手部静脉图像预处理算法实验研究

一、选题背景

随着科学技术的不断突破，人们身上越来越多的生物特征被运用到各行业之中。比如常见的指纹识别，人脸识别，虹膜识别、声纹识别、静脉识别等。生物识别技术，通过计算机与光学、声学、生物传感器和生物统计学原理等高科技手段密切结合，利用人体固有的生理特性（如指纹、脸象、虹膜等）和行为特征（如笔迹、声音、步态等）来进行个人身份的鉴定。

（一）人脸识别技术

人脸识别应用最为广泛，使用最便捷，人脸识别系统的研究始于 20 世纪 60 年代，80 年代后随着计算机技术和光学成像技术的发展得到提高，而真正进入初级的应用阶段则在 90 年后期，并且以美国、德国和日本的技术实现为主。如今，人脸识别技术的应用已经不仅限在商务场所中，它已经以各种智能家居的形式逐步渗透到平常百姓家。

缺点：人脸识别系统信息存储仍是以计算机能识别的语言为主，即数字或特定代码，安全性便要打折了。人脸识别在具备较高便利性的同时，其安全性也相对较弱一些。识别准确率会受到环境的光线、识别距离等多方面因素影响;另外，当用户通过画妆、整容对于面部进行一些改变时也会影响人脸识别的准确性。

（二）指纹识别技术

每个指纹都有几个独一无二可测量的特征点，每个特征点都有大约七个特征，人们的十个手指产生最少 4900 个独立可测量的特征。指纹识别技术通过分析指纹可测量的特征点，从中抽取特征值，然后进行认证。当前，我国第二代身份证便实现了指纹采集，且各大智能手机都纷纷实现了指纹解锁功能。与其他生物识别技术相比，指纹识别早已经在消费电子、安防等产业中广泛应用，通过时间和实践的检验，技术方面也在不断的革新。

缺点：虽然每个人的指纹识别都是独一无二的，但并不适用于每一个行业、每一个人。例如，双手长期手工作业的人们便会为指纹识别而烦恼，他们的手指若有丝毫破损或干湿环境里、沾有异物则指纹识别功能要失效了。

（三）虹膜识别生物

相对于其他生物识别技术而言，虹膜识别误识率和拒真率已经达到了零几率的识别水平，而虹膜识别又属于非接触式的识别，识别方便高效。虹膜是每个人特有的，具有不可复制的唯一性，安全等级来说是目前最高的。

缺点：但是虹膜识别的应用价格也因其技术难度成正比，相比其他的识别技术，略显贵态。

（四）声纹识别

所谓声纹，是用电声学仪器显示的携带言语信息的声波频谱。它非常适合远程身份确认，只需要一个麦克风或电话、手机就可以通过网路(通讯网络或互联网络)实现远程登录。

缺点：不过，声纹识别的缺点也十分明显，对环境的要求非常高，在嘈杂的环境、混合说话下声纹不易获取；人的声音也会随着年龄、身体状况、年龄、情绪等的影响而变化；不同的麦克风和信道对识别性能有影响等。

（五）步态识别

是通过人的走路方式来进行身份识别的生物特征识别技术。步态识别是一种非接触的生物特征识别技术。因为它不需要人的行为配合，特别适合于远距离的身份识别，这是任何生物特征识别所无法比拟的，不容易伪装。是让犯罪分子防不胜防的追捕手段，它不仅可以分析闭路电视捕捉到的嫌犯的行动情况，还能把它们同嫌犯走路的姿态进行比较。在一些凶杀案中，往往凶犯不让你看到他们的脸，但却能看到凶手走路的样子。采集装置简单、经济。因为只需要一个监控摄像头就行。

缺点：正面识别率低；容易受到性别、步长、节奏、速度等的干扰；相机角度、天气条件、甚至衣服光照等都会影响准确性；步态识别易受生病、道路崎岖、衣服光照等的影响。步态识别技术难以获取的信息，如年龄、性别等信息。远距离低分辨率以及背景灰暗造成人脸识别较困难。非普遍性，如残疾人不适合步态识别技术。

（六）静脉识别

再就是我们这次实验所做的静脉识别，静脉识别是基于静脉血管中的纹理特征进行身份识别的一种生物识别技术。静脉识别系统就是首先通过静脉识别仪取得个人静脉分布图，从静脉分布图依据专用比对算法提取特征值，通过红外线 CCD 摄像头获取手指、手掌、手背静脉的图像，将静脉的数字图像存贮在计算机系统中，将特征值存储。

静脉识别具有高度防伪、简便易用、快速识别及高度准确四大特点。最为重要的一点是，指静脉识别的特征已被国际公认具有唯一性，且和视网膜相当，在其拒真率(相同结构图，而被算法识别为不同)低于万分之一的情况下，其识假率(不同结构图，而被算法识别为相同)可低于 10 万分之一。

但它同样有着难以规避的缺点:

1.手背静脉仍可能随着年龄和生理的变化而发生变化，永久性尚未得到证实;

2.仍然存在无法成功注册登记的可能;

3.由于采集方式受自身特点的限制，产品难以小型化;

4.采集设备有特殊要求，设计相对复杂，制造成本高。

在应用方面：

①考勤和门禁：传统的考勤市场几乎是指纹识别的天下，但是随着其它生物识别技术的发展，目前这种状况正在慢慢改变。静脉识别凭借其非接触的优势已经逐渐介入考勤市场。据悉，目前这一市场的容量已经达到每年近 10 亿人民币的规模；中国的家庭安防市场现在仍处于起步阶段，与智能家居领域类似，所有的企业都在探索，但可以预计未来的市场规模很大。中国现有城镇居民家庭约 2 亿户，未来 5 年预计至少有 5%的家庭会考虑在家中安装安防产品，平均每年将有 20 亿元左右的市场需求。

②司法应用：主要包括司法鉴证系统。2014 年，公安部部级的生物识别采购金额为 819.2 万元，2015 年采购金额为 899.78 万元，2016 年采购金额最少 1500 万元。如果地方公安机关的采购额能够实现相似的增速，生物识别将在公安行业将迎来翻倍增长。

③公共项目应用：国内医疗市场容量预计 9 亿元；

教育包括考勤、考试、校园、接送、食堂、图书馆的市场容量为 27 亿元；
公共交通如地铁、公交、飞机、游轮等；
涉密单位如银行、机械仓库、军工行业等；
社会保险及财政补贴，全国人社部门生物识别市场的空间约为 45 亿元；
监狱，从市场需求来看，全国拥有司法 675 座监狱、3200 座看守所、350 所劳教所的巨大市场；
旅游景区、社区及小区物业管理、智能电梯、社区保安巡逻的市场容量约为 18 亿元。
④公共与社会安全应用：主要包括证照系统、出入境控制系统、会议及办事大厅身份认证系统、黑名单追踪系统、敏感岗位任职人员背景调查系统、房产交易系统等政务市场应用。
⑤金融：金融设备的配套设备
ATM、POS 机上的掌静脉识别模块，根据国外的发展趋势，这将是一个很大的市场。
无卡消费和会员管理
掌静脉身份识别将会为会员提供一个安全、便捷的消费方式，是金融领域最大的应用市场。目前已在商场、无人超市、高端会员制场所应用。

二、方案论证(设计理念)

（一）手指部分：
1. 根据实验室器材采集手指信息，并获得图像
2. 为了更好的处理图像信息，我们需要进行预处理，包括①手指轮廓分割②将手指图像旋转至统一方向（如水平），这里需要检测中线③根据需要提取感兴趣区域
3. 将预处理后的图像进行静脉纹理增强（图像增强），并进行静脉纹理分割以便观察
4. 对图像增强后的感兴趣区域图像进行①二值纹理特征提取②LBP 特征提取
5. 根据实验进行图像匹配①二值化模板匹配②LBP 直方图特称匹配
6. 根据匹配等实验结果进行分析，并总结创新实践经验
（二）手掌部分：

步骤 2. 图像预处理部分：①获取手掌前景图，用 OTSU 算法分割前背景，提取手部轮廓。②获取手腕中心基准点，假设手掌为竖直方向，在二值化图像基础上，从下往上遍历每一行，直到出现白色像素点，该行往上挪指定值作为手腕基准线，再遍历该行获取两边颜色跳变点，取平均值作为中点。③进行轮廓边缘查找，并且获取最大面积的轮廓点集。④提取峰谷点，以手腕中心 Pref 为基准点，计算基准点以上轮廓点到基准点水平线的距离，获得 5 个局部极大点和 4 个局部极小点，与指尖和指缝一一对应。⑤获取食指与中指中间的点和无名指与尾指中间的点作为 ROI 截取的基准点，计算两点连线与水平线的夹角得到旋转角度，对图片进行旋转矫正。⑥从 ROI 截取基准线截取正方形的手掌 ROI。

其余步骤同手指。

三、过程论述

（一）简介：

静脉生物特征识别系统由静脉图像采集系统和静脉图像识别系统构成。静脉图像采集系统中，近红外光源以反射式、透射式或侧射式的打光方式，照射手指、手掌或手背，光线进入皮下组织后，浅层皮下静脉血管中的血红蛋白吸收近红外光，从而在红外图像传感器上形成阴影，得到静脉图像。如图 1 为采集静脉图像示例。

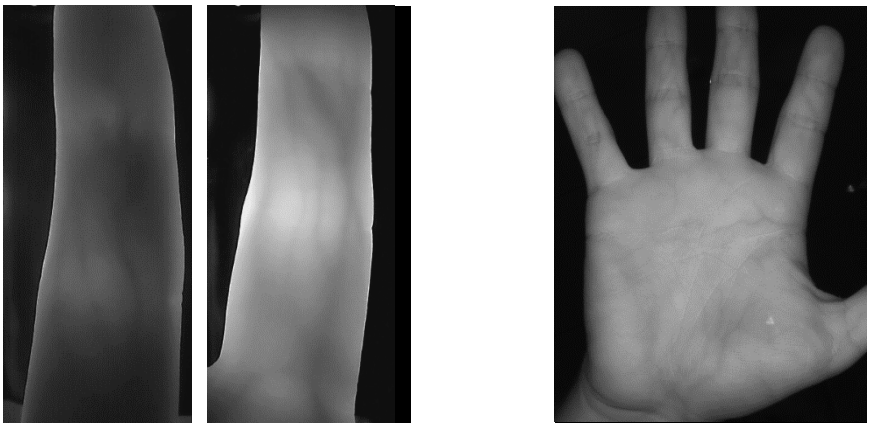


图 1 手指和手掌静脉图片示例

静脉图像识别系统则一般由感兴趣区域(ROI)截取、静脉纹理增强、静脉纹理分割、特征提取和特征匹配组成。

本课程要求深入了解手部静脉生物特征识别的各个环节，重点研究手指和手掌静脉图像的预处理算法，具体为手指静脉图像和手掌静脉图像的手部轮廓分割、感

兴趣区域截取、静脉纹理增强、静脉纹理分割。最后分别使用 LBP 特征和二值纹理特征对预处理后的静脉图像进行测评，获得两张静脉图像的相似度。根据实验结果，分析影响匹配相似度分数的原因，并改进预处理算法。

图示见上。

（二）手部边缘提取，图像预处理：

手指轮廓和手掌轮廓分割往往作为 ROI 截取的第一个环节，是稳定截取 ROI 的关键。此外，手部形状作为生物特征的一种，也可以用于身份识别。但要从复杂背景中鲁棒地分割出手部轮廓十分困难，目前手部轮廓分割算法一般分三步：①前背景分割，如使用 OTSU 算法得到阈值，进行阈值二值化得到手部前景；②剔除小区域，由于背景噪声二值化图像往往存在一些小区域，使用数学形态学操作剔除小区域；③形状对齐，为便于识别或 ROI 截取，将手部轮廓与标准手形轮廓进行对齐操作。具体操作见下一步。

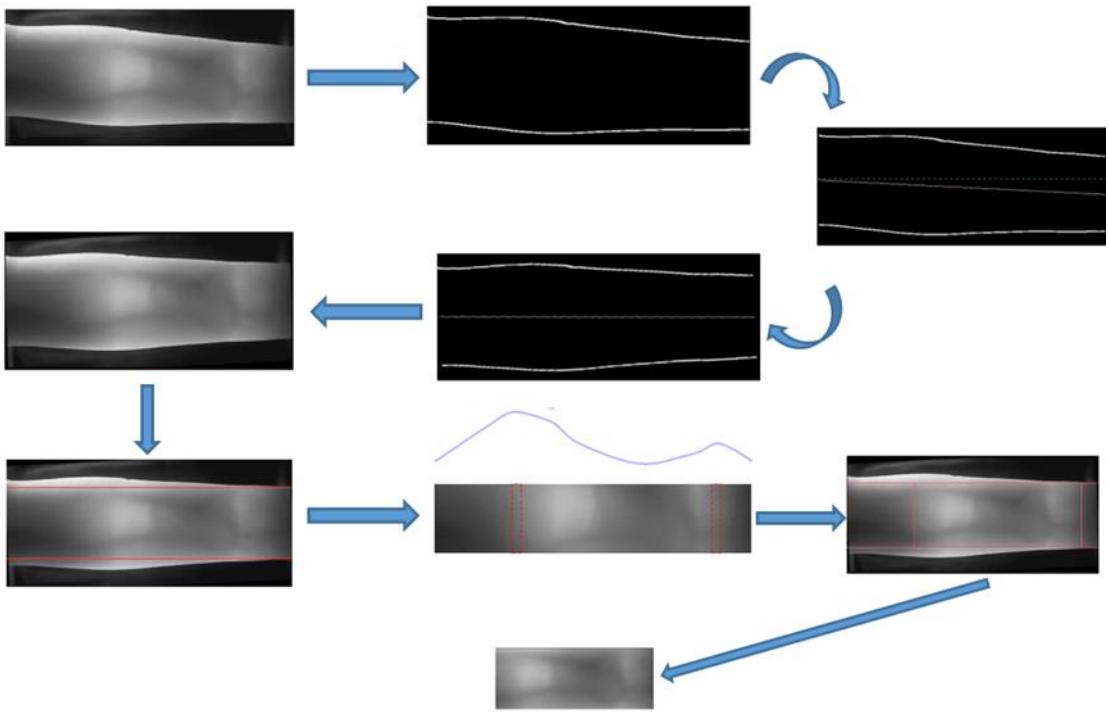
（三）截取 ROI 区域：

1.手指部分：

静脉采集过程中，手指和手掌的自由度较大，会出现平移、旋转等现象，导致同一个手指或手掌两次采集的图像出现差异；采集图片以手作为前景区域和以环境为背景区域，而背景区域对识别没有任何贡献，甚至会很大程度上影响特征提取的正确率。因此在使用手部静脉做身份认证和识别时，需截取采集图像中手指和手掌的 ROI 区域作为输入图像进行认证和识别。

常用手指 ROI 区域截取算法步骤如下：①对图像进行手指边缘检测。②根据手指边缘拟合手指中线，求出手指旋转角度，进行旋转角度校正。③根据已检测的手指边缘确定 ROI 区域上下边界，确定手指关节位置（由于手指的组织结构，手指关节处往往亮度比较高），根据手指关节位置确定 ROI 区域左右边界，获得 ROI 区域。

如图所示：



具体解析：

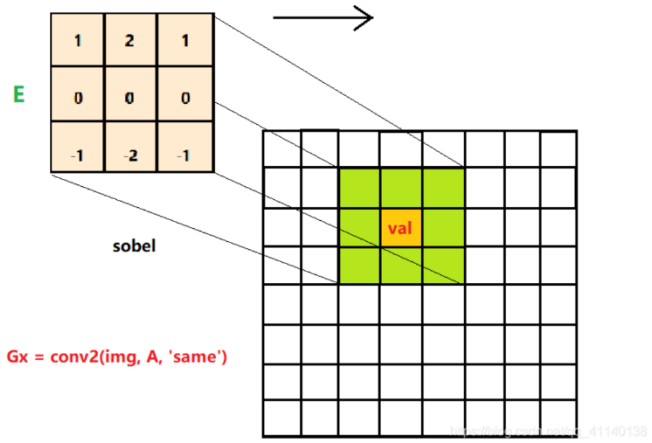
1. 利用边缘检测画出边缘：

我们这里用到的是 canny 算法，其主要步骤包括：高斯滤波、梯度计算、非极大值抑制以及双阈值检测。

使用高斯滤波的目的是平滑图像，滤除图像中的部分噪声（因为微分算子对噪声很敏感）。高斯滤波具体办法是生成一个高斯模板，使用卷积进行时域滤波。

可以利用 sobel 算子进行梯度计算：

一幅图像可以表示为函数 I = f(x, y)，其中(x, y)为坐标，I 表示该像素点的灰度值，梯度 gradient 表示函数 f(x, y)在点(x, y)处最大的变化率，计算的表达式为：$G(x) = \frac{df(x,y)}{dx} + \frac{df(x,y)}{dy}$，对于图像，我们也可以计算梯度，由于数字图像是有离散的像素点的灰度值构成，所以微分运算就变成了差分，我们可以用相邻两个像素点之间的差分值表示该像素点在某个方向上灰度的变化情况。由于图像在边缘的变化情况很剧烈，而在非边缘处变化平缓，所以计算一幅图像的梯度得到的结果中，图像的边缘将被凸显出来,sobel 算子是性能不错的微分算子，下图描述了使用 sobel 算子计算梯度的卷积过程：



上图所示为一个卷积过程（由于 sobel 算子做一个 180 旋转后和旋转前差别不大，因此可以用卷积替代相关运算），橙色的矩阵 E 为 sobel 算子，它可以用来计算 x 方向上的灰度变化，如果我们把矩阵 E 与图像中某个像素及其 8 邻域的元素构成的矩阵的对应元素相乘，可以得到该像素点在 x 方向的差分，所有像素点在 x 方向上的差分构成矩阵 Gx，同理我们也可以计算得到 y 方向上的差分 Gy，

于是梯度 G 的大小为：$G = (Gx^2 + Gy^2)^{\frac{1}{2}}$

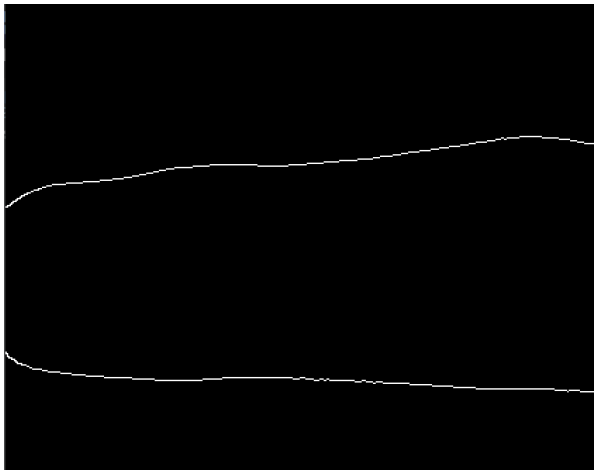
非极大值抑制目的是为了细化边缘，在上面的图像中，梯度计算得到的边缘很粗，一条边缘中央一般很亮，两边亮度逐渐降低，可以根据这个特点去掉非局部灰度最高的“假边”，达到细化边缘的目的。

双阈值检测的目的是减少伪边缘点非极大值抑制之后，检测到的边缘线条比较多，我们可以滤掉一些很暗的边缘，并让主要的边缘凸显出来，主要步骤有：①设置两个阈值 threshold ： threshold_low 和 threshold_high②当某个像素点值高于 threshold_low 时，则可以认为它是边缘，把它的灰度置为 1③当某个像素点值高于 threshold_low 时，则认为它不是边缘，把它的灰度置为 0④处于 threshold_low 和 threshold_high 之间的像素点，如果它的八邻域有真边上的点，则认为它也是边缘，并把灰度置为 1。

代码如下图所示：

```python
def get_edges(img):  # img : single channel img which has been processed by cv.Canny
    rows, cols = img.shape
    center = rows / 2
    index = []
    up_index = []
    for i in range(cols):
        temp_index = np.argwhere(img[:, i] == 255)
        temp_index = temp_index.tolist()
        for j in temp_index:
            if j[0] < center:
                up_index.append(center - j[0])
        up = len(up_index) - 1
        up_index.clear()
        down = up + 1
        try:
            index.append([temp_index[up][0], i])
        except IndexError:
            index.append([index[-2][0], i])
        try:
            index.append([temp_index[down][0], i])
        except IndexError:
            index.append([index[-2][0], i])
    result_img = np.zeros((rows, cols), dtype="uint8")
    for j in index:
        result_img[j[0], j[1]] = 255  # put the Corresponding coordinate into 255
    return result_img
```

我们找一张手指图片运行代码，结果如下：成功检测出了它的边缘（注：代码这里进行的函数编写和判断是，仅仅得到 canny 的效果还不够，因为我们要的仅仅是手指的轮廓，其他位置的任何轮廓都是多余的。所以需要一个函数用来专门提取我们需要的部分，try 的部分的意义是：如果当列的上半部分或者下半部分没有找到灰度值为 255 的像素点，那么就以最近添加的两个像素点的坐标代替。来保证边缘的准确性和光滑性）



2. 根据边缘点得到中点坐标：

```python
# 得到中点坐标
def get_center_line(image):
    rows, cols = image.shape
    code = []
    for i in range(cols):
        temp_index = np.argwhere(image[:, i] == 255)
        temp = temp_index.tolist()
        center_point = round((temp[0][0] + temp[1][0]) / 2)
        temp.clear()
        code.append([center_point, i])
    points = np.array(code)
    return points
```

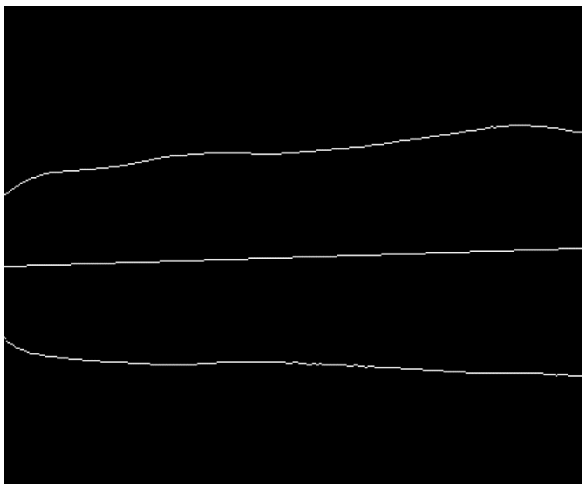
函数这里是进行了一个循环，对于边缘图像，每一列有两个白色的点，记录下这两个点的位置，并取中点，这就是这一列我们要求的中点坐标，历遍每一列，就得到了一组中点坐标即为所求。

3. 将中点拟合成一条直线，并将其与边缘一起画出：

这里我们用到的是 opencv 自带的函数 fitline（），之后再利用 line()将这条直线画在我们之前得到的边缘图像上，输出结果

```python
# 将中线虚拟呈直线并在边缘图上画出
c_test = cv.imread('05.jpg', 0)
sobel_x = cv.Sobel(c_test, cv.CV_64F, 1, 0, ksize=5)
laplacian = cv.Laplacian(c_test, cv.CV_64F)
canny = cv.Canny(c_test, 50, 100)
finger_edges = get_edges(canny)
points = get_center_line(finger_edges)
line_vir = cv.fitLine(points, cv.DIST_L2, 0, 0.01, 0.01)
# print(line_vir)
line = cv.line(finger_edges, (int(line_vir[2] - 1000 * line_vir[1] / line_vir[0]), int(line_vir[2] - 1000)),
               (int(line_vir[2] + 1000 * line_vir[1] / line_vir[0]), int(line_vir[2] + 1000)), 255)
```

这一步骤与上一步骤运行结果如图：

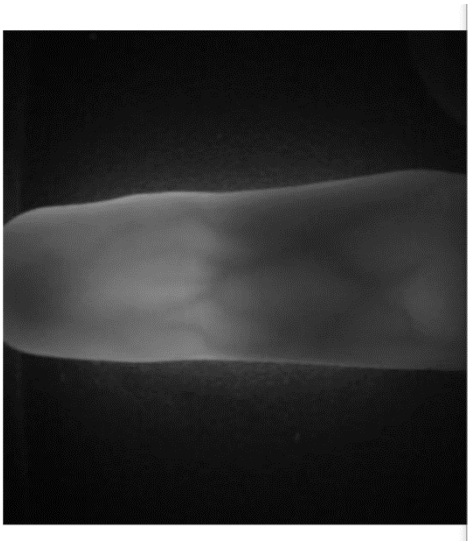


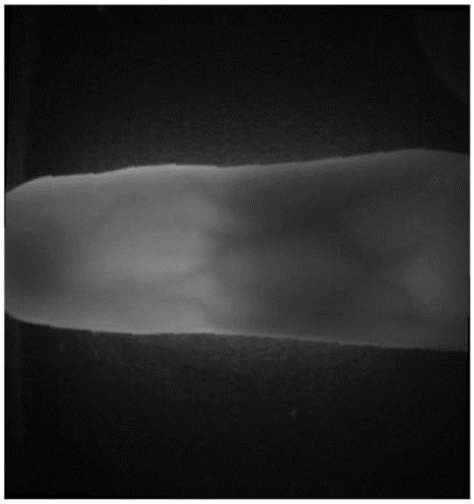
4. 根据中线与水平线的夹角旋转图片至水平：

根据拟合直线函数 fitline（）的输出求出这条直线的斜率，从而得到它距水平线的角度，将图片旋转相应的角度后输出显示，并将这张图片保存。

```
# 根据中线角度旋转图形
k = line_vir[1] / line_vir[0]
angle = math.atan(k)
im = Image.open("05.jpg")
im.show()
im1 = im.rotate(angle)
im1.show()
cv.imshow("line", line)
im1.save("tt.jpg")
cv.waitKey(0)
cv.destroyAllWindows()
```

旋转后运行结果如下，很明显看到图片进行了旋转：





5. 提取 ROI 区域，并保存 ROI 图片：

```
pic_test = cv.imread('tt.jpg', 0)
rows, cols = pic_test.shape
gray_map = pic_test[240]
sobel_x_pic = cv.Sobel(pic_test, cv.CV_64F, 1, 0, ksize=5)
laplacian_pic = cv.Laplacian(pic_test, cv.CV_64F)
canny_pic = cv.Canny(pic_test, 50, 100)
finger_edges_pic = get_edges(canny_pic)
points_pic = get_center_line(finger_edges_pic)
# print(points_pic)
line_vir_pic = cv.fitLine(points_pic, cv.DIST_L2, 0, 0.01, 0.01)
```

首先将旋转后的图片进行边缘处理，并找到它的中线（重复之前的步骤）

```
def find_peak(arr):
    peaks = []
    step = 1
    pos = 40
    while pos < 400:  # 实际上两个peak主要分布在100-200和450-550
        if (arr[pos] >= arr[pos + step]) and (arr[pos] > arr[pos - step]):
            if (arr[pos] > arr[pos + 20]) and (arr[pos] > arr[pos - 20]):  # 这个if语句可以滤除很多的噪音peak
                peaks.append(pos)
        pos = pos + step
    peak1 = 0
    peak2 = 300  # 取的图片中间靠后的位置，但是有可能存在后面的关节腔灰度值比中间位置低的情况
    # print(peaks)
    for i in peaks:  # 这个循环从好多个peak中找到我们要的两个最大的peak
        if i < 200:
            if arr[i] >= arr[peak1]:
                peak1 = i
        else:
            if arr[i] >= arr[peak2]:
                peak2 = i
    return peak1, peak2
```

定义一个函数，找到图像的灰度值，发现有两个极大值，这两个位置就是图像上比较亮的地方，即为关节处，是我们要找的两个位置，那么怎么确定这两个位置呢？这里用的是登山算法，并利用简单的判断过滤一些噪声，得到这两个位置。

```
divide_point = find_peak(gray_map)
# print(divide_point)
points_pic_1 = line_vir_pic[3]
min_x = int(divide_point[0] - 30)
max_x = int(divide_point[1] + 20)
min_y = int(points_pic_1 - 60)
max_y = int(points_pic_1 + 60)
pic_image = pic_test[min_y:max_y, min_x:max_x]
cv.imshow("ROI", pic_image)
cv.waitKey(0)
cv.destroyAllWindows()
cv.imwrite("roi.jpg", pic_image)
```

截取 ROI 区域，我们用中线向两边扩展得到上下边界，用上述函数得到的关节坐标得到左右边界，从而裁剪出矩形 ROI 区域。

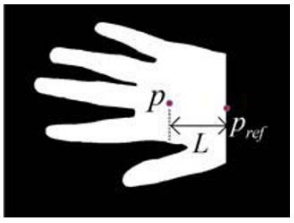
截取后的 ROI 图像如下图所示，并将图片保存。

2.手掌部分：

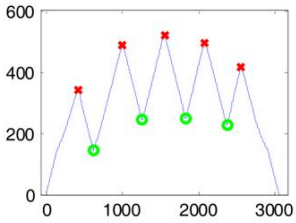
ROI 区域截取算法步骤如下：①用 OTSU 算法分割前背景，提取手部轮廓。②手腕剔除，从右往左每列的所有像素值不断清零，直至右端到手掌轮廓中心点距离为 L。③提取峰谷点，以手掌轮廓右端的中点 Pref 为基准点，该基准点与所有手掌前景轮廓计算径向距离函数(Radial Distance Function，RDF)，求函数极值得到 5 个极大点和 4 个极小点，与指尖和指根一一对应。③ROI 截取基准线确定，通过指尖和指根点确定 ROI 截取基准线 $\overline{p_1 p_2}$ 。④通过 ROI 截取基准线与垂直方向夹角，旋转图片使基准线垂直。⑤从 ROI 截取基准线截取正方形的手掌 ROI。实验步骤如下图所示：

(a)手掌静脉图片

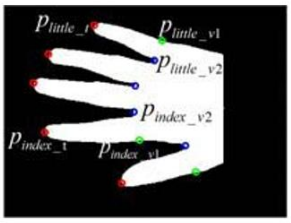
(b)前背景分割



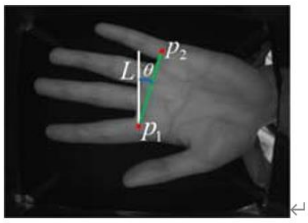
(c)手腕剔除



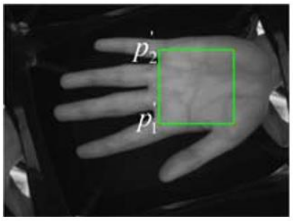
(d)提取峰谷点



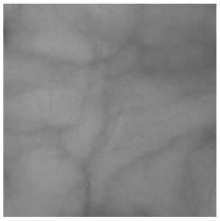
(e)标记指尖和指根点



(f)确定 ROI 截取基准线



(g)确定 ROI 区域



(h)截取 ROI

1）获取手腕中心基准点：

函数 count_size(),假设手掌为竖直方向，在二值化图像基础上，从下往上遍历每一行，直到出现白色像素点，该行往上挪 step 值作为手腕基准线，再遍历该行获取两边颜色跳变点，取平均值作为中点。step 的确定要根据整个手掌的大小，所以还要计算从最高指尖到手腕的距离，方法分别是从上到下和从下到上遍历每一行，直到出现白色像素点。

2）获取所有轮廓：

函数 getMaxRegion（），调用 opencv 库函数获取所有轮廓，再计算所有轮廓的面积，得到面积最大的即为我们要的手掌轮廓集。由于库函数的输出是 list，我们还需要做转化，转化为二维数组的轮廓点集。

3）提取手掌轮廓的峰谷点：

以手腕中心 Pref 为基准点，计算基准点以上轮廓点到基准点水平线的距离，获得 5 个局部极大点和 4 个局部极小点，与指尖和指缝一一对应。首先找到手掌最左和最右的点及对应下标。由于我们获取到的轮廓是逆时针的，并且第一个点为最高点（即中指指尖），所以我们要进行两边循环，循环范围分别是第一个点到手掌最左点，和手掌最右点到点集结果。当获取齐九个点之后，结束循环。

在判断极大值极小值的时候，引入一个标志位 raise_flag，为 1 表示当前遍历的趋势是上升（指根到指尖），为 0 表示当前遍历的趋势是下降（指尖到指根）。结合 python-opencv 的图片坐标，图片左上点为原点，故更高的点其实行坐标值更小。我们定义极值点：若该点的行坐标低于后 10 个点的行坐标，并且当前为上升趋势，若符合限制条件，则说明后续为下降趋势，即该点为极大值点；反之则为极小值点。由于轮廓点集的第一个点是中指指尖，所以遍历的时候先将第一个点放入峰谷点结果集；限制条件为放入结果集的点，其索引在轮廓点集中至少要相隔 100，与结果集中最后的点相隔至少 30 行，这里两个数字为可调参数，根据实际情况调整。

最后我们再将得到的 9 个峰谷点按照从右到左的顺序排列，方便后续处理。

4）旋转矫正图像：

矫正流程为：获取食指与中指中间的点和无名指与尾指中间的点作为 ROI 截取的基准点，计算两点连线与水平线的夹角得到旋转角度，对图片进行旋转矫正。

上一小节中得到的 9 个峰谷点为从右到左排序，在此基础上我们判断左右手，以

便正确找到基准点。若第一个点和第三个点的距离大于第七个点和第九个点，则为右手，第四个点和第八个点为我们的基准点；反之则为左手，第二个点和第六个点是我们的基准点。

获取基准点之后，根据两点坐标计算斜率，调用库函数 atan 获取倾斜角，最后调用 cv2.getRotationMatrix2D 和 cv2.warpAffine 两个函数让原图像绕第一个基准点旋转。

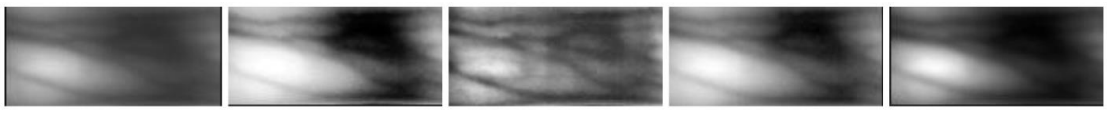
5）截取 ROI 区域：

由于上一小节中绕着第一个基准点旋转，故该点坐标保存不变，该点作为 ROI 正方形的右上角顶点，计算两个基准点之间的距离作为边长，截取出正方形 ROI 区域。

（四）静脉纹理增强：

静脉纹理增强有很多方法，如：线性变换、分段线性变换、伽马变换、直方图正规化、直方图均衡化、局部自适应直方图均衡化等。

获取的原始图片由于成像机理、相机参数、光照变化、手移动及旋转等因素会使得获取的静脉图像存在噪声、模糊、光照不均等图像质量不佳的问题，会很大程度上影响实验效果。图像增强是一种基本的图像预处理手段，主要目的是针对一幅给定的图像，经预处理后，突出图像中的某些信息，削弱或去除某些无用和干扰信息，增强后的图像更有利于特定图像任务的处理。因此采用一种或多种图像增强方法来增强静脉纹理信息，以便后续的特征提取、匹配等过程。常用图像增强方法有全局直方图均衡化、限制对比度自适应直方图均衡化(CLAHE)、Retinex 和灰度归一化等，实验效果可参考下图：



a) 原图　b)全局直方图均衡化　c) CLAHE　d) retinex　e)灰度归一化

经过检验，我们采用直方图正规化的的方法得到的效果最好，方法简介如下：

假设输入图像为 I，宽为 W，高为 H，I(r,c)代表 I 的第 r 行第 c 列的灰度值，将 I 中出现的最小灰度级记为 I_{min}，最大灰度级记为 I_{max}，即 I(r,c) \in [I_{min},I_{max}]，为使输出图像 O 的灰度级范围为 [O_{min},O_{max}]，I(r,c)和 O(r,c)做以下映射关系：

$$O(r,c)=\frac{O_{max}-O_{min}}{I_{max}-I_{min}}(I(r,c)-I_{min})+O_{min},0\leqslant r<H,0\leqslant c<W$$

这个过程就是直方图正规化，直方图正规化是一种自动选取 a 和 b 的值的线性变换方法，其中

$$a=\frac{O_{max}-O_{min}}{I_{max}-I_{min}},b=O_{min}-\frac{O_{max}-O_{min}}{I_{max}-I_{min}}*I_{min}$$

代码如下：

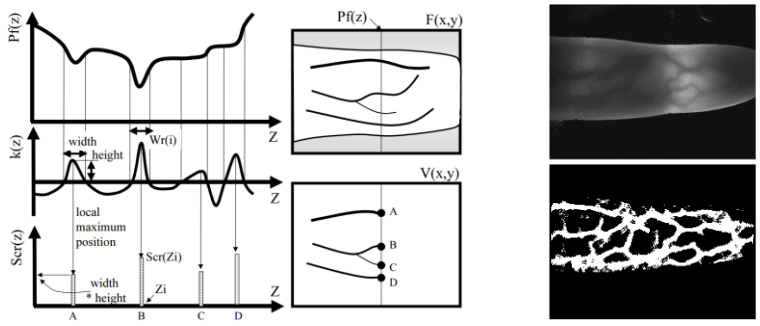
```
img = cv.imread("roi.jpg", 0)
I_min, I_max = cv.minMaxLoc(img)[:2]
O_min, O_max = 0, 255
# 计算a和b的值
a = float(O_max - O_min) / (I_max - I_min)
b = O_min - a * I_min
out = a * img + b
out = out.astype(np.uint8)
cv.imshow("img", img)
cv.imshow("out", out)
cv.waitKey(0)
cv.destroyAllWindows()
cv.imwrite("roi_strong.jpg", out)
```

运行结果如下：

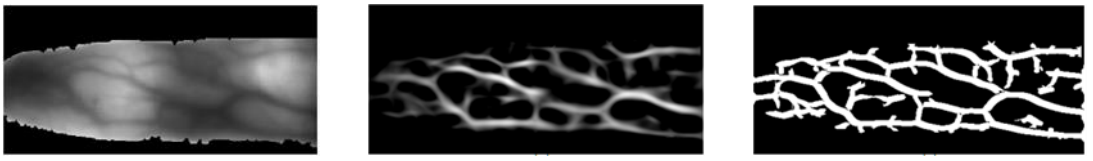
可以发现图像明显得到了增强。

（五）静脉纹理分割：

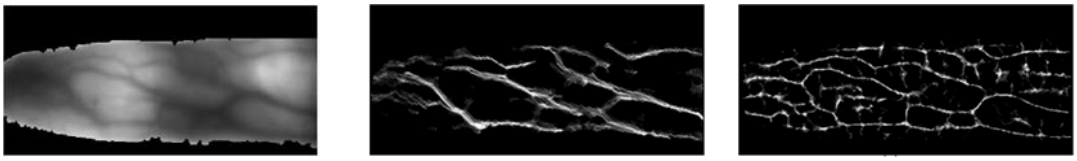
静脉纹理是静脉图像中最关键的身份特征信息，但影响静脉图像中静脉纹理的因素很多，主要有：光线在皮下组织散射导致纹理对比度低和模糊、环境光照变化、手部姿态变化和身体状况差异等，因此要鲁棒、准确、真实地分割出静脉纹理十分困难。除一般图像分割算法外，研究者专门设计了一些经典的静脉纹理分割算法，如图像增强再二值化[1]、宽线检测(Wide LineDetector)[2]、重复线跟踪(repeated line tracking)[3]、最大曲率(maximum curvature)[4]、主曲率(principal curvature)[5]、平均曲率(mean curvature)[6]等。但在实际使用中，对某些静脉图像仍难以很好分割静脉纹理。最大曲率是非常经典的静脉纹理分割算法，①用局部曲率极大值确定横截面上静脉纹理的中心点，如图 1 所示，静脉横截面上灰度值的低谷可认为是静脉纹路的中心点，使用局部曲率极大值求解。对图片水平、垂直及对应 45° 旋转后图片做相同的横截面求静脉中心点，得到 4 个方向的曲率极值合并得到的图片；②滤波剔除噪声影响，连接静脉中心点；③设置阈值对二值化静脉纹理。



试实现几种静脉分割算法，分析实验结果并改进。实验效果可参考图 2：



(a) 原图　　(b) Gabor 增强　　(c) 二值化



(d) 原图　　(e) 重复线跟踪算法　　(f) 最大曲率算法

图 1 几种经典的静脉纹理分割算法效果图

（六）图像特征提取：

特征提取是静脉识别中非常关键的环节，对识别性能有直接影响。现有静脉特征提取算法主要可分为三类，即基于纹理的特征提取、基于编码的特征提取、基于细节点的特征提取。下面介绍对应的三种最简单最常用的特征提取算法：

二值纹理特征提取。首先截取原始图像的 ROI 区域，然后使用多方向 Gabor 滤波器对 ROI 区域进行图像增强，再使用二值化算法将静脉纹理二值化，从而提取得到二值纹理特征，示例如图 3。

局部二进制模式（LBP）特征提取。LBP 特征提取算法是一种高效的纹理特征提取算法，具有亮度不变性和旋转不变性。算法的核心思想在于统计每个像素点周围纹理的灰度变化，将这种变化量转化为一个数值，用该数值代替该像素点的灰度值，就得到了一张 LBP 编码图。目前，LBP 的一些改进方法主要有旋转不变 LBP、Uniform LBP 等，其中 Uniform LBP 使用最为频繁。对 ROI 进行预处理后，提取 ROI 区域的 Uniform LBP 编码图，将该编码图划分为 $m*n$ 个 block，统计每个 block 的 Uniform LBP 编码的直方图，最后对所有直方图进行幅值归一化，并串接得到静脉图像的 LBP 直方图特征。示例如图 4。

局部不变特征提取。局部不变特征最早用于图像配准与拼接，其主要思想是检测图像中的特征点，并对特征点的局部区域进行描述。常用的局部不变特征提取算法有 SIFT、RootSIFT、SURF 等。局部不变特征提取的示例如图 5。

分析各种特征提取方法在静脉识别中的优缺点，配合匹配算法进行分析验证。

图 3 二值化纹理特征提取示例

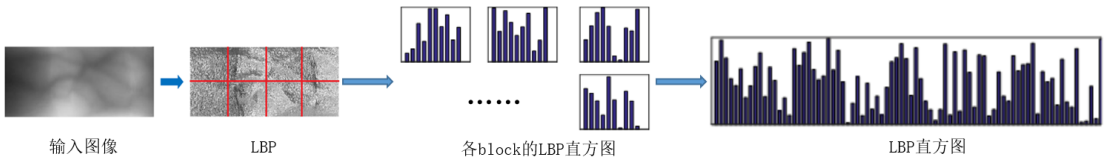


图 4 LBP 直方图特征提取示例

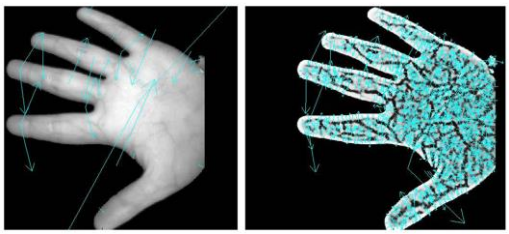


图 5 局部不变性特征提取示例

我们开始采用了二值化的方法，但结果不理想（如图注所示），以至于无法进行下一步操作，所以放弃了这种方法：

图注 二值化提取特征

由前面几个步骤，我们可以得到四组文件，即手指 1 的处理结果，手指二的处理结果，手掌一的处理结果，手掌二的处理结果：

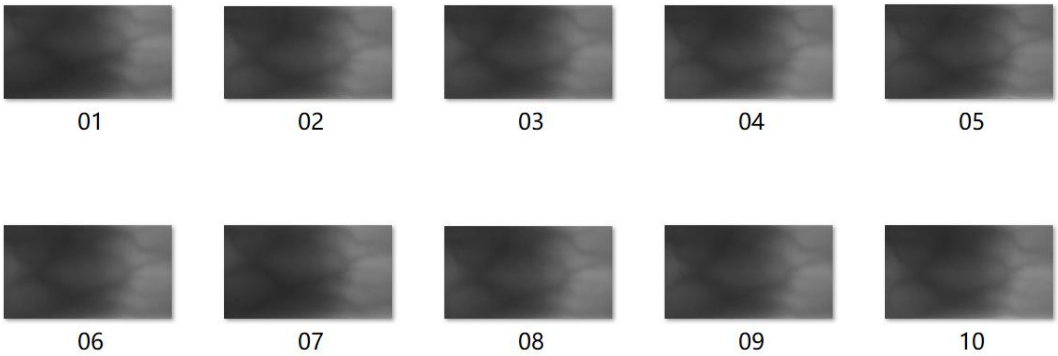


图 6 手指 1 的处理结果

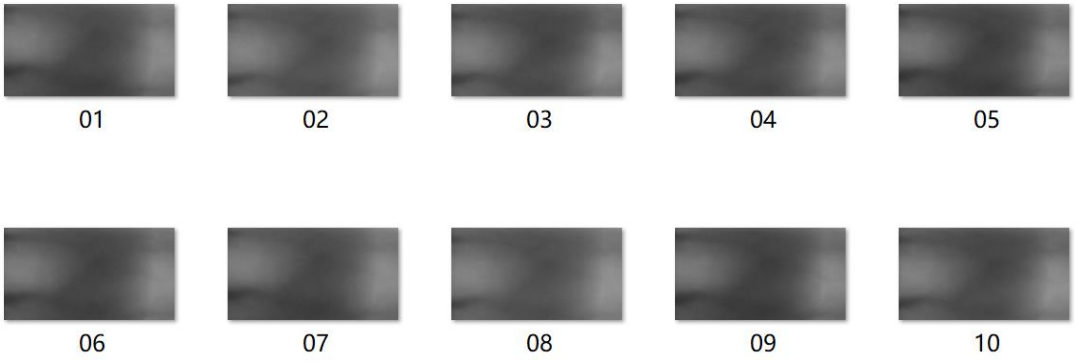


图 7 手指 2 的处理结果

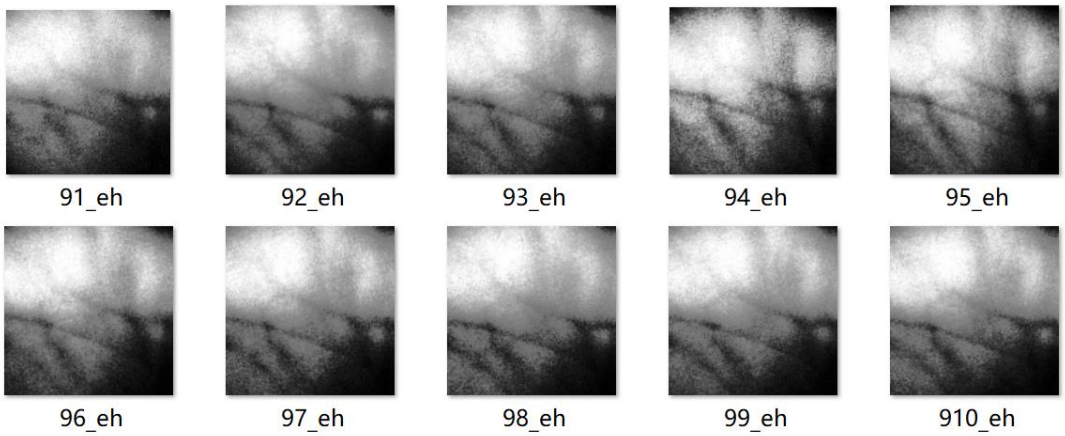


图 8 手掌 1 的处理结果

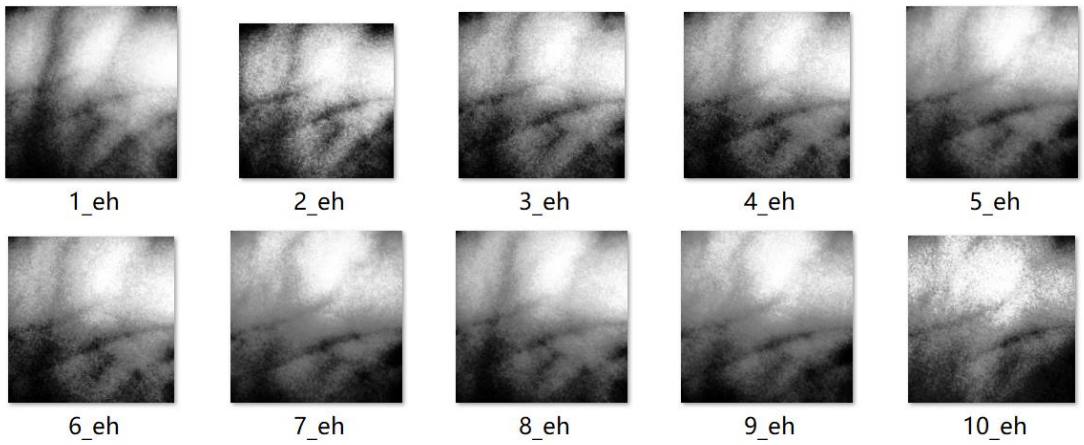


图 9 手掌二的处理结果

（七）图像匹配：

图像匹配是进行静脉识别的最后一步，不同的提取特征类型往往有其对应的匹配方法，而合适的匹配方法能够极大提高识别精度。现有静脉图像匹配算法常

采用某种距离去度量类内样本匹配分数和类间样本匹配分数。若两样本特征的距离小于某阈值，则认为是同一类，反之，则不是同一类。希望所使用匹配算法能使样本的同类样本间距离（类内距离）越小越好，类间样本间距离（或类间距离）越大越好，如此样本更易区分。由于图像匹配需与提取特征配合，因此结合上述特征提取方法，可参考如下匹配方法：

模板匹配，根据上述二值纹理特征提取方法，获取手指静脉纹路的二值纹理图像，即纹理前景图像。将已注册图像的 ROI 区域与待认证图片的 ROI 区域均采用二值纹理特征提取方法获得两张纹理前景图像，然后计算两张纹理前景图像的交集（重叠）像素点数与两张纹理前景图像的并集像素点数之比，这个比值即为两张图像的匹配分数。

直方图相似性匹配，可以采用任何一种合适的直方图比较方法进行匹配，现主要介绍直方图相交法，即根据直方图统计的重合度判断直方图间的距离。令 H_1，H_2 分别为两幅待匹配图像的统计直方图，则基于直方图相交法的 LBP 直方图匹配分数定义为：

$$\text{score}_{LBP} = \sum_{i=1}^{m} \min(H_1^{(i)}, H_2^{(i)}) \tag{1}$$

局部不变特征点匹配，提取局部不变特征点后，将两幅图像的局部特征点进行匹配，计算两张图片中配对成功点对数与特征点数较多的图片中所包含的特征点数目之比，这个比值即为两张图像的匹配分数。

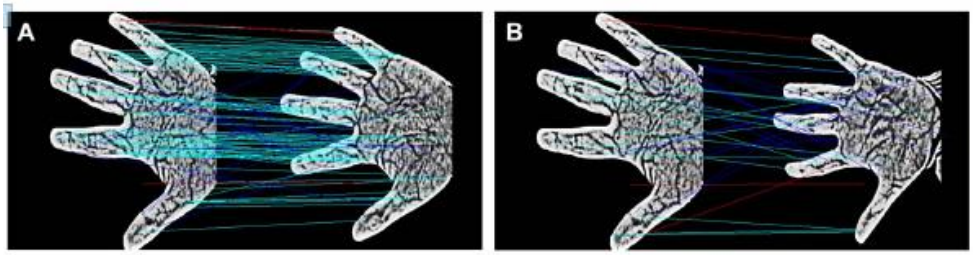
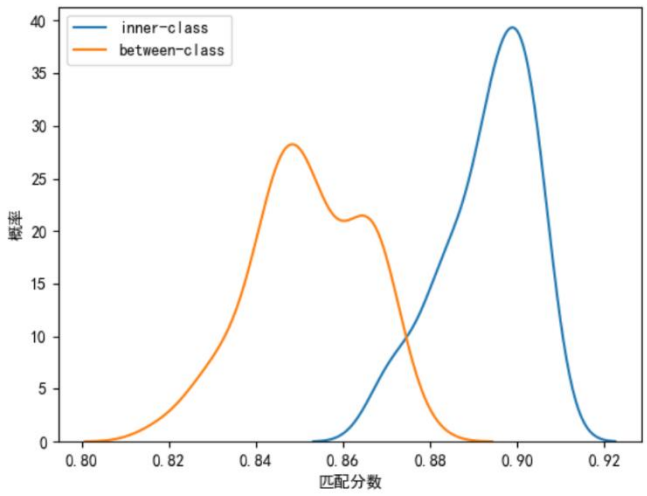


图 6 局部不变性特征点匹配示例

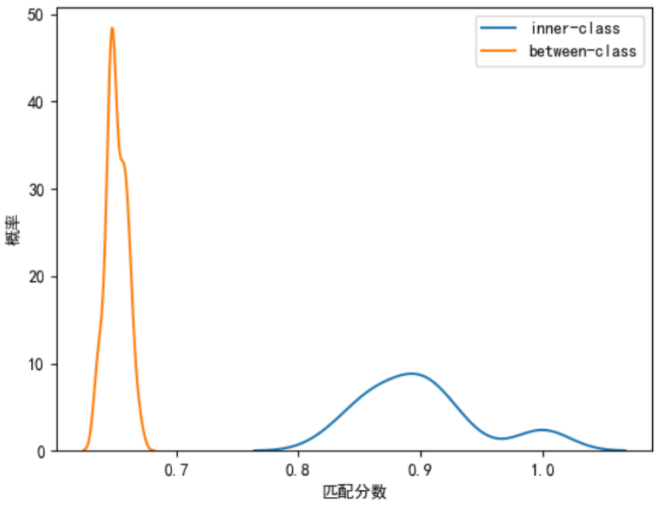
采用上述三种方法进行匹配，根据匹配结果绘制出类内样本和类间样本的距离分布图。

根据之前的得到的四组数据进行匹配得到结果：

手指：



手掌：

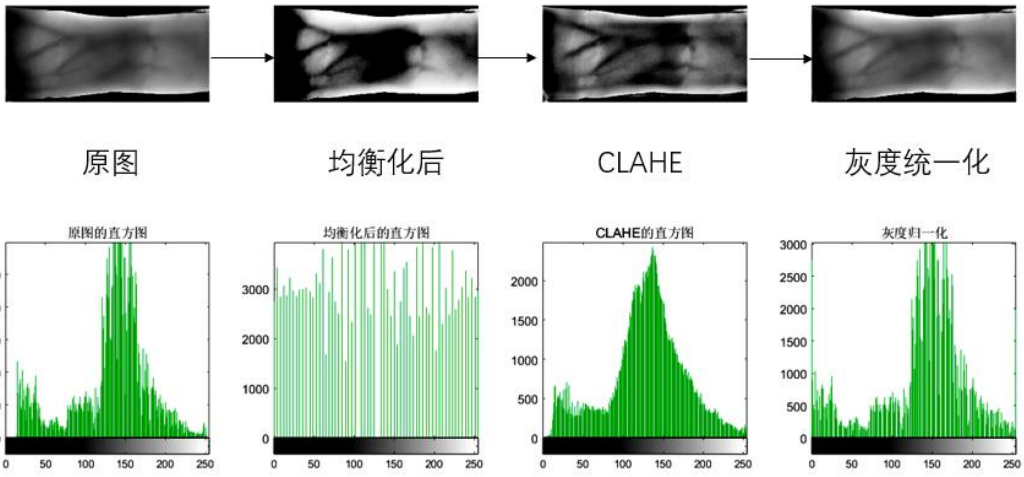


四、结果分析

对研究过程中所获得的主要的数据、现象进行定性或定量分析，得出结论和推论。

（一）图像增强：

各自采用全局直方图均衡化、限制对比度自适应直方图均衡化（CLAHE）、灰度归一化三种方法对图像进行增强，增强后的图像及其对应的直方图如图所示。



根据图显示的效果可以看出，在几种方法中，限制对比度自适应直方图均衡化（CLAHE）得出的指静脉图像较清晰，直方图分布跟原图相似，图像效果较好。

（二）特征提取：

对进行图像增强后的 ROI 图像进行二值化、像素取反、开操作等处理，得到提取的二值化纹理特征，其二值纹理特征提取过程如图所示。

可以看出，二值化纹理特征提取方法的效果良好。为校正手指的不同图像有细小偏移的问题，其中二值化的灰度阈值选取要适当，便于之后的指静脉图像互相匹配。

（三）模板匹配：

不同的提取特征类型往往有其对应的匹配方法，而合适的匹配方法能够极大提高识别精度。现有静脉图像匹配算法常采用某种距离去度量类内样本匹配分数和类间样本匹配分数。若两样本特征的距离小于某阈值，则认为是同一类，反之，则不是同一类。希望所使用匹配算法能使样本的同类样本间距离（类内距离）越小越好，类间样本间距离（或类间距离）越大越好，如此样本更易区分。在我们这次试验中，最终分析出的结果事实上并不理想，可能很大一部分原因是图像提取过程不够精确导致的。

五、创新实践总结

这次的课题比想象中要复杂难解的多，在实践过程中也遇到了各种各样的问题，

如对 python 语言的使用不够熟悉，在调用文件时路径出现问题，各种模块的调用不太清楚，数组的使用经常出现问题等等。

虽然困难，但仍旧在同学老师的帮助下简单初步的实现了题目所要求的功能，通过此次创新实践，能够更好的将理论应用到实践中去，清楚的认识到了手指、手掌静脉识别的过程和步骤，也更加了解各个步骤对应的方法，结合之前学习的数字图像处理，让我对数字图像处理的一些概念和原理有了直观的体会和更深刻的认识，比如直方图均衡化、开操作等，巩固了我对图像处理相关知识的理解，做到学以致用。

但，工程还存在很多需要改进的问题，如有些图片无法处理导致程序报错等等，还需要继续查找资料，提高能力，做进一步的改进。

参考文献

[1] Kumar, A., & Zhou, Y. (2012). Human Identification Using Finger Images. IEEE Transactions on Image Processing, 21, 2228-2244.

[2] Huang, B., Dai, Y., Li, R., Tang, D., & Li, W. (2010). Finger-Vein Authentication Based on Wide Line Detector and Pattern Normalization. 2010 20th International Conference on Pattern Recognition, 1269-1272.

[3] Miura, N., Nagasaka, A., & Miyatake, T. (2004). Feature extraction of finger-vein patterns based on repeated line tracking and its application to personal identification. Machine Vision and Applications, 15, 194-203.

[4] Miura, N., Nagasaka, A., & Miyatake, T. (2005). Extraction of Finger-Vein Patterns Using Maximum Curvature Points in Image Profiles. IEICE Trans. Inf. Syst., 90-D, 1185-1194.

[5] Choi, J., Song, W., Kim, T., Lee, S., & Kim, H. (2009). Finger vein extraction using gradient normalization and principal curvature. Electronic Imaging.

[6] Song, W., Kim, T., Kim, H., Choi, J.H., Kong, H., & Lee, S. (2011). A finger-vein verification system using mean curvature. Pattern Recognit. Lett., 32, 1541-1547.

[7]Rafael C.Gonzalez 著.数字图像处理（第三版）.北京:电子工业出版社, 2011.

[8]Rafael C.Gonzalez 主编.数字图像处理（MATLAB 版）.北京:电子工业出版社, 2005.

[9]Carsten Steger 等著.机器视觉算法与应用.北京:清华大学出版社, 2008.

附件：

1. 手指 ROI 区提取：

```
# 边缘检测画出边缘
def get_edges(img):  # img : single channel img which has been processed by cv.Canny
    rows, cols = img.shape
    center = rows / 2
    index = []
    up_index = []
    for i in range(cols):
        temp_index = np.argwhere(img[:, i] == 255)
        temp_index = temp_index.tolist()
        for j in temp_index:
            if j[0] < center:
                up_index.append(center - j[0])
        up = len(up_index) - 1
        up_index.clear()
        down = up + 1
        try:
            index.append([temp_index[up][0], i])
        except IndexError:
            index.append([index[-2][0], i])
        try:
            index.append([temp_index[down][0], i])
        except IndexError:
            index.append([index[-2][0], i])
    result_img = np.zeros((rows, cols), dtype="uint8")
    for j in index:
        result_img[j[0], j[1]] = 255  # put the Corresponding coordinate into 255
    return result_img


# 得到中点坐标
```

```
def get_center_line(image):
    rows, cols = image.shape
    code = []
    for i in range(cols):
        temp_index = np.argwhere(image[:, i] == 255)
        temp = temp_index.tolist()
        center_point = round((temp[0][0] + temp[1][0]) / 2)
        temp.clear()
        code.append([center_point, i])
    points = np.array(code)
    return points


# 将中线虚拟呈直线并在边缘图上画出
c_test = cv.imread('05.jpg', 0)
sobel_x = cv.Sobel(c_test, cv.CV_64F, 1, 0, ksize=5)
laplacian = cv.Laplacian(c_test, cv.CV_64F)
canny = cv.Canny(c_test, 50, 100)
finger_edges = get_edges(canny)
points = get_center_line(finger_edges)
line_vir = cv.fitLine(points, cv.DIST_L2, 0, 0.01, 0.01)
# print(line_vir)
line = cv.line(finger_edges, (int(line_vir[2] - 1000 * line_vir[1] / line_vir[0]),
int(line_vir[2] - 1000)),
               (int(line_vir[2] + 1000 * line_vir[1] / line_vir[0]),
int(line_vir[2] + 1000)), 255)

# 根据中线角度旋转图形
k = line_vir[1] / line_vir[0]
angle = math.atan(k)
im = Image.open("05.jpg")
# im.show()
im1 = im.rotate(angle)
im1.show()
```

```
# cv.imshow("line", line)
iml.save("tt.jpg")

# test_0 = cv.imread('tt.jpg', 0)
# gray_map = test_0[240]
# print(gray_map)
# plt.plot(gray_map)
# plt.show()
pic_test = cv.imread('tt.jpg', 0)
rows, cols = pic_test.shape
gray_map = pic_test[240]
sobel_x_pic = cv.Sobel(pic_test, cv.CV_64F, 1, 0, ksize=5)
laplacian_pic = cv.Laplacian(pic_test, cv.CV_64F)
canny_pic = cv.Canny(pic_test, 50, 100)
finger_edges_pic = get_edges(canny_pic)
points_pic = get_center_line(finger_edges_pic)
# print(points_pic)
line_vir_pic = cv.fitLine(points_pic, cv.DIST_L2, 0, 0.01, 0.01)


# cv.waitKey(0)
# cv.destroyAllWindows()


# 找到最亮的两个横坐标
def find_peak(arr):
    peaks = []
    step = 1
    pos = 40
    while pos < 400:  # 实际上两个 peak 主要分布在 100-200 和 450-550
        if (arr[pos] >= arr[pos + step]) and (arr[pos] > arr[pos - step]):
            if (arr[pos] > arr[pos + 20]) and (arr[pos] > arr[pos - 20]):  # 这
个 if 语句可以滤除很多的噪音 peak
                peaks.append(pos)
```

```
        pos = pos + step
    peak1 = 0
    peak2 = 300  # 取的图片中间靠后的位置，但是有可能存在后面的关节腔灰度值比中间位置低的情况
    # print(peaks)
    for i in peaks:  # 这个循环从好多个 peak 中找到我们要的两个最大的 peak
        if i < 200:
            if arr[i] >= arr[peak1]:
                peak1 = i
        else:
            if arr[i] >= arr[peak2]:
                peak2 = i
    return peak1, peak2


divide_point = find_peak(gray_map)
# print(divide_point)
points_pic_1 = line_vir_pic[3]
min_x = int(divide_point[0] - 30)
max_x = int(divide_point[1] + 20)
min_y = int(points_pic_1 - 60)
max_y = int(points_pic_1 + 60)
pic_image = pic_test[min_y:max_y, min_x:max_x]
cv.imshow("ROI", pic_image)
cv.waitKey(0)
cv.destroyAllWindows()
cv.imwrite("roi.jpg", pic_image)
```

2. 手掌 ROI 区提取：

```
import numpy as np
import cv2
from matplotlib import pyplot as plt
import scipy.signal as signal
import math
```

```
def getRoiImg(PalmveinImg):
    # **********说明**********
    # 这个函数是总的截取ROI的函数，输入原始手掌图片，返回手掌ROI图片，这个函数里面也包含了其他的辅助函数
    # *Input : the original PalmImage
    # *Output : the Palm ROI Image

    # ********伪代码**********
    Original_img = PalmveinImg.copy()
    PalmveinImg_gray = cv2.cvtColor(PalmveinImg, cv2.COLOR_BGR2GRAY)
    BackImg = PalmveinImg_gray.copy()
    # otsu method
    threshold, imgOtsu = cv2.threshold(BackImg, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    cv2.imwrite('otsu.jpg', imgOtsu)

    # for row in range(0, 100):
    #     for col in range(800, 900):
    #         imgOtsu[row][col] = 255

    # cv2.imshow('Maximum inscribed circle', imgOtsu)
    # cv2.waitKey(0)

    ################################## 计 算 手 掌 的 大 小 ##################################
    ############计算图片的大小############
    rows = len(imgOtsu)
    cols = len(imgOtsu[0])
    length = count_size(imgOtsu, rows, cols)

    ################################ 找 手 掌 下 面 ( 手 腕 ) 的 中 点 ############################
    Pref_row = 0
```

```
    Pref_col = int(cols / 2)
    for row in range(0, rows):
        if imgOtsu[rows - 1 - row][Pref_col] == 255:
            Pref_row = rows - 1 - row - int(length * 0.01)
            # Pref_row = rows - 1 - row
            break
    col_boundary = np.zeros(2)
    i = 0
    for col in range(1, cols):
        if imgOtsu[Pref_row][col] != imgOtsu[Pref_row][col - 1]:
            col_boundary[i] = col
            i = i + 1
            if (i == 2):
                break
    Pref_col = int((col_boundary[0] + col_boundary[1]) / 2)
    # print(Pref_x,Pref_y)

    ############################## 找 最 大 轮 廓 ， 再 根 据 计 算 极 值 点
##########################
    # Point* maxarea_point = new Point[10000];
    # totalnum = 0
    # 获取最大轮廓点集
    max_contours = getMaxRegion(imgOtsu, PalmveinImg)
    # print(len(max_contours))
    # print(len(max_contours[0]))

    # 获取极值点
    point9 = np.zeros((9, 2))
    point9 = find9point(Pref_row, max_contours)
    point9 = sortpoint(point9)
    # 打印 9 个点坐标
    # print(point9)

    #                                                cv2.circle(PalmveinImg,
```

```
(int(max_contours[10][0]),int(max_contours[10][1])), 5, (0,255,0))
    # cv2.circle(PalmveinImg, (Pref_row,Pref_col), 5, (0,0,255))
    # # 将点在原图中画出来并且显示
    for i in range(0, 9):
        cv2.circle(PalmveinImg, (int(point9[i][0]), int(point9[i][1])), 5, (255,
0, 0))
    cv2.imwrite('point9.jpg', PalmveinImg)

    # 判断左右手，右手：第一个点(0)和第三个点(2)的距离大于第七个点(6)和第九个点
(8)
    # 若为右手，连接第四个点(3)和第八个点(7)作为基准线
    # 若为左手，连接第二个点(1)和第六个点(5)作为基准线
    # （其实旋转角度不需要画出直线，计算斜率得到倾斜角就可以进行旋转矫正）
    base_point = np.zeros((2, 2))
    if (distance(point9[0], point9[2]) > distance(point9[6], point9[8])):
        base_point[0] = point9[3]
        base_point[1] = point9[7]
    else:
        base_point[0] = point9[1]
        base_point[1] = point9[5]
    # cv2.circle(PalmveinImg, (int(base_point[0][0]),int(base_point[0][1])), 5,
(255,0,0))
    # cv2.circle(PalmveinImg, (int(base_point[1][0]),int(base_point[1][1])), 5,
(255,0,0))
    # cv2.imshow('point9',PalmveinImg)
    # cv2.waitKey(0)
    k = (base_point[0][1] - base_point[1][1]) / (base_point[0][0] -
base_point[1][0])
    # print(k)
    angle = math.atan(k)
    # print(angle)

    ################################ 按 算 出 的 angle 旋 转 图 片
##############################
```

```
    # rotation_img = cv2.GetRotationMatrix2D((Pref_row, Pref_col), angle, 1.0, PalmveinImg)
    rot_mat  =  cv2.getRotationMatrix2D((base_point[0][0],  base_point[0][1]), angle * 180 / 3.14, 1.0)
    rotation_img = cv2.warpAffine(PalmveinImg, rot_mat, PalmveinImg.shape[1::-1], flags=cv2.INTER_LINEAR)

    # # 保存矫正图片
    # plt.subplot(1,2,1), plt.imshow(Original_img), plt.title('Original Image'), plt.xticks([]), plt.yticks([])
    # plt.subplot(1,2,2), plt.imshow(rotation_img), plt.title('rotation_img'), plt.xticks([]), plt.yticks([])
    # plt.savefig('rotation_2072.jpg')

    # cv2.imshow('rotation_img',rotation_img)
    # cv2.waitKey(0)

    ################################## 提 取 矩 形 ROI 区 域 ################################
    # Point* squarepoint = new Point[2];
    # cv.rectangle(img_rotation_rectangle, (left_line, up_line), (right_line, down_line), (255, 255, 255), 2)
    Square_length = distance(base_point[0], base_point[1])
    # img_rotation[up_line:down_line,left_line:right_line]
    # print(base_point)
    # print(Square_length)
    img_rotation_ROI = rotation_img[int(base_point[0][1] + 10):int(base_point[0][1] + Square_length + 10),
                       int(base_point[0][0] - Square_length):int(base_point[0][0])]

    return img_rotation_ROI
```

```
#################################################################################################################################

def getMaxRegion(imgOtsu, PalmveinImg):
    # /**********说明***********
    # 这个函数是根据二值化图片找出面积最大对应的轮廓点坐标
    # *Input : binary Image, the return Image, maxarea_point, the totalnum of the maxarea_point
    # *Output : None
    # *************************/

    # /********伪代码**********
    # 提取轮廓
    # cv2.imshow('Maximum inscribed circle', imgOtsu)
    # cv2.waitKey(0)
    contours, hierarchy = cv2.findContours(imgOtsu, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)

    max_area = 0
    for i in range(0, len(contours)):
        area = cv2.contourArea(contours[i])
        # print(area)
        if (area > max_area):
            max_area = area
    # print(max_area)

    # 标记轮廓
    cv2.drawContours(imgOtsu, contours, -1, (255, 0, 255), 3)

    size = imgOtsu.shape
    tempImg = np.zeros(imgOtsu.shape, np.uint8)

    i = 0
```

```
    for i in range(0, len(contours)):
        area = cv2.contourArea(contours[i])
        if (area == max_area):
            # print(contours[i])
            # totalnum = contours[i].size()
            # for j in range(0, totalnum-1):
            #     maxarea_point[j] = contours[i][j]
            # cv2.drawContours(PalmveinImg, contours[i], -1, 255, 1)
            # cv2.imwrite("contour.jpg",tempImg)
            break
    max_contours = np.zeros((len(contours[i]), 2))

    for j in range(len(contours[i])):
        max_contours[j][0] = int(contours[i][j][0][0])
        max_contours[j][1] = int(contours[i][j][0][1])
        # print(max_contours[j][0])
        # print(max_contours[j][1])
    return max_contours


#################################################################################################################################

###########找出手掌最高点和最低点，返回差值，后续按比例移动相应的距离#########
def count_size(img, rows, cols):
    up = -1
    down = -1
    for row in range(0, rows):
        for col in range(0, cols):
            # if(row == 00 and col > 300 and col < 800):
            #     print(img[row][col])
            if (img[row][col] == 255):
                up = row
                break
```

```
        if (up != -1):
            break
    for row in range(0, rows):
        for col in range(0, cols):
            if (img[rows - 1 - row][col] == 255):
                down = rows - 1 - row
                break
        if (down != -1):
            break
    # print(up, down)
    return down - up


#########################################################################################
###################################

############################找出手掌九个凹凸点#############################
# 传入参数：
#       level：手腕高度线
#       points：轮廓点集
# 输出：
#       result：9个凹凸点
def find9point(level, points):
    left = 1300
    left_index = 0
    right = 0
    right_index = 0
    # 找到手掌最左和最右的点及对应下标
    for i in range(len(points)):
        if (points[i][1] < level and points[i][0] < left):
            left = points[i][0]
            left_index = i
        if (points[i][1] < level and points[i][0] > right):
            right = points[i][0]
```

```
            right_index = i

    # print(left,right)
    # 定义结果集和结果索引
    result = np.zeros((9, 2))
    result[0] = points[0]
    result_index = 1
    # 轮廓是逆时针，第一个点是最高点，遍历的时候从最高点往左，到达最左，再进行一次从最右点到中间最高点的遍历
    # 记录当前趋势是上升还是下降，1 为上升，0 为下降
    # 思路 1：计算纵坐标差
    # 思路 2：计算索引差，要记录上一个索引值
    # 思路 3：结合思路 1 和 2
    raise_flag = 0
    last_index = 0
    dis = 30
    for i in range(0, left_index + 1):
        if (i > 10 and points[i][1] < (level - 100) and points[i][1] > points[i - 10][1]):
            if (raise_flag == 1):
                if (result_index == 0 or abs(i - last_index) > 100 and abs(
                        points[i][1] - result[result_index - 1][1]) > dis):
                    # if(result_index==0 or abs(i - last_index) >150):
                    last_index = i
                    result[result_index] = points[i - 10]
                    result_index += 1
            raise_flag = 0
        elif (i > 10 and points[i][1] < (level - 100) and points[i][1] < points[i - 10][1]):
            if (raise_flag == 0):
                # if(result_index==0 or abs(points[i][0] - result[result_index-1][0]) >50):
                if (result_index == 0 or abs(i - last_index) > 100 and abs(
                        points[i][1] - result[result_index - 1][1]) > dis):
```

```
                    # if(result_index==0 or abs(i - last_index) >150):
                    last_index = i
                    result[result_index] = points[i - 10]
                    result_index += 1
            raise_flag = 1
        if (result_index == 9):
            break

    raise_flag = 1
    for i in range(right_index, len(points)):
        if (i > 10 and points[i][1] < (level - 100) and points[i][1] > points[i
- 10][1]):
            if (raise_flag == 1):
                # if(result_index==0 or abs(points[i][0] - result[result_index-
1][0]) >50):
                if (result_index == 0 or abs(i - last_index) > 100 and abs(
                        points[i][1] - result[result_index - 1][1]) > dis):
                    # if(result_index==0 or abs(i - last_index) >150):
                    last_index = i
                    result[result_index] = points[i - 10]
                    result_index += 1
            raise_flag = 0
        elif (i > 10 and points[i][1] < (level - 100) and points[i][1] < points[i
- 10][1]):
            if (raise_flag == 0):
                # if(result_index==0 or abs(points[i][0] - result[result_index-
1][0]) >50):
                if (result_index == 0 or abs(i - last_index) > 100 and abs(
                        points[i][1] - result[result_index - 1][1]) > dis):
                    # if(result_index==0 or abs(i - last_index) >150):
                    last_index = i
                    result[result_index] = points[i - 10]
                    result_index += 1
            raise_flag = 1
```

```
        if (result_index == 9):
            break

    if (result_index < 9):
        print("错误：未找齐9个点!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! ")

    return result


#########################################################################################
####################################

#########################将9个点按从右到左的顺序排序#########################
# 传入参数：
#       points：9个凹凸点
# 输出：
#       result：从右到左排序的9个凹凸点
def sortpoint(points):
    result = np.zeros((9, 2))
    j = 0
    i = 1
    # 找到左右换边的点
    while i < 9:
        if (points[i][0] > points[i - 1][0]):
            break
        i += 1
    mid = i
    while i < 9:
        result[j] = points[i]
        i += 1
        j += 1
    for i in range(0, mid):
        result[j] = points[i]
        j += 1
```

```
    return result


#########################################################################################
####################################

############################计算两点之间的距离#############################
# 传入参数：
#       p1：点 1
#       p2：点 2
# 输出：
#       距离
def distance(p1, p2):
    return int(math.sqrt((p1[0] - p2[0]) * (p1[0] - p2[0]) + (p1[1] - p2[1]) *
(p1[1] - p2[1])))


#########################################################################################
####################################

###################################开始运行#################################
# imgFile = '208_1.bmp'
imgFile = '209_2.bmp'

# # load an original image
img = cv2.imread(imgFile)

# # 获取掌心 ROI 区域
roi = getRoiImg(img)
cv2.imwrite('roi.jpg', roi)
3.  图像增强：
img = cv.imread("roi.jpg", 0)
I_min, I_max = cv.minMaxLoc(img)[:2]
O_min, O_max = 0, 255
```

```
# 计算 a 和 b 的值
a = float(O_max - O_min) / (I_max - I_min)
b = O_min - a * I_min
out = a * img + b
out = out.astype(np.uint8)
cv.imshow("img", img)
cv.imshow("out", out)
cv.waitKey(0)
cv.destroyAllWindows()
cv.imwrite("roi_strong.jpg", out)
```

4. 图像分析：

```
import numpy as np
import cv2 as cv
import seaborn as sns
from scipy.spatial.distance import pdist, squareform
import math
import os
import sys
import matplotlib.pyplot as plt
import scipy.signal as signal
from skimage import feature


def load_image(path):
    img_gray = cv.imread(path, 0)
    return img_gray


def LBP_Extract(path):
    img_gray = load_image(path)

    # 使用 LBP 方法提取图像的纹理特征
    img = feature.local_binary_pattern(img_gray, 8, 1, 'default')
    # 转换数据类型便于后面处理
```

```
    img = img.astype('uint8')
    # cv.imshow("img",img)

    # 將編碼圖分块為4×2
    tile_cols = 4
    tile_rows = 2
    n = 1
    rows, cols = img.shape
    step_rows = round(rows / tile_rows) - 1
    step_cols = round(cols / tile_cols) - 1
    LBP_feature = []
    for i in range(tile_rows):
        for j in range(tile_cols):
            tile = img[i * step_rows:(i + 1) * step_rows - 1, j * step_cols:(j
+ 1) * step_cols - 1]
            # 统计直方图
            hist = cv.calcHist([tile], [0], None, [255], [0, 256])
            # 對直方圖幅值進行归一化
            hist = normalization(hist)
            # 將靜脈圖像的LBP直方圖特徵串接
            LBP_feature.append(hist)

            # plt.figure(n)
            # plt.subplot(4,2,n)
            # TODO:調整子圖間的距離
            # plt.subplots_adjust(left=0.04, top=0.96, right=0.96, bottom=0.04,
wspace=0.01, hspace=0.01)
            # plt.plot(hist)
            # plt.axis('off')
            # title_name = 'LBP'+str(n)
            # plt.title(title_name)
            # plt.show()
            # plt.clf()
            # 計算直方圖的數量
```

```
            n += 1
    # plt.show()

    LBP_feature = np.array(LBP_feature)  # 8*255
    LBP_feature = LBP_feature.ravel()  # 2040*1

    # 繪製串接后的LBP特徵圖
    # plt.figure(2)
    # plt.plot(LBP_feature)
    # plt.show()
    return LBP_feature


def LBP_match(path1, path2):
    hist_template = LBP_Extract(path1)
    hist_test = LBP_Extract(path2)
    template_sum = hist_template.sum()
    score = 0
    # score就是取兩直方圖對應點的最小值,計算所有點之和
    # 取最小值是因爲最小值相當於是兩直方圖重叠的部分
    for i in range(hist_template.size):
        if hist_template[i] <= hist_test[i]:
            score = score + hist_template[i]
        else:
            score = score + hist_test[i]
    score = score / template_sum
    return score


# 對直方圖的幅值進行歸一化處理
def normalization(array):
    result = []
    array = array.ravel()
    array_sum = array.sum()
```

```
    for i in array:
        result.append(i / array_sum)
    return np.array(result)


if __name__ == "__main__":

    dir_1 = 'roi_hand_1'
    dir_2 = 'roi_hand_2'
    imgList_1 = os.listdir(dir_1)
    imgList_2 = os.listdir(dir_2)
    # 读取图片路径
    im_path_1 = []
    im_path_2 = []
    for count_1 in range(0, len(imgList_1)):
        im_name_1 = imgList_1[count_1]
        im_path_1.append(os.path.join(dir_1, im_name_1))

    for count_2 in range(0, len(imgList_2)):
        im_name_2 = imgList_2[count_2]
        im_path_2.append(os.path.join(dir_2, im_name_2))
    # 储存20张图片的路径

    # 类内概率密度图
    score_1 = []
    score_2 = []
    i = 0
    j = 1
    while i < 9:
        while j < 10:
            score_1.append(LBP_match(im_path_1[i], im_path_1[j]))  # 類内
            score_2.append(LBP_match(im_path_1[i], im_path_2[j]))  # 類間
            j = j + 1
        j = i + 2
```

```
    score_2.append(LBP_match(im_path_1[i], im_path_2[i]))
    i = i + 1
# print(score_1)
# print(score_2)

sns.kdeplot(score_1, label="inner-class")  # 類内
sns.kdeplot(score_2, label="between-class")  # 類間
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
plt.xlabel('匹配分数')  # 设置 X 轴标签
plt.ylabel('概率')  # 设置 Y 轴标签
plt.legend()
plt.show()
cv.destroyAllWindows()
```