改进 LBP+PCA 人脸识别算法开发文档 V2

一、主要内容

基于 LBP+PCA 人脸识别算法开发文档 V1 版本的基础上做出改进，本文通过 LBP 等价模式算子的特征提取，将人脸分成 8*8 子区域，然后通过连接这些子区域的 LBP 直方图生成人脸特征向量，由于生成的特征向量的维数过高，通过 PCA 算法降维压缩，最后用余弦夹角或皮尔森相关系数计算相似度，通过实验得出比较好的人脸识别效果。

二、建模过程

2.1、读入人脸图像

```
filename1 = "E:/1017.jpg"
filename2 = "E:/1018.jpg"
image1 = cv2.imread(filename1)
image2 = cv2.imread(filename2)
```

2.2、人脸图像预处理

将读入的人脸图像统一进行灰度化，归一化大小，直方图均衡化操作。核心代码如下：

```
def process_img(image1):
    roi_gray1 = cv2.cvtColor(image1, cv2.COLOR_BGR2GRAY)
    img1 = cv2.resize(roi_gray1, (96, 112), interpolation=cv2.INTER_CUBIC)
    img4 = cv2.equalizeHist(img1)
    return img4
```

效果如下：

2.3、LBP 特征提取

2.3.1、初始 LBP 算法

最初的 LBP 是定义在像素 3x3 邻域内的，以邻域中心像素为阈值，将相邻的 8 个像素的灰度值与其进行比较，若周围像素值大于中心像素值，则该像素点的位置被标记为 1，否则为 0。这样，3x3 邻域内的 8 个点经比较可产生 8 位二进制数（通常转换为十进制数即 LBP 码，共 256 种），即得到该邻域中心像素点的 LBP 值，并用这个值来反映该区域的纹理信息。如下图所示：

63	48	37
79	42	52
48	54	77

图2(a) 局部图像灰度值

1	1	0
1		1
1	1	1

图2(b) 以中心点为阈值二值化处理后的结果

由定义知,中心点的LBP的模式为:

$$LBP_{(8,1)}=(11011111)_2=223$$

2.3.2、LBP 改进-等价模式

为了解决二进制模式过多的问题 ,提高统计性 ,Ojala 提出了采用一种“等价模式”(Uniform Pattern) 来对 LBP 算子的模式种类进行降维。Ojala 等认为，在实际图像中，绝大多数 LBP 模式最多只包含两次从 1 到 0 或从 0 到 1 的跳变。因此，Ojala 将 “等价模式” 定义为：当某个 LBP 所对应的循环二进制数从 0 到 1 或从 1 到 0 最多有两次跳变时，该类型保留；跳变次数超过 2 次时，均归为一类。

通过这样的改进，二进制模式的种类大大减少，而不会丢失任何信息。模式数量由原来的 2P 种减少为 P (P-1)+3 种，其中 P 表示邻域集内的采样点数。对于 3×3 邻域内 8 个采样点来说，二进制模式由原始的 256 种减少为 59 种，这使得特征向量的维数更少，并且可以减少高频噪声带来的影响。

LBP 等价模式核心代码如下：

```
#uniform_map 为等价模式的 58 种特征值从小到大进行序列化编号得到的字典
self.uniform_map={0:0,1:1,2:2,3:3,4:4,6:5,7:6,8:7,12:8,
                  14:9,15:10,16:11,24:12,28:13,30:14,31:15,32:16,
                  48:17,56:18,60:19,62:20,63:21,64:22,96:23,112:24,
                  120:25,124:26,126:27,127:28,128:29,129:30,131:31,135:32,
                  143:33,159:34,191:35,192:36,193:37,195:38,199:39,207:40,
                  223:41,224:42,225:43,227:44,231:45,239:46,240:47,241:48,
                  243:49,247:50,248:51,249:52,251:53,252:54,253:55,254:56,
                  255:57}

#获取图像的 LBP 等价模式特征
def lbp_uniform(self,image_array):
    uniform_array=np.zeros(image_array.shape, np.uint8)
    basic_array=self.lbp_basic(image_array)
    width=image_array.shape[0]
    height=image_array.shape[1]
    for i in range(1,width-1):
        for j in range(1,height-1):
            k= basic_array[i,j]<<1
            if k>255:
                k=k-255
            xor=basic_array[i,j]^k
            num=self.calc_sum(xor)
            if num<=2:
```

```
                uniform_array[i,j]=self.uniform_map[basic_array[i,j]]
            else:
                uniform_array[i,j]=58
    return uniform_array
```

2.3.3、LBP 等价模式直方图

将人脸分成 8*8 子区域，每个子区域分别统计其 LBP 直方图，并且直方图归一化大小，核心代码如下：

```
##直方图归一化
def MaxMinNormalization(hist1):
    x1=[]
    sum1=np.sum(hist1)
    for i in range(0,len(hist1)):
        x =hist1[i]/sum1
        x1.append(round(x,4))
    return x1

##等价模式直方图
def lbp_uniform_hist(gray1):
    lbp = LBP()
    basic_array1 = lbp.lbp_uniform(gray1)
    k1 = []
    hist11 = []
    hist1 = cv2.calcHist([basic_array1], [0], None, [59], [0,59])
    for each in hist1:
        k1.append(int(each))
        for each in k1:
            hist11.append(int(each))
    hist11 = MaxMinNormalization(hist11)
     return hist11
```

获取每个子区域 LBP 直方图之后，连接这些子区域的 LBP 直方图生成人脸特征向量，核心代码如下：

```
def get_imageLBP_unoform_histall(img1):
    q1=[]
    img0 = cv2.resize(img1, (96, 112), interpolation=cv2.INTER_CUBIC)
    m=112
    n=96
    img_lbp_hist1=[]
    stepx=14
    srepy=12
    for i in range(0,m,stepx):
        for j in range(0,n,srepy):
```

```
                roi=img0[i:i+stepx,j:j+srepy]
                m1=lbp_uniform_hist(roi)
                img_lbp_hist1.append(m1)
    for each in img_lbp_hist1:
        for each1 in each:
                q1.append(each1)
    return q1
```

2.3.4 对 LBP 特征向量进行提取的步骤

（1）首先将检测窗口划分为 8×8 的小区域（cell）；
（2）对于每个 cell 中的一个像素，将相邻的 8 个像素的灰度值与其进行比较，若周围像素值大于中心像素值，则该像素点的位置被标记为 1，否则为 0。这样，3*3 邻域内的 8 个点经比较可产生 8 位二进制数，即得到该窗口中心像素点的 LBP 值；
（3）然后计算每个 cell 的等价模式直方图，即每个数字（假定是十进制数 LBP 值）出现的频率；然后对该直方图进行归一化处理。
（4）最后将得到的每个 cell 的统计直方图进行连接成为一个特征向量，也就是整幅图的 LBP 纹理特征向量；

2.4 PCA 降维

由于 LBP 直方图向量维数较高，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的 LBP 等价模式直方图，将每个图像的子区域 LBP 连接成特征向量之后，按照一个样本一行排列，就得到训练样本 LBP 特征向量矩阵。核心代码如下：

```
def img2vector2(filename):
    img = cv2.imread(filename)
    img1=image_processing(img)
    imgVector =get_imageLBP_unoform_histall(img1)
```

```
        return imgVector

    def loadDataSet(m,k):
        dataSetDir = 'E:/att_faces'
        choose = random.permutation(10) + 1
        train_face = zeros((m * k,8*8*59))
        for i in xrange(m):
            print i
            people_num = i + 1
            for j in xrange(10):
                if j < k:
                    print "第"+str(j)+"个"
                    filename = dataSetDir + '/s' + str(people_num) + '/' + str(choose[j]) +
'.pgm'
                    img = img2vector2(filename)
                    train_face[i*k+j, :] = img

        return train_face,people_num
```

2、将 LBP 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
        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：

```
    def facefind():
        train_face, people_num=loadDataSet(40,9)
        data_train_new, data_mean, V = pca(train_face,30)
        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（灰度化，直方图均衡化）之后，将图像划分 8*8 子区域，计算每个子区域的直方图，最后连接每个子区域的直方图为一行，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 作为余弦相似度或者皮尔森相关系数的输入参数。

核心代码如下：

```
img1 = process_img(image1)
img2 = process_img(image2)
imgVector1 = get_imageLBP_unoform_histall(img1)
print imgVector1
imgVector2 = get_imageLBP_unoform_histall(img2)
print imgVector2
data_mean, V1 = facefind()
imgVector11 = imgVector1 - tile(data_mean, (1, 1))
p1 = imgVector11 * V1
p1 = p1.getA()
for each in p1:
    p11 = each
    print p11
imgVector22 = imgVector2 - tile(data_mean, (1, 1))
p2 = imgVector22 * V1
p2 = p2.getA()
for each in p2:
    p22 = each
    print p22
```

2.5、相似度计算

2.5.1 余弦夹角相似度

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))
```

2.5.2 皮尔森相关系数

皮尔森相关系数（Pearson correlation coefficient）也叫皮尔森积差相关系数（Pearson product-moment correlation coefficient），是用来反应两个变量相似程度的统计量。或者说可以用来计算两个向量的相似度。

两个变量之间的皮尔逊相关系数定义为两个变量之间的协方差和标准差的商。计算公式如下：

$$\rho_{X,Y} = \frac{\sum XY - \frac{\sum X \sum Y}{N}}{\sqrt{\left(\sum X^2 - \frac{(\sum X)^2}{N}\right)\left(\sum Y^2 - \frac{(\sum Y)^2}{N}\right)}}$$

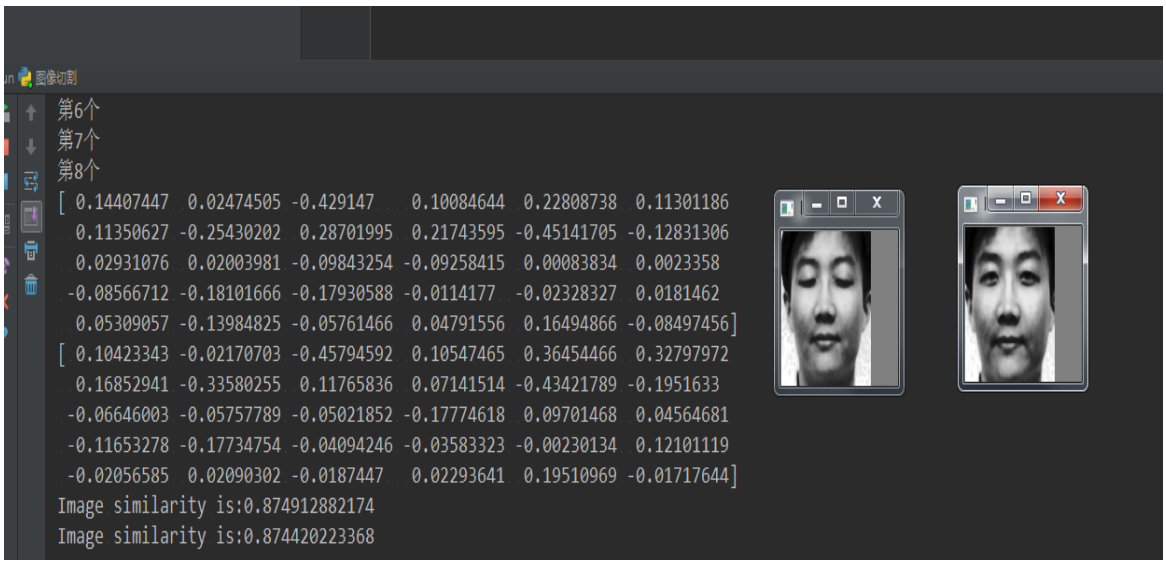
实现代码如下：

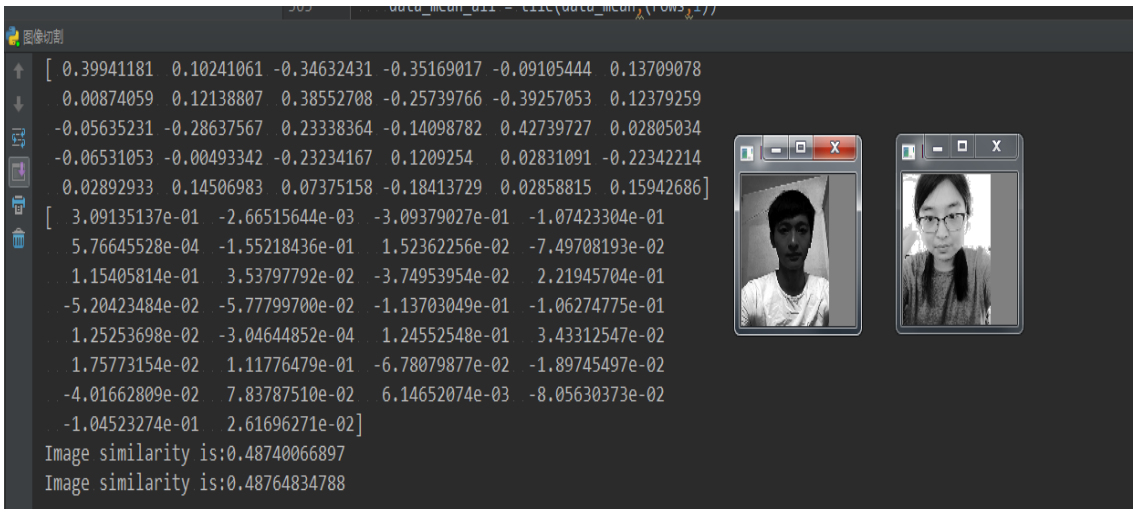
```
def multipl(a,b):
    sumofab=0.0
    for i in range(len(a)):
        temp=a[i]*b[i]
        sumofab+=temp
    return sumofab

def corrcoef(x,y):
    n=len(x)
```

```
#求和
sum1=sum(x)
sum2=sum(y)
#求乘积之和
sumofxy=multipl(x,y)
#求平方和
sumofx2 = sum([pow(i,2) for i in x])
sumofy2 = sum([pow(j,2) for j in y])
num=sumofxy-(float(sum1)*float(sum2)/n)
#计算皮尔逊相关系数
den=sqrt((sumofx2-float(sum1**2)/n)*(sumofy2-float(sum2**2)/n))
return num/den
```

效果如下：

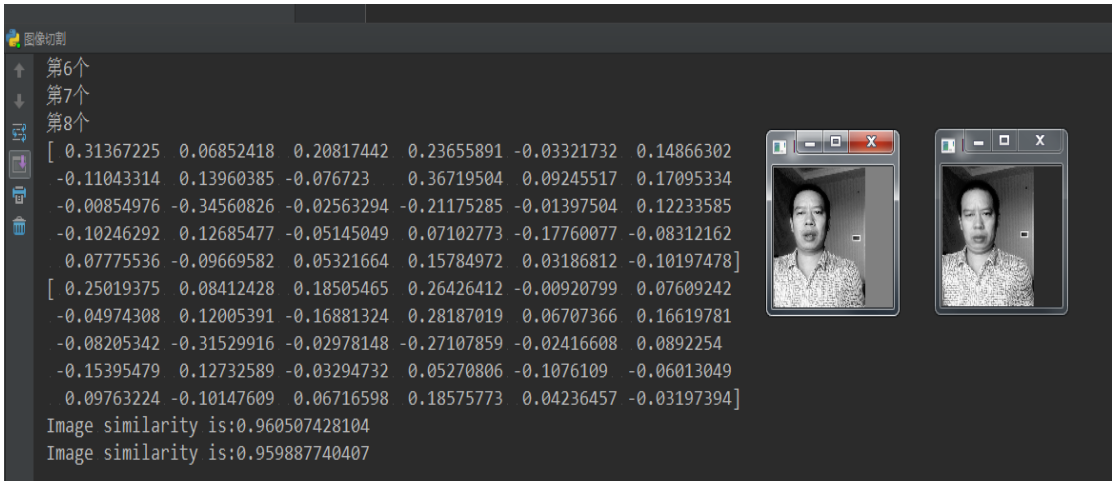


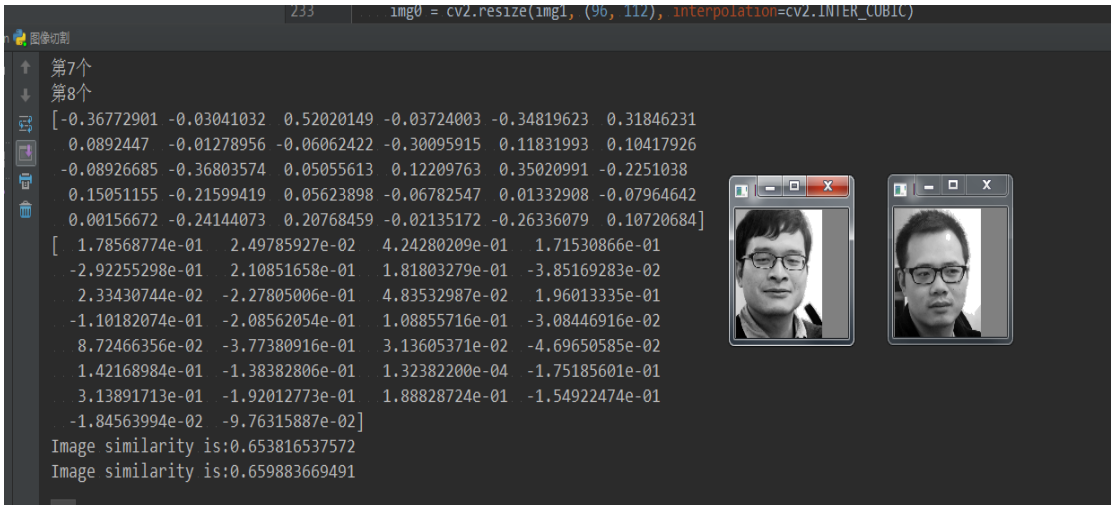


```
408        train_face, people_num=loadDataSet(40,9)
图像切割
第6个
第7个
第8个
[ 0.27119968  0.07828229 -0.35170753  0.09901887  0.03214036  0.07572361
  0.09003303 -0.10185238 -0.11917762  0.19739373 -0.07759397 -0.0091899
 -0.10978783  0.22280092 -0.06770382  0.09016713 -0.1016636   0.12384857
  0.00936371 -0.06695103 -0.17907351 -0.02900024 -0.2309512   0.04492862
 -0.0357286  -0.04675717 -0.05767037 -0.03446853  0.05326676 -0.06581215]
[ 0.23135626  0.07532711 -0.32226026  0.12852732  0.10535639  0.07460604
  0.05786867 -0.14208122 -0.08436476  0.22836796 -0.06545756 -0.05044216
 -0.05372076  0.09150211 -0.04725202  0.14677325 -0.16020135  0.10778482
  0.0158924  -0.10047203 -0.20621279 -0.00138011 -0.25566161  0.02320197
  0.06108424 -0.01681033 -0.06368523 -0.04063139  0.06497256 -0.04602909]
Image similarity is:0.942326630537
Image similarity is:0.94212389226
```

```
图像切割
第6个
第7个
第8个
[ 0.28989258  0.14907597 -0.46929757  0.03376906  0.23151418 -0.11240049
  0.04478061  0.05070348  0.07976529 -0.01135673  0.06691868  0.30157322
  0.0271242  -0.2195377  -0.14059093 -0.01572558 -0.06900423 -0.06708911
  0.14859731 -0.00309507 -0.17345592 -0.16451747 -0.12911097  0.16135846
  0.22885361 -0.09637643 -0.13522016  0.11706033 -0.16397644 -0.03142095]
[ 0.28609014  0.12765326 -0.37624671  0.10858358  0.23189736 -0.08946717
 -0.02510142  0.06271581  0.07672194  0.01650272  0.06563085  0.31759749
  0.08086746 -0.19145875 -0.15759913  0.02198572 -0.09263925 -0.08403879
  0.20847886 -0.00230851 -0.15873974 -0.08350834 -0.08722034  0.07096195
  0.11586156 -0.11450719 -0.09928989  0.10535704 -0.09556682 -0.0226645 ]
Image similarity is:0.960522312186
Image similarity is:0.96246193143
```

```
图像切割
第6个
第7个
第8个
[ 0.29837001 -0.1962143  -0.46392073  0.01377    -0.26359301 -0.10005862
  0.03411732 -0.05779035  0.07913156 -0.00157715 -0.07756853  0.22682833
  0.22031994  0.06725651 -0.13437188  0.11182578 -0.17775335  0.04822816
  0.16866293 -0.18724227  0.04745704 -0.15065964 -0.1357578  -0.10785072
  0.01293537 -0.19708086  0.07928346  0.15026235  0.04921385  0.11496709]
[ 0.52939074  0.08988901 -0.30937835 -0.18473929  0.00405738 -0.17872855
 -0.14214669  0.15085056 -0.11217314  0.23137165 -0.07963985 -0.06110121
 -0.01852562 -0.26190934 -0.1118247  -0.00259207 -0.0117131  -0.0937475
  0.03843245 -0.03238669 -0.04569062 -0.10732337 -0.16737296 -0.17817064
 -0.15310272  0.06610329 -0.08152334 -0.04336852 -0.06326315  0.0849081 ]
Image similarity is:0.383359105337
Image similarity is:0.37029639085
```


图像切割
第6个
第7个
第8个
[0.31367225 0.06852418 0.20817442 0.23655891 -0.03321732 0.14866302
 -0.11043314 0.13960385 -0.076723   0.36719504 0.09245517 0.17095334
 -0.00854976 -0.34560826 -0.02563294 -0.21175285 -0.01397504 0.12233585
 -0.10246292 0.12685477 -0.05145049 0.07102773 -0.17760077 -0.08312162
  0.07775536 -0.09669582 0.05321664 0.15784972 0.03186812 -0.10197478]
[0.25019375 0.08412428 0.18505465 0.26426412 -0.00920799 0.07609242
 -0.04974308 0.12005391 -0.16881324 0.28187019 0.06707366 0.16619781
 -0.08205342 -0.31529916 -0.02978148 -0.27107859 -0.02416608 0.0892254
 -0.15395479 0.12732589 -0.03294732 0.05270806 -0.1076109 -0.06013049
  0.09763224 -0.10147609 0.06716598 0.18575773 0.04236457 -0.03197394]
Image similarity is:0.960507428104
Image similarity is:0.959887740407


233 img0 = cv2.resize(img1, (96, 112), interpolation=cv2.INTER_CUBIC)
图像切割
第7个
第8个
[-0.36772901 -0.03041032 0.52020149 -0.03724003 -0.34819623 0.31846231
  0.0892447 -0.01278956 -0.06062422 -0.30095915 0.11831993 0.10417926
 -0.08926685 -0.36803574 0.05055613 0.12209763 0.35020991 -0.2251038
  0.15051155 -0.21599419 0.05623898 -0.06782547 0.01332908 -0.07964642
  0.00156672 -0.24144073 0.20768459 -0.02135172 -0.26336079 0.10720684]
[1.78568774e-01   2.49785927e-02   4.24280209e-01   1.71530866e-01
  -2.92255298e-01   2.10851658e-01   1.81803279e-01 -3.85169283e-02
   2.33430744e-02 -2.27805006e-01   4.83532987e-02   1.96013335e-01
  -1.10182074e-01 -2.08562054e-01   1.08855716e-01 -3.08446916e-02
   8.72466356e-02 -3.77380916e-01   3.13605371e-02 -4.69650585e-02
   1.42168984e-01 -1.38382806e-01   1.32382200e-04 -1.75185601e-01
   3.13891713e-01 -1.92012773e-01   1.88828724e-01 -1.54922474e-01
  -1.84563994e-02 -9.76315887e-02]
Image similarity is:0.653816537572
Image similarity is:0.659883669491