心代码如下：

```
src=cv.equalizeHist(src)
```

3.3、定义一个 5 尺度 8 方向的 Gabor 变换

近年来，Gabor 小波在图像处理中得到广泛应用。Gabor 小波是一组窄带通滤波器，在空间域、频率域都有很好的分辨能力，拥有多尺度和多方向特性。

Gabor 小波的核函数与哺乳动物初级视觉皮层的感受野细胞的刺激响应十分相似 ,能够很好地提取目标图像的不同空间位置、频率和方向上的特征。利用 Gabor 小波提取的特征能够克服光照、尺度、角度等全局干扰对识别效果的影响 ,因此 Gabor 小波变换能够很好地解决由于光照强度和人脸表情等变化引起的图像问题，因此在人脸识别领域获得了广泛的应用。

Gabor 小波是一种加窗的傅里叶变换方法，2 维 Gabor 小波定义为：

$$\psi_{\mu,\nu}(z)=\frac{\|k_{\mu,\nu}\|^2}{\sigma^2}\exp\left(\frac{-\|k_{\mu,\nu}\|^2\|z\|^2}{2\sigma^2}\right)\left(\exp(ik_{\mu,\nu}z)-\exp\left(-\frac{\sigma^2}{2}\right)\right).$$

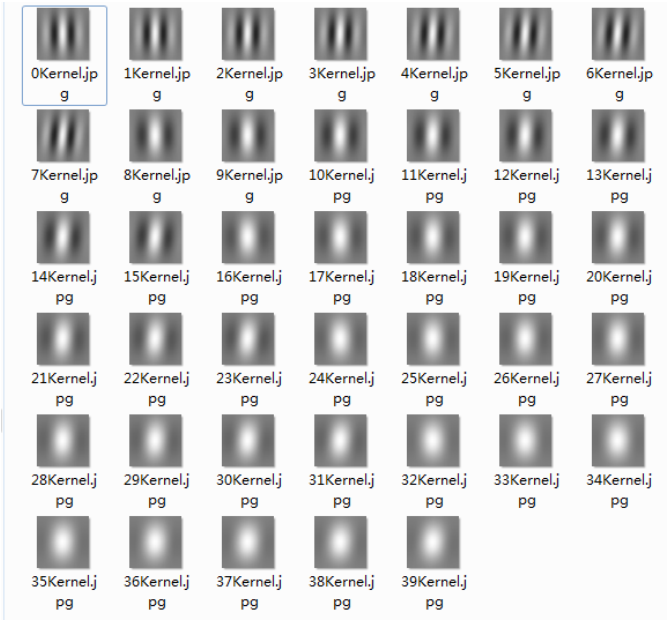
其中，μ和 v 分别表示 Gabor 滤波器的方向和尺度;z(x,y)为图像某一像素点坐标，‖·‖表示范数，σ为高斯包络，$k_{u,v}$控制高斯窗的宽度、震荡部分波长以及方向，定义为

$$k_{u,v}=k_v e^{i\varphi}\mu$$

其中：$k_v=\frac{k_{max}}{f^v}$,为滤波器采样频率，k_{max} 为最大频率， f^v为限制频域中核函距离的间隔因子.

对于人脸纹理特征提取，通常选取 5 个尺度，8 个方向，即 v=0,1,...,4 ,u=0,1...,7。

定义一个 5 尺度 8 方向的 Gabor 变换,效果如下：



3.4、预处理后的图像与 Gabor 滤波器进行卷积

对预处理后的人脸图像 和 Gabor 小波核函数进行卷积：

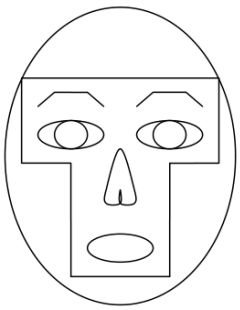
$$I_{\mu,\nu}(z) = I(z) \times \Psi_{\mu,\nu}(z).$$

其中：$I_{u,v}(z)$表示卷积后的结果，I(z)表示预处理后的人脸图像，Ψ(z)表示 Gabor 核幅值特性.对 5 个不同尺度和 8 个方向 Gabor 核幅值变化与图像分别卷积，得到 40 个不同参数的 Gabor 小波图像。

原图

3.5、选取 T 形分布的 Gabor 特征向量

经过 Gabor 小波变换后得到的特征矩阵，含有大量的冗余信息，而且计算量很大，严重影响识别效率，考虑到人脸提取的特征应该主要集中于人脸内部，主要是人脸的五官信息而不是背景信息，因此，采用 T 形分布的 Gabor 特征，不仅可以保留绝大部分重要的 Gabor 特征点，同时很好地降低 Gabor 矢量的特征维数。人脸 T 形分布模型如下图：



人脸 T 形分布模型

本文所采用的选取 T 形分布 Gabor 特征，主要由两部分组成，第一部分选取滤波后图像第 21 到 50 行的中间 72 列 ,包含 2160 个像素点 ;第二部分是第 51 到 100 行的中间 50 列 ,包含 2500 个像素点。所以图像 T 形分布的总像素数为 4660，大约为原图像（112×92=10304）的一半，从而有效地降低了图像的特征维数。而且采用 T 形分布的方法保留了人脸绝大部分重要的特征点，同时还有效地去除了部分干扰信息。

将 40 个 Gabor 变换后的图像分别提取 T 形分布的特征之后 ,转换为一个行向量。整体核心代码如下：

```
#定义了一个 5 尺度 8 方向的 Gabor 变换
def img2vector(image):
    hist1=[]
    #图像预处理
    image=cv.imread(image,1)
    image = cv.resize(image, (92, 112), interpolation=cv.INTER_CUBIC)
    src = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
    src=GammaCorrectiom(src,0.8)
    src=DoG(src,0.9,0.3)
    src=cv.equalizeHist(src)
    ##gabor 变换
    src_f = np.array(src, dtype=np.float32)
    src_f /= 255.
    us=[7,12,17,21,26]                    #5 种尺度
    # vs=[0,30,60,90,120,150]        #6 个方向
    vs=[0,pi/4,2*pi/4, 3*pi/4, 4*pi/4, 5*pi/4, 6*pi/4,7*pi/4]#8 个方向
    kernel_size =21
    sig = 5                              #sigma  带宽，取常数 5
    gm = 1.0                              #gamma  空间纵横比，一般取 1
    ps = 0.0                             #psi  相位，一般取 0
    i=0
    for u in us:
        for v in vs:
            lm = u
            th = v*np.pi/180
            kernel = cv.getGaborKernel((kernel_size,kernel_size),sig,th,lm,gm,ps)
            kernelimg = kernel/2.+0.5
            dest = cv.filter2D(src_f, cv.CV_32F,kernel)
            dst=np.power(dest,2)
            p1 = dst[20:50, 10:82]
            p2 = dst[50:100, 21:71]
            for each in p1:
                for each1 in each:
                    hist1.append(each1)
            for each0 in p2:
                for each2 in each0:
                    hist1.append(each2)

    return hist1
```

3.6、将行向量进行 PCA 降维，得到特征向量

PCA 方法是统计学中一个重要的分析理论，也是被广泛使用的人脸识别方法之一．其方法主要是通过对协方差矩阵进行特征分解，以得到数据的主成分（即特征向量）与它们的权值（即特征值）将高维人脸图像经过 PCA 变换投影到特征子空间，使得数据在一个低维的特征空间里被处理，减少数据冗余的同时保留原始数据的绝大部分信息，解决了数据空间维数过高所产生的计算复杂度等问题。

由于经过 gabor 变换后的向量维数较高，通过主元变换 PCA 用一个低维子空间描述人脸图像，力图在剔除分类干扰分量的同事保留有利于分类的判别信息。

PCA 人脸识别的基本思想就是从人脸图像中找出最能代表人脸的特征空间，去除一些不能代表人脸特征的属性。一个单个的人脸图片映射到这个特征空间得到这个特征空间的一组系数，这组系数就表示这张人脸图片的特征脸特征。如果两张人脸图片映射到这个特征空间的系数差不多，就表示这两张人脸是同一个人。

简单来说，PCA 方法就是将图像投影到训练集经过 K-L 变换所得到的特征空间，然后在投影空间中计算距离。PCA 的作用主要是降低数据集的维度，然后挑选出主要的特征。

算法步骤如下：

(1) 每列减去均值

(2) 计算协方差矩阵（协方差矩阵表示不同随机变量之间的相互关系，图像中也即求任意两个像素之间的关系）

(3) 计算协方差矩阵的特征值和特征向量

(4) 选择主成分(保留最重要的 n 个特征)

训练阶段：

1、载入训练样本人脸图像，分别计算每个样本图像经过 gabor 变换的 T 形特征向量，然后将他们组合成一个特征向量矩阵。核心代码如下：

```
#load dataset
def loadDataSet(k):   #choose k(0-10) people as traintest for everyone
    ##step 1:Getting data set
    print "--Getting data set---"
    #note to use '/'    not '\'
    dataSetDir = 'E:/att_faces'
    #显示文件夹内容
    choose = random.permutation(10)+1 #随机排序 1-10 (0-9）+1
    # print choose
    train_face = zeros((40*k,4660*40))
    train_face_number = zeros(40*k)
    test_face = zeros((40*(10-k),4660*40))
    test_face_number = zeros(40*(10-k))
    for i in xrange(40): #40 sample people
        # print i
        people_num = i+1
        for j in xrange(10): #everyone has 10 different face
```

```
            if j < k:
                filename = dataSetDir+'/s'+str(people_num)+'/'+str(choose[j])+'.pgm'
                img = img2vector(filename)
                train_face[i*k+j,:] = img
                train_face_number[i*k+j] = people_num
            else:
                filename = dataSetDir+'/s'+str(people_num)+'/'+str(choose[j])+'.pgm'
                img = img2vector(filename)
                test_face[i*(10-k)+(j-k),:] = img
                test_face_number[i*(10-k)+(j-k)] = people_num
    return train_face,train_face_number,test_face,test_face_number
```

2、将 gabor 变换后的 T 形特征向量矩阵进行主成分分析 PCA 降维，核心代码如下：

```
# define PCA
def pca(data,k):
    data = float32(mat(data))
    rows,cols = data.shape#取大小
    data_mean = mean(data,0)#对列求均值
    data_mean_all = tile(data_mean,(rows,1))
    Z = data - data_mean_all
    T1 = Z*Z.T #使用矩阵计算，所以前面 mat
    D,V = linalg.eig(T1) #特征值与特征向量
    V1 = V[:,0:k]#取前 k 个特征向量
    V1 = Z.T*V1
    for i in xrange(k): #特征向量归一化
        L = linalg.norm(V1[:,i])
        V1[:,i] = V1[:,i]/L

    data_new = Z*V1 # 降维后的数据
    return data_new,data_mean,V1
```

3、开始训练，核心代码如下：

```
# calculate facefind
def facefind():
    # Getting data set
    # 选择每个人中随机 9 张作为训练集
    train_face,train_face_number,test_face,test_face_number = loadDataSet(9)
    # 选择每个人中随机 9 张作为训练集，其他的作为测试集，并且降维到 30 维
    data_train_new,data_mean,V = pca(train_face,100)
    return   data_mean,V
```

识别阶段：

1、读入一张新的人脸图像，进行图像预处理（归一化，灰度化，gamma 校正，DoG 滤波，直方图均衡化），然后将预处理后的图像进行 Gabor 变换，提取 T 形人脸特征向量，最后连接为一行行向量，作为该图的特征向量。

2、将改图的特征向量到平均脸 avg 的向量差记为 D，D 的大小 1×n

3、D 乘以上面训练得到的特征 LBP 向量 C（n*k）得到这个图片向量 D 在 C 下的投影向量 P ，p 的大小 1×k。

4、得到的投影向量 P 作为余弦相似度输入参数。

核心代码如下：

```
 img11=cv.imread(filename1)
img22=cv.imread(filename2)
img11=cv.resize(img11, (92,112), interpolation=cv.INTER_CUBIC)
img22=cv.resize(img22, (92,112), interpolation=cv.INTER_CUBIC)
img11=cv.cvtColor(img11, cv.COLOR_BGR2GRAY)
img22=cv.cvtColor(img22, cv.COLOR_BGR2GRAY)
imgVector1=img2vector(filename1)
imgVector2=img2vector(filename2)
data_mean,V1=facefind()
imgVector11 = imgVector1 - tile(data_mean, (1, 1))
p1 = imgVector11 * V1
p1 = p1.getA()
for each in p1:
    p11 = each
imgVector22 = imgVector2 - tile(data_mean, (1, 1))
p2 = imgVector22 * V1
p2 = p2.getA()
for each in p2:
    p22 = each
```

3.7、通过余弦夹角计算两个特征向量的相似度

余弦相似度，又称为余弦相似性。通过计算两个向量的夹角余弦值来评估他们的相似度。余弦相似度用向量空间中两个向量夹角的余弦值作为衡量两个个体间差异的大小。余弦值越接近 1，就表明夹角越接近 0 度，也就是两个向量越相似，这就叫"余弦相似性"。

对于二维空间，根据向量点积公式 ，显然可以得知：

$$\cos\theta = \frac{A \cdot B}{\|A\|\|B\|}$$

假设向量 a、b 的坐标分别为(x1,y1)、(x2,y2) 。则：

$$\cos\theta = \frac{x_1 x_{2+} y_1 y_2}{\sqrt{{x_1}^2 + {y_1}^2} \times \sqrt{{x_2}^2 + {y_2}^2}}$$

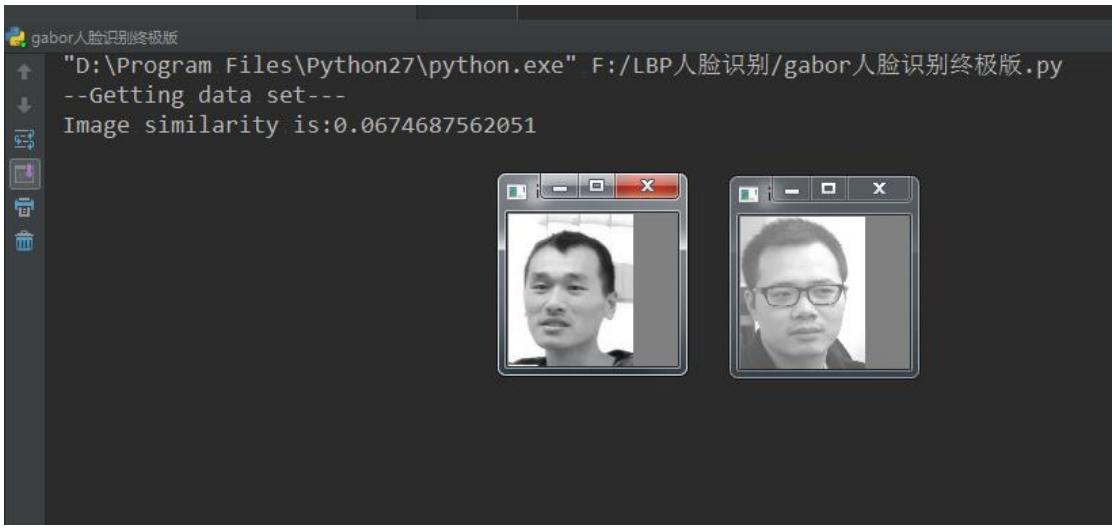
设向量 A = (A1,A2,...,An)，B = (B1,B2,...,Bn) 。推广到多维:

$$\cos\theta = \frac{\sum_1^n (A_i \times B_i)}{\sqrt{\sum_1^n {A_i}^2} \times \sqrt{\sum_1^n {B_i}^2}}$$

核心代码如下：

```
def cos_dist(a, b):
    if len(a) != len(b):
        return None
    part_up = 0.0
    a_sq = 0.0
    b_sq = 0.0
    for a1, b1 in zip(a,b):
        part_up += a1*b1
        a_sq += a1**2
        b_sq += b1**2
    part_down = math.sqrt(a_sq*b_sq)
    if part_down == 0.0:
        return None
    else:
        return   abs((part_up / part_down))
```

效果如下：


gabor人脸识别终极版
"D:\Program Files\Python27\python.exe" F:/LBP人脸识别/gabor人脸识别终极版.py
--Getting data set---
Image similarity is:0.0674687562051


gabor人脸识别终极版
"D:\Program Files\Python27\python.exe" F:/LBP人脸识别/gabor人脸识别终极版.py
--Getting data set---
Image similarity is:0.217729592997


gabor人脸识别终极版
"D:\Program Files\Python27\python.exe" F:/LBP人脸识别/gabor人脸识别终极版.py
--Getting data set---
Image similarity is:0.13925409638


gabor人脸识别终极版
"D:\Program Files\Python27\python.exe" F:/LBP人脸识别/gabor人脸识别终极版.py
--Getting data set---
Image similarity is:0.923348687123

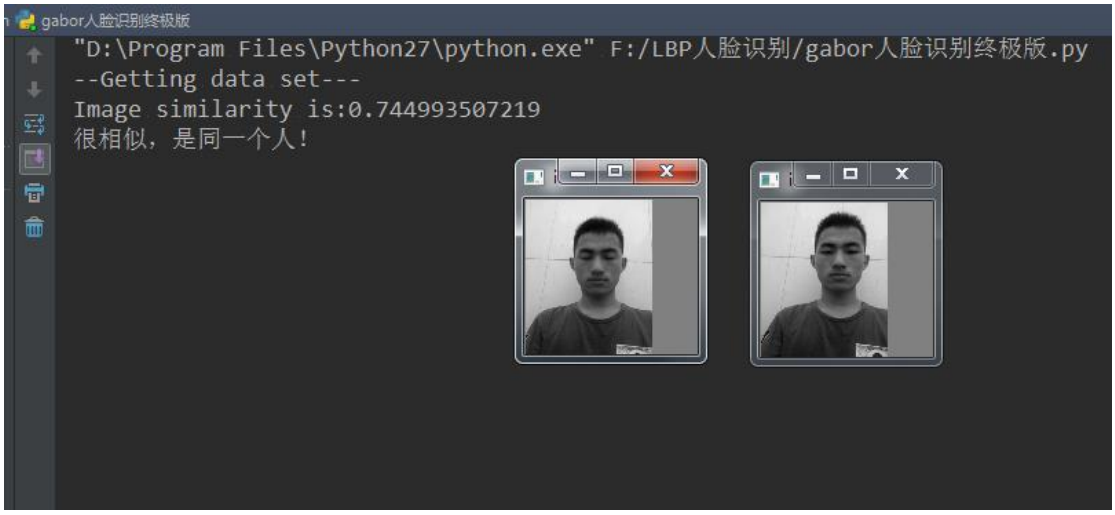

gabor人脸识别终极版
"D:\Program Files\Python27\python.exe" F:/LBP人脸识别/gabor人脸识别终极版.py
--Getting data set---
Image similarity is:0.957357586985


gabor人脸识别终极版
"D:\Program Files\Python27\python.exe" F:/LBP人脸识别/gabor人脸识别终极版.py
--Getting data set---
Image similarity is:0.943561312632


gabor人脸识别终极版
"D:\Program Files\Python27\python.exe" F:/LBP人脸识别/gabor人脸识别终极版.py
--Getting data set---
Image similarity is:0.866285384055


gabor人脸识别终极版
"D:\Program Files\Python27\python.exe" F:/LBP人脸识别/gabor人脸识别终极版.py
--Getting data set---
Image similarity is:0.959279565289
很相似，是同一个人！


gabor人脸识别终极版
"D:\Program Files\Python27\python.exe" F:/LBP人脸识别/gabor人脸识别终极版.py
--Getting data set---
Image similarity is:0.871017415548
很相似，是同一个人！

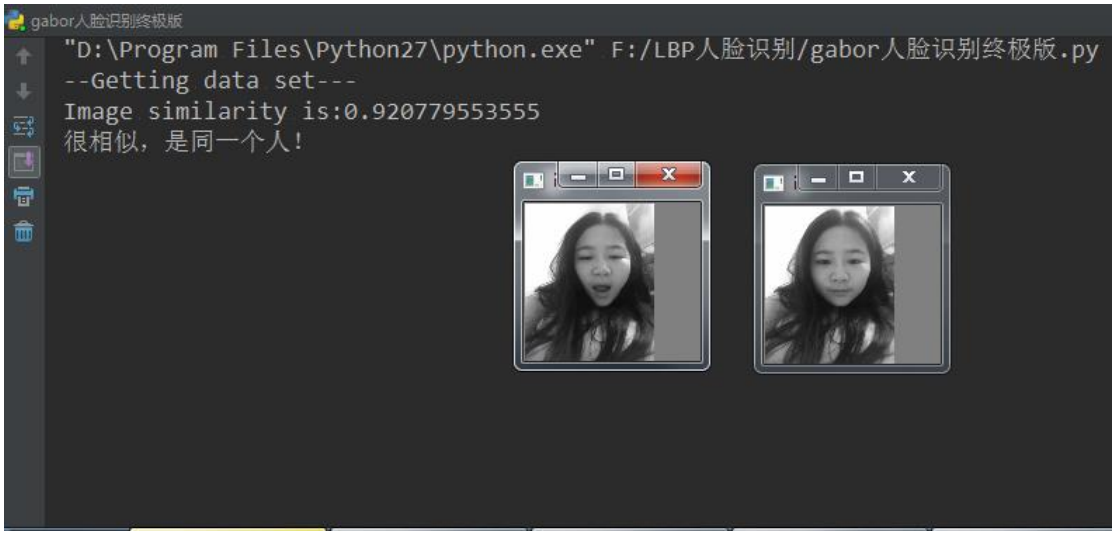

gabor人脸识别终极版
"D:\Program Files\Python27\python.exe" F:/LBP人脸识别/gabor人脸识别终极版.py
--Getting data set---
Image similarity is:0.744993507219
很相似，是同一个人！


gabor人脸识别终极版
"D:\Program Files\Python27\python.exe" F:/LBP人脸识别/gabor人脸识别终极版.py
--Getting data set---
Image similarity is:0.653738195562
很相似，是同一个人！


gabor人脸识别终极版
"D:\Program Files\Python27\python.exe" F:/LBP人脸识别/gabor人脸识别终极版.py
--Getting data set---
Image similarity is:0.230083159769
不太像，不是同一个人！


gabor人脸识别终极版
"D:\Program Files\Python27\python.exe" F:/LBP人脸识别/gabor人脸识别终极版.py
--Getting data set---
Image similarity is:0.779972432753
很相似，是同一个人！


gabor人脸识别终极版
"D:\Program Files\Python27\python.exe" F:/LBP人脸识别/gabor人脸识别终极版.py
--Getting data set---
Image similarity is:0.950730513817
很相似，是同一个人！


gabor人脸识别终极版
"D:\Program Files\Python27\python.exe" F:/LBP人脸识别/gabor人脸识别终极版.py
--Getting data set---
Image similarity is:0.920779553555
很相似，是同一个人！