Московский авиационный институт (национальный исследовательский университет) (МАИ)

Группа М80-103М-19

Тема: Обработка изображений

Лабораторная работа №2

Подготовил: Ерещенко Алексей Владимирович

Постановка задачи

Разработать и реализовать программу для классификации изображений ладоней, обеспечивающую:

• Ввод и отображение на экране изображений в формате TIF;

• Сегментацию изображений на основе точечных и пространственных преобразований;

• Генерацию признаковых описаний формы ладоней на изображениях;

• Вычисление меры сходства ладоней;

• Кластеризацию изображений.

Описание данных

В качестве исходных данных прилагается набор из 99 цветных изображений ладоней разных людей, полученных с помощью сканера, в формате 489×684 с разрешением 72 dpi.

В данной работе используется только 7 изображений и их вариации.

Описание метода решения

В начале изображения были пере-масштабированы и приведены к размеру 700 на 700. Это было сделано в связи с тем, что в дальнейшем для классификации ладоней будет использоваться в том числе и их пиксельная площадь, и разная размерность изображений может негативно повлиять на точность классификации. Кроме того, размерность изображения влияет на работу различных операций, например морфологических. Также изображения конвертируются из Tif в Png, для упрощения работы с ними.

Затем изображения сделаны полутоновыми. Для создания алгоритма, который подходил бы для всевозможных изображений, была проведена бинаризация методом Оцу.

Алгоритм Оцу заключается в поиске порога, который бы максимизировал дисперсию между классами:

$$\sigma_b^2(t) = \sigma^2 - \sigma_\omega^2(t) = \omega_1(t)\omega_2(t)[\mu_1(t) - \mu_2(t)]^2$$

и включает следующие шаги: 1) вычисляется гистограмма изображения и его частота для каждого уровня интенсивности изображения

2) вычисляются начальные значения для $\omega_1(0), \omega_2(0), \mu_1(0), \mu_2(0)$

3) для каждого значения t от 1 до максимального значения в изображении:

• обновляются вероятности классов ω_1, ω_2 и их среднее арифметическое μ_1, μ_2

• вычисляется $\sigma_b^2(t) = \omega_1(t)\omega_2(t)[\mu_1(t) - \mu_1(2)]^2$

• если $\sigma_b^2(t)$ больше чем имеющееся, то это значение и порог t фиксируются

4) порог будет соответствовать максимуму $\sigma_b^2(t)$

На данном этапе, основной задачей было отделение очертания непосредственно ладони от кисти без серьезной потери формы. Бинаризация по Оцу не подошла, т.к. изображения не достаточно сильно сегментировались для дальнейшей работы. В связи с этим экспериментально был подобран порог бинаризации, равный порогу по Оцу с коэффициентом 1.2.

Далее, для дальнейшей сегментации, была применена морфологическая операция размыкания:

$A \circ B = (A \ominus B) \oplus B$

где A - изображение, B - массив (структурный элемент), определяющий, каких соседей пикселя использовать. Экспериментальным путем был подобран массив размера 4 на 4, и количество итераций операции равное 5.

Для дальнейшей обработки изображения, и сглаживания "острых" углов и закрытия дыр в изображении, был применен фильтр размытия по Гауссу. Экспериментальным путем было подобрано ядро (61,61), и сигма равная 0.

После данной обработки, контур ладони четко отделен от кисти руки. Были обрисованы контуры руки и части кисти, затем был взять наибольший контур, который в любом случае будет контуром руки.

На контуре ладони с пальцами были найдены 4 крайние точки - нижняя, верхняя, и также боковые. Эти точки были получены для вычисления длины от нижней точки ладони до наиболее длинного пальца (обычно это средний палец). Данная длина будет одним из признаков руки. Причина для фиксации также и боковых точек следующая - некоторые из изображений повернуты, и не всегда их верхние и нижние точки будут соответствовать длине от низа ладони до пальца. Т.к. мы заранее не можем знать, будет ли изображение наклонено, из этих двух расстояний берется наибольшее и фиксируется как длина руки.

В данной работе не используются такие потенциальные признаки как расстояние между пальцами, или от мизинца до большого пальца по следующей причине - как показывает набор входных данных, нет никаких ограничений на постановку пальцев руки. Один и тот же человек может их сдвинуть, или широко расставить, что может сильно повлиять на данные признаки и делает их малопригодными для классификации. Если бы всегда на всех изображениях руки были бы в фиксированном положении, эти признаки можно было бы использовать.

Также на данном этапе фиксируются пиксельная площадь ладони с пальцами и периметр контура, как будущие признаки.

Следующим этапом является получение изображения непосредственно ладони для попытки обрисовки рисунка линий ладони. Создается новое изображение на которое "вырезается" квадрат с контуром ладони с пальцами, затем с помощью морфологической операции размыкания с большим массивом (20 на 20, 4 итерации), достигается удаление с изображения пальцев. На данном этапе, фиксируется отношение площади ладони к площади ладони с пальцами как будущий признак, т.к. у разных людей это соотношение может быть разным.

Затем ладонь вырезается на новое изображение и приводится к масштабу 300 на 300, так чтобы на картинке была только ладонь без черного фона.

К данному изображению ладони применяется бинаризация по Оцу с коэффициентом порога 0.87 (был подобран экспериментально). Это делается с целью выделения линий на руке. Далее применяется операция замыкания:

$$A \cdot B = (A \oplus B) \ominus B$$

с массивом 4 на 4 на 2 итерациях (подобрано экспериментально), после чего применяется фильтр размытия по Гауссу (11 на 11, сигма 0). Проведение данных операций показывает, что несмотря на максимальное приближение непосредственно ладони, поймать точный рисунок такими методами не удается - складки кожи дают более сильный след, чем большинство линий на руке, в связи с чем бинаризация не может четко их зафиксировать. Тем не менее, определенные зоны удается выделить. Как признаки фиксируются следующие величины: количество найденных зон, их средняя площадь и длина контура, стандартное отклонение этих величин, а также сумма всех площадей и длин контуров.

Важно отметить, что признаки, основанные на рисунке ладони, требуют чтобы все исследуемые руки были только правые или только левые, т.к. рисунок правой руки одного человека отличается от рисунка левой руки того же самого человека. Так как никакой информации за пределами самих изображений рук не дано, дальнейшая работа ведется из предположения что все руки на изображениях были либо левые либо правые, либо все принадлежали разным людям, в противном случае использовать рисунок ладони для идентификации не имеет никакого смысла.

На основе всех выше описанных признаков для каждого изображения был сформирован вектор признаков. Для тестирования алгоритма, на основе выбранных 7 изображений были сформированы дополнительные 2 на каждое, являющиеся поворотами изначальных изображений. Это было сделано для тестирования поиска одного и того же человека и кластеризации изображений, т.к. по сути эти новые изображения просто несколько измененные оригиналы, руки изображены те же самые.

Для сравнения рук был разработан следующий алгоритм.

Для каждого соответствующего признака двух сравниваемых изображений получается следующее соотношение:

$$(\frac{a_{iV1}}{(a_{iV1} + a_{iV2})} - 0.5) * 100$$

где a_{iV1}, a_{iV2} i-тые признаки первой и второй картинки соответственно. По сути, данная величина измеряет степень близости значения признака одной картинки к тому же признаку другой картинки, и находится в поле значений $[0, 100]$, где 100 означает что величины бесконечно далеки друг от друга (недостижимо в реальной ситуации), а 0 - что величины признаков идентичны.

Схожесть рук определяется как среднее от сумм всех данных величин по каждому признаку.

В рамках оценки, каждая рука сравнивается с каждой. В отдельном массиве хранятся результаты сравнения, где прописан номер изображения для сравнения, номер изображения с которым оно сравнивается, метрика, а также отдельное поле для определения, принадлежат ли эти руки одному и тому же человеку (1 (да) или None (нет)).

В ходе тестовых экспериментов было выявлено, что часть признаков сильно варьируется при поворотах одного и того же изображения (количество внутренних линий, среднее и стандартное отклонение их площадей и длин контуров), и эти признаки были исключены из работы алгоритма. После этого, был определен порог сходства по выше указанной метрике в 98% для фиксирования принадлежности рук к одному и тому же человеку. Данный порог показал удовлетворительный результат на всех выбранных изображениях и их вариациях.

Далее был подготовлен код для вывода соответствующих условию задания таблиц на экран.

Описание программной реализации и метода решения

Для программной реализации проекта используется язык программирования Python версии 3.7

Импортирую необходимые библиотеки:

In [1]:

```
import numpy as np
from matplotlib import pyplot as plt
from PIL import  Image
from pylab import *
import cv2
from scipy.ndimage import measurements, morphology
import imutils
```

Выгружаю тестовое изображение, привожу его к масштабу 700 на 700 и оттенкам серого (для конвертации используется библиотека для обработки изображений PIL и OpenCV):

In [2]:

```
im = cv2.imread("003.tiff", 1)
cv2.imwrite("003.png", im);
```

In [3]:

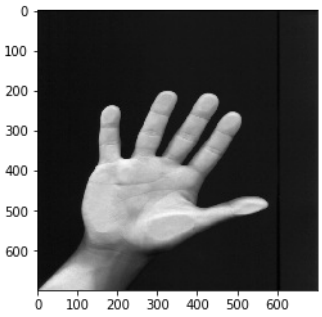
```
im = array(Image.open('003.png'))
width = 700
height = 700
dim = (width, height)
```

In [4]:

```
im = cv2.resize(im, dim, interpolation = cv2.INTER_AREA)
```

In [5]:

```
figure()
gray()
test_image = Image.fromarray(im).convert('L')
imshow(test_image)
show()
```



Бинаризую изображение методом Оцу с подобранным экспериментально коэффициентом порога в 1.2 (используется готовая реализация из библиотеки для обработки изображений OpenCV):

In [6]:

```
test_image = array(test_image)
ret, image = cv2.threshold(test_image,0,255,cv2.THRESH_OTSU)
```

In [7]:

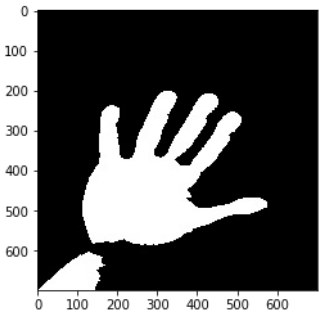
```
image = 1*(test_image>(ret*1.2))
```

In [8]:

```
imshow(Image.fromarray(uint8(image)))
```

Out[8]:

```
<matplotlib.image.AxesImage at 0x1724cbb9608>
```



Прменяю морфологическую операцию размыкания (используется готовая реализация библиотеки Scipy) для большего разделения кисти и ладони. Количество итераций и размер массива были подобраны экспериментальным путем:

In [9]:

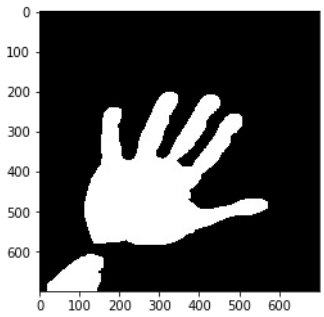
```
image_open = morphology.binary_opening(image, np.ones((4,4)), iterations = 5)
```

In [10]:

```
pil_image = Image.fromarray(uint8(image_open))
imshow(pil_image)
```

Out[10]:

```
<matplotlib.image.AxesImage at 0x1724cc2b608>
```



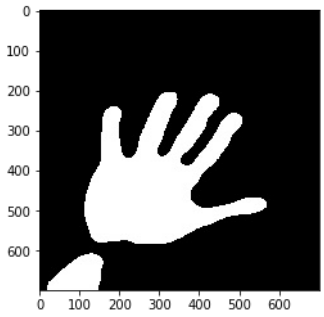
Применяю фильтр размытия Гаусса для "заделывания" неровностей (используется готовая реализация библиотеки OpenCV). Ядро было подобрано экспериментальным путем:

In [11]:

```
image_g = cv2.GaussianBlur(uint8(image_open), (61,61), 0)
```

In [12]:

```
figure()
imshow(Image.fromarray(uint8(image_g)))
show()
```



Нахожу контуры кисти и руки, сохраняю наибольший контур (это всегда будет рука), фиксирую пиксельную площадь и длину периметра руки, обрисовываю контур белым цветом:

In [13]:

```
image, contours, hierarchy = cv2.findContours(image_g,cv2.RETR_LIST,cv2.CHAIN_APPROX_SIMPLE)
```

In [14]:

```
cnt = max(contours, key=cv2.contourArea)
area = cv2.contourArea(cnt)
perimeter = cv2.arcLength(cnt,True)
```

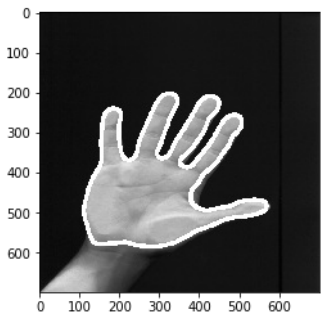
In [15]:

```
binary_image_contours = cv2.drawContours(test_image, [cnt], -1, (255,255,255), 10)
```

In [16]:

```
figure()
imshow(Image.fromarray(uint8(binary_image_contours)))
print("Number of objects:", len([cnt]))
show()
```

```
Number of objects: 1
```



Нахожу крайние точки контура руки, нахожу расстояние от нижней и верхней, а также расстояние между боковыми точками, беру наибольшее как длину руки (сделано для случаев, когда изображение может быть повернуто):

In [17]:

```
extLeft = tuple(cnt[cnt[:, :, 0].argmin()][0])
extRight = tuple(cnt[cnt[:, :, 0].argmax()][0])
extTop = tuple(cnt[cnt[:, :, 1].argmin()][0])
extBot = tuple(cnt[cnt[:, :, 1].argmax()][0])
```

In [18]:

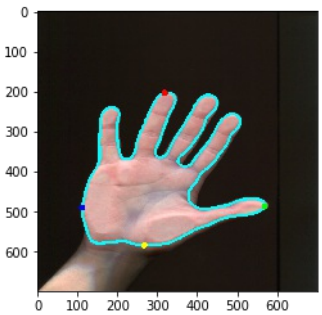
```
def calculateDistance(x1,x2):
    dist = math.sqrt((x2[0] - x1[0])**2 + (x2[1] - x1[1])**2)
    return dist
```

In [19]:

```
dist_width = calculateDistance(extLeft, extRight)
dist_height = calculateDistance(extBot, extTop)
dist = 0
if dist_height >= dist_width:
    dist = dist_height
else:
    dist = dist_width
```

In [20]:

```
figure()
image_dots = cv2.drawContours(im.copy(), [cnt], -1, (0, 255, 255), 5)
image_dots = cv2.circle(image_dots, extLeft, 8, (0, 0, 255), -1)
image_dots = cv2.circle(image_dots, extRight, 8, (0, 255, 0), -1)
image_dots = cv2.circle(image_dots, extTop, 8, (255, 0, 0), -1)
image_dots = cv2.circle(image_dots, extBot, 8, (255, 255, 0), -1)
imshow(Image.fromarray(uint8(image_dots)))
show()
```



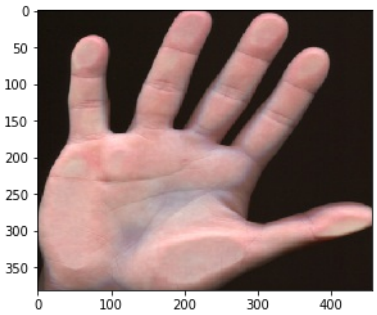
Вырезаю руку на новое изображение, привожу к серому:

In [21]:

```
cropped_img = Image.fromarray(im).crop((extLeft[0], extTop[1], extRight[0], extBot[1]))
```

In [22]:

```
figure()
imshow(cropped_img)
show()
```



In [23]:

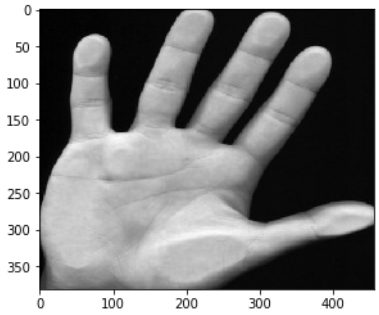
```
cropped_img_g = array(cropped_img.convert('L'))
```

In [24]:

```
imshow(Image.fromarray(uint8(cropped_img_g)))
```

Out[24]:

```
<matplotlib.image.AxesImage at 0x1724dd57408>
```



Бинаризую изображение по Оцу:

In [25]:

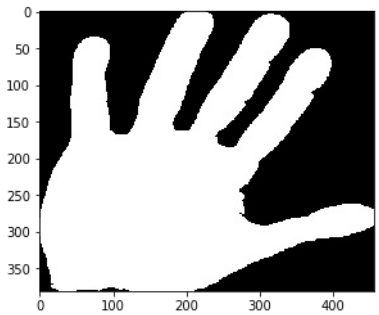
```
ret, image = cv2.threshold(cropped_img_g,0,255,cv2.THRESH_OTSU)
```

In [26]:

```
imshow(Image.fromarray(uint8(image)))
```

Out[26]:

```
<matplotlib.image.AxesImage at 0x1724ddbba08>
```



Применяю морфологическую операцию размыкания:

In [27]:

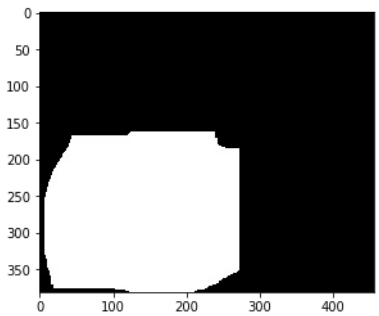
```
image_open = morphology.binary_opening(image, np.ones((20,20)), iterations = 4)
```

In [28]:

```
pil_image = Image.fromarray(uint8(image_open))
imshow(pil_image)
```

Out[28]:

```
<matplotlib.image.AxesImage at 0x1724dd15808>
```



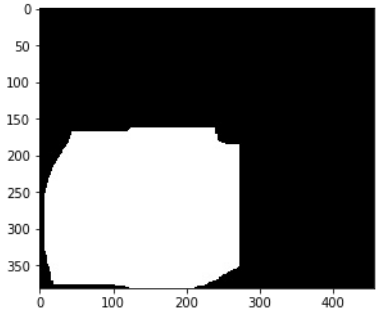
(здесь дополнительно используется фильтр по Гауссу для целей, связанных с реализацией кода, функциональной нагрузки у этой операции нет)

In [29]:

```
image_g = cv2.GaussianBlur(uint8(image_open), (1,1), 0)
```

In [30]:

```
figure()
imshow(Image.fromarray(uint8(image_g)))
show()
```



Нахожу контур, его площадь, а также отношение площади ладони к площади руки. Периметр не учитывается т.к. в данном случае этот признак не будет информативен, значительной вариации между разными руками не будет.

In [31]:

```
image, contours, hierarchy = cv2.findContours(image_g,cv2.RETR_LIST,cv2.CHAIN_APPROX_SIMPLE)
```

In [32]:

```
cnt = max(contours, key=cv2.contourArea)
palm_area = cv2.contourArea(cnt)
area_diff = palm_area/area
```

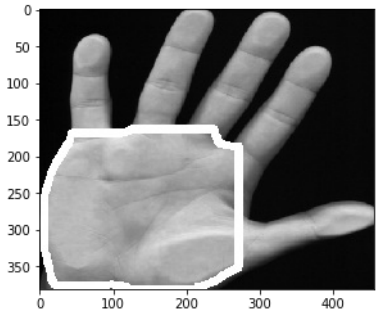
In [33]:

```
binary_image_contours = cv2.drawContours(cropped_img_g, [cnt], -1, (255,255,255), 10)
```

In [34]:

```
figure()
imshow(Image.fromarray(uint8(binary_image_contours)))
print("Number of objects:", len([cnt]))
show()
```

```
Number of objects: 1
```

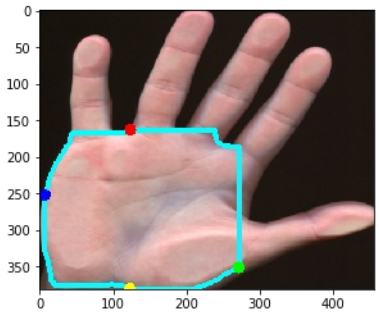


In [35]:

```
extLeft = tuple(cnt[cnt[:, :, 0].argmin()][0])
extRight = tuple(cnt[cnt[:, :, 0].argmax()][0])
extTop = tuple(cnt[cnt[:, :, 1].argmin()][0])
extBot = tuple(cnt[cnt[:, :, 1].argmax()][0])
```

In [36]:

```
figure()
image_dots = cv2.drawContours(array(cropped_img), [cnt], -1, (0, 255, 255), 5)
image_dots = cv2.circle(image_dots, extLeft, 8, (0, 0, 255), -1)
image_dots = cv2.circle(image_dots, extRight, 8, (0, 255, 0), -1)
image_dots = cv2.circle(image_dots, extTop, 8, (255, 0, 0), -1)
image_dots = cv2.circle(image_dots, extBot, 8, (255, 255, 0), -1)
imshow(Image.fromarray(uint8(image_dots)))
show()
```



По координатам крайних точек вырезаю ладонь на новую картинку, привожу к серому:

In [37]:

```
cropped_img_2 = cropped_img.crop((extLeft[0], extTop[1], extRight[0], extBot[1]))
```

In [38]:

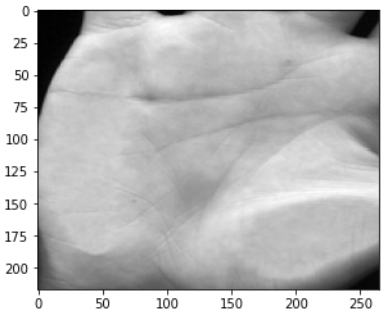
```
cropped_img_2 = array(cropped_img_2.convert('L'))
```

In [39]:

```
imshow(Image.fromarray(uint8(cropped_img_2)))
```

Out[39]:

```
<matplotlib.image.AxesImage at 0x1724c9c28c8>
```



Обрезаю изображение так, чтобы не осталось фона, который бы повлиял на бинаризацию и поиск рисунка ладони, затем привожу к масштабу 300 на 300:

In [40]:

```
height, width = cropped_img_2.shape[:2]
```

In [41]:

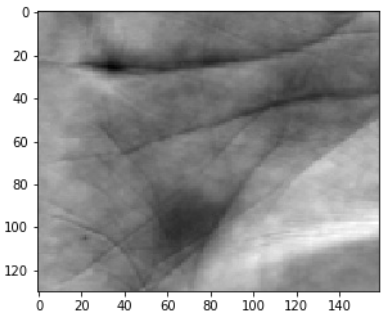
```
crop_img = cropped_img_2[int(height*0.2):int(height*0.8),int(width*0.2):int(width*0.8)]
```

In [42]:

```
imshow(Image.fromarray(uint8(crop_img)))
```

Out[42]:

```
<matplotlib.image.AxesImage at 0x1724de206c8>
```



In [43]:

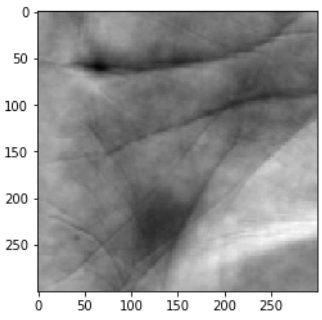
```
width = 300
height = 300
dim = (width, height)
resized = cv2.resize(crop_img, dim, interpolation = cv2.INTER_AREA)
```

In [44]:

```
imshow(Image.fromarray(uint8(resized)))
```

Out[44]:

```
<matplotlib.image.AxesImage at 0x1724ca81648>
```



Провожу бинаризацию по Оцу с коэффициентом порога 0.87:

In [45]:

```
ret, image = cv2.threshold(resized, 0,255,cv2.THRESH_OTSU)
```

In [46]:

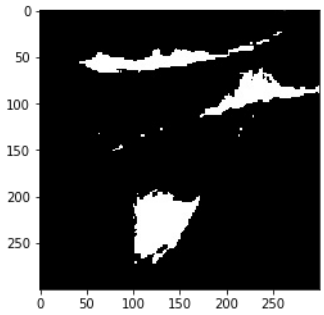
```
image = 1*(resized<(ret*0.87))
```

In [47]:

```
imshow(Image.fromarray(uint8(image)))
```

Out[47]:

```
<matplotlib.image.AxesImage at 0x1724caeefc8>
```



Провожу операцию замыкания для заделивания "дыр" после бинаризации:

In [48]:

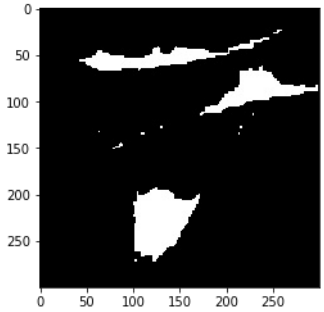
```
image = morphology.binary_closing(image, np.ones((4,4)), iterations = 2)
```

In [49]:

```
pil_image = Image.fromarray(uint8(image))
imshow(pil_image)
```

Out[49]:

```
<matplotlib.image.AxesImage at 0x1724cb54948>
```



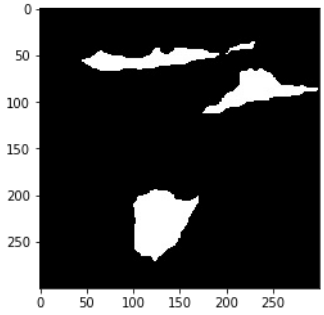
Применяю фильтр по Гауссу для удаления слишком маленьких элементов и сглаживания контуров:

In [50]:

```
image_ga = cv2.GaussianBlur(uint8(image), (11,11), 0)
```

In [51]:

```
figure()
imshow(Image.fromarray(uint8(image_ga)))
show()
```



Нахожу и обрисовываю контура:

In [52]:

```
image, contours, hierarchy = cv2.findContours(image_ga,cv2.RETR_LIST,cv2.CHAIN_APPROX_SIMPLE)
```

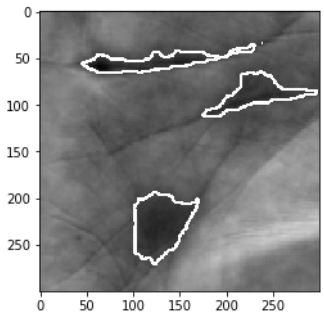
In [53]:

```
binary_image_contours = cv2.drawContours(array(resized), contours, -1, (255,255,255), 2)
```

In [54]:

```
figure()
imshow(Image.fromarray(uint8(binary_image_contours)))
print("Number of objects:", len(contours))
show()
```

```
Number of objects: 4
```



Как можно видеть на изображении выше, складки кожи дают куда больший след на изображении, чем отдельные линии. Тем не менее, некоторое признаковое описание, уникальное для конкретной руки, получить удается. Получаю из найденных контуров их площади, периметры, количество, суммы их площадей и периметров, а также среднее и стандартное отклонение этих величин:

In [55]:

```
def contour_feature_taker(contour):
    num = len(contour)
    contour_areas = []
    contour_perimeters = []

    for c in contour:
        contour_areas.append(cv2.contourArea(c))
    for c in contour:
        contour_perimeters.append(cv2.arcLength(c, True))

    stdc = std(contour_areas)
    meanc = mean(contour_areas)
    sm = sum(contour_areas)
    num_cnt = len(contour_areas)

    p_std = std(contour_perimeters)
    p_mean = mean(contour_perimeters)
    p_sm = sum(contour_perimeters)

    return stdc,meanc,sm,num_cnt,p_std,p_mean,p_sm
```

In [56]:

```
stdc,meanc,sm,num_cnt,p_std,p_mean,p_sm = contour_feature_taker(contours)
stdc,meanc,sm,num_cnt,p_std,p_mean,p_sm
```

Out[56]:

```
(1193.1012373222986,
 1823.75,
 7295.0,
 4,
 152.51429477425398,
 243.00966650247574,
 972.038666009903)
```

In [57]:

```
feat_vect = np.array([area, perimeter, dist, area_diff, stdc, meanc, sm, p_std, p_mean, p_sm])
feat_vect
```

Out[57]:

```
array([8.96665000e+04, 2.61220640e+03, 4.57009847e+02, 5.93281772e-01,
       1.19310124e+03, 1.82375000e+03, 7.29500000e+03, 1.52514295e+02,
       2.43009667e+02, 9.72038666e+02])
```

Задаю итоговую функцию, которая проводит все вышеописанные операции и получает вектор признаков из изображения:

In [58]:

```
def calculateDistance(x1,x2):
        dist = math.sqrt((x2[0] - x1[0])**2 + (x2[1] - x1[1])**2)
        return dist
```

In [59]:

```
def get_features(img):

    width = 700
    height = 700
    dim = (width, height)

    im = cv2.resize(img, dim, interpolation = cv2.INTER_AREA)

    test_image = Image.fromarray(im).convert('L')

    ret, image = cv2.threshold(array(test_image),0,255,cv2.THRESH_OTSU)
    image_1 = 1*(array(test_image)>(ret*1.2))

    image_2 = morphology.binary_opening(image_1, np.ones((4,4)), iterations = 5)
    image_3 = cv2.GaussianBlur(uint8(image_2), (61,61), 0)

    image_d, contours_f, hierarchy = cv2.findContours(image_3,cv2.RETR_LIST,cv2.CHAIN_APPROX_SIMPLE)

    cnt = max(contours_f, key=cv2.contourArea)
    area = cv2.contourArea(cnt)
    perimeter = cv2.arcLength(cnt,True)

    extLeft = tuple(cnt[cnt[:, :, 0].argmin()][0])
    extRight = tuple(cnt[cnt[:, :, 0].argmax()][0])
    extTop = tuple(cnt[cnt[:, :, 1].argmin()][0])
    extBot = tuple(cnt[cnt[:, :, 1].argmax()][0])

    dist_width = calculateDistance(extLeft, extRight)
    dist_height = calculateDistance(extBot, extTop)
    dist = 0
    if dist_height >= dist_width:
        dist = dist_height
    else:
        dist = dist_width

    cropped_img = Image.fromarray(im).crop((extLeft[0], extTop[1], extRight[0], extBot[1]))
    cropped_img_g = array(cropped_img.convert('L'))
    ret, image_4 = cv2.threshold(cropped_img_g,0,255,cv2.THRESH_OTSU)
    image_5 = morphology.binary_opening(image_4, np.ones((20,20)), iterations = 4)
    image_6 = cv2.GaussianBlur(uint8(image_5), (1,1), 0)

    image, contours, hierarchy = cv2.findContours(image_6,cv2.RETR_LIST,cv2.CHAIN_APPROX_SIMPLE)
    cnt = max(contours, key=cv2.contourArea)
    palm_area = cv2.contourArea(cnt)
    area_diff = palm_area/area

    extLeft_2 = tuple(cnt[cnt[:, :, 0].argmin()][0])
    extRight_2 = tuple(cnt[cnt[:, :, 0].argmax()][0])
    extTop_2 = tuple(cnt[cnt[:, :, 1].argmin()][0])
    extBot_2 = tuple(cnt[cnt[:, :, 1].argmax()][0])

    cropped_img_2 = cropped_img.crop((extLeft_2[0], extTop_2[1], extRight_2[0], extBot_2[1]))
    cropped_img_2 = array(cropped_img_2.convert('L'))
    height_2, width_2 = cropped_img_2.shape[:2]
    crop_img = cropped_img_2[int(height_2*0.2):int(height_2*0.8),int(width_2*0.2):int(width_2*0.8)]

    width_2 = 300
    height_2 = 300
    dim_2 = (width_2, height_2)
    resized = cv2.resize(crop_img, dim_2, interpolation = cv2.INTER_AREA)

    ret, image = cv2.threshold(resized, 0,255,cv2.THRESH_OTSU)
    image_7 = 1*(resized<(ret*0.87))

    image_8 = morphology.binary_closing(image_7, np.ones((4,4)), iterations = 2)
    image_9 = cv2.GaussianBlur(uint8(image_8), (11,11), 0)

    image, contours_2, hierarchy = cv2.findContours(image_9,cv2.RETR_LIST,cv2.CHAIN_APPROX_SIMPLE)

    stdc,meanc,sm, num, p_std,p_mean,p_sm = contour_feature_taker(contours_2)
    feat_vect = np.array([area, perimeter, dist, area_diff, stdc, meanc, sm, num, p_std, p_mean, p_sm])

    return feat_vect
```

Тестирую алгоритм на отдельно выбранном изображении и двух его разворотах:

In [60]:

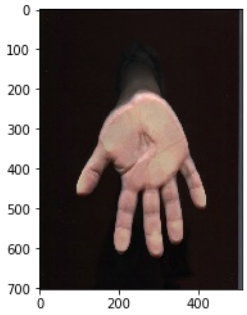
```
test = get_features(array(Image.open('145.png')))
```

In [61]:

```
imshow(array(Image.open('145.png')))
```

Out[61]:

```
<matplotlib.image.AxesImage at 0x1724cb1f048>
```



In [62]:

```
test
```

Out[62]:

```
array([8.55500000e+04, 2.40508656e+03, 4.44986517e+02, 5.92501461e-01,
       1.63400000e+03, 3.14350000e+03, 6.28700000e+03, 2.00000000e+00,
       9.92964627e+01, 3.75362478e+02, 7.50724957e+02])
```

In [63]:

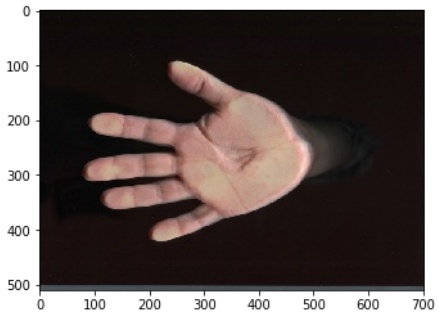
```
test = get_features(array(Image.open('145_1.png')))
```

In [64]:

```
imshow(array(Image.open('145_1.png')))
```

Out[64]:

```
<matplotlib.image.AxesImage at 0x1724c095bc8>
```



In [65]:

```
test
```

Out[65]:

```
array([8.55550000e+04, 2.40508656e+03, 4.44154252e+02, 5.92466834e-01,
       1.48875000e+03, 3.10975000e+03, 6.21950000e+03, 2.00000000e+00,
       9.48319964e+01, 3.73483799e+02, 7.46967597e+02])
```

In [66]:

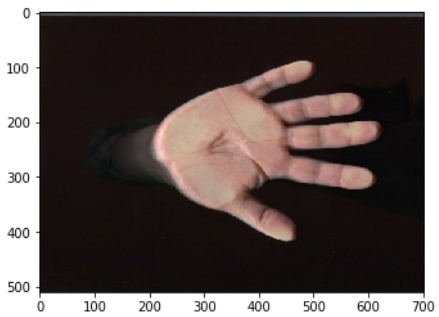
```
test = get_features(array(Image.open('145_2.png')))
```

In [67]:

```
imshow(array(Image.open('145_2.png')))
```

Out[67]:

```
<matplotlib.image.AxesImage at 0x1724cc69348>
```



In [68]:

```
test
```

Out[68]:

```
array([8.55550000e+04, 2.40508656e+03, 4.43948195e+02, 5.92466834e-01,
       1.66650000e+03, 3.12100000e+03, 6.24200000e+03, 2.00000000e+00,
       1.02296463e+02, 3.70705625e+02, 7.41411250e+02])
```

Как можно видеть по значениям векторов признаков, признаки, связанные с количеством, а также средним и стандартным отклонением контуров линий ладони слишком сильно варьируются при повороте одного и того же изображения. В связи с этим, эти признаки были отброшены. Формирую итоговые функции с учетом поправок:

In [69]:

```
def contour_feature_taker(contour):
    num = len(contour)
    contour_areas = []
    contour_perimeters = []

    for c in contour:
        contour_areas.append(cv2.contourArea(c))
    for c in contour:
        contour_perimeters.append(cv2.arcLength(c, True))


    sm = sum(contour_areas)
    p_sm = sum(contour_perimeters)

    return sm,p_sm
```

In [70]:

```
def get_features(img):

    width = 700
    height = 700
    dim = (width, height)

    im = cv2.resize(img, dim, interpolation = cv2.INTER_AREA)

    test_image = Image.fromarray(im).convert('L')

    ret, image = cv2.threshold(array(test_image),0,255,cv2.THRESH_OTSU)
    image_1 = 1*(array(test_image)>(ret*1.2))

    image_2 = morphology.binary_opening(image_1, np.ones((4,4)), iterations = 5)
    image_3 = cv2.GaussianBlur(uint8(image_2), (61,61), 0)

    image_d, contours_f, hierarchy = cv2.findContours(image_3,cv2.RETR_LIST,cv2.CHAIN_APPROX_SIMPLE)

    cnt = max(contours_f, key=cv2.contourArea)
    area = cv2.contourArea(cnt)
    perimeter = cv2.arcLength(cnt,True)

    extLeft = tuple(cnt[cnt[:, :, 0].argmin()][0])
    extRight = tuple(cnt[cnt[:, :, 0].argmax()][0])
    extTop = tuple(cnt[cnt[:, :, 1].argmin()][0])
    extBot = tuple(cnt[cnt[:, :, 1].argmax()][0])

    dist_width = calculateDistance(extLeft, extRight)
    dist_height = calculateDistance(extBot, extTop)
    dist = 0
    if dist_height >= dist_width:
        dist = dist_height
    else:
        dist = dist_width

    cropped_img = Image.fromarray(im).crop((extLeft[0], extTop[1], extRight[0], extBot[1]))
    cropped_img_g = array(cropped_img.convert('L'))
    ret, image_4 = cv2.threshold(cropped_img_g,0,255,cv2.THRESH_OTSU)
    image_5 = morphology.binary_opening(image_4, np.ones((20,20)), iterations = 4)
    image_6 = cv2.GaussianBlur(uint8(image_5), (1,1), 0)

    image, contours, hierarchy = cv2.findContours(image_6,cv2.RETR_LIST,cv2.CHAIN_APPROX_SIMPLE)
    cnt = max(contours, key=cv2.contourArea)
    palm_area = cv2.contourArea(cnt)
    area_diff = palm_area/area

    extLeft_2 = tuple(cnt[cnt[:, :, 0].argmin()][0])
    extRight_2 = tuple(cnt[cnt[:, :, 0].argmax()][0])
    extTop_2 = tuple(cnt[cnt[:, :, 1].argmin()][0])
    extBot_2 = tuple(cnt[cnt[:, :, 1].argmax()][0])

    cropped_img_2 = cropped_img.crop((extLeft_2[0], extTop_2[1], extRight_2[0], extBot_2[1]))
    cropped_img_2 = array(cropped_img_2.convert('L'))
    height_2, width_2 = cropped_img_2.shape[:2]
    crop_img = cropped_img_2[int(height_2*0.2):int(height_2*0.8),int(width_2*0.2):int(width_2*0.8)]

    width_2 = 300
    height_2 = 300
    dim_2 = (width_2, height_2)
    resized = cv2.resize(crop_img, dim_2, interpolation = cv2.INTER_AREA)

    ret, image = cv2.threshold(resized, 0,255,cv2.THRESH_OTSU)
    image_7 = 1*(resized<(ret*0.87))

    image_8 = morphology.binary_closing(image_7, np.ones((4,4)), iterations = 2)
    image_9 = cv2.GaussianBlur(uint8(image_8), (11,11), 0)

    image, contours_2, hierarchy = cv2.findContours(image_9,cv2.RETR_LIST,cv2.CHAIN_APPROX_SIMPLE)

    sm, p_sm = contour_feature_taker(contours_2)
    feat_vect = np.array([area, perimeter, dist, area_diff, sm, p_sm])

    return feat_vect
```

Провожу тест на наборе из нескольких изображений:

In [71]:

```
test_1 = get_features(array(Image.open('003.png')))
```

In [72]:

```
test_1
```

Out[72]:

```
array([8.96665000e+04, 2.61220640e+03, 4.57009847e+02, 5.93281772e-01,
       7.29500000e+03, 9.72038666e+02])
```

In [73]:

```
im = cv2.imread("003_1.tif", 1)
cv2.imwrite("003_1.png", im);
```

In [74]:

```
test_2 = get_features(array(Image.open('003_1.png')))
```

In [75]:

```
test_2
```

Out[75]:

```
array([8.96660000e+04, 2.61279219e+03, 4.57184864e+02, 5.94272076e-01,
       7.16600000e+03, 9.69352373e+02])
```

In [76]:

```
im = cv2.imread("003_2.tif", 1)
cv2.imwrite("003_2.png", im);
test_3 = get_features(array(Image.open('003_2.png')))
```

In [77]:

```
test_3
```

Out[77]:

```
array([8.96660000e+04, 2.61279219e+03, 4.57088613e+02, 5.93379876e-01,
       7.23950000e+03, 9.71109733e+02])
```

In [78]:

```
im = cv2.imread("145.tif", 1)
cv2.imwrite("145.png", im);
test_4 = get_features(array(Image.open('145.png')))
```

In [79]:

```
test_4
```

Out[79]:

```
array([8.55500000e+04, 2.40508656e+03, 4.44986517e+02, 5.92501461e-01,
       6.28700000e+03, 7.50724957e+02])
```

In [80]:

```
im = cv2.imread("145_1.tif", 1)
cv2.imwrite("145_1.png", im);
test_5 = get_features(array(Image.open('145_1.png')))
```

In [81]:

```
test_5
```

Out[81]:

```
array([8.55550000e+04, 2.40508656e+03, 4.44154252e+02, 5.92466834e-01,
       6.21950000e+03, 7.46967597e+02])
```

Как можно видеть, значения скорректированных признаков одинаковых рук намного ближе, что и должно быть (названия изображений с _ и числом обозначают развороты одного и того же изображения). Теперь пишу функцию сравнения признаков и тестирую ее:

In [82]:

```
ft_lst = [test_1,test_2,test_3,test_4,test_5]
```

In [83]:

```
def feature_comp(lst):
    features = lst
    lst_metrics = []
    for i in range(0,len(features)):
        score_1 = 0
        score_2 = 0
        score_3 = 0
        score_3 = 0
        score_4 = 0
        score_5 = 0
        score_6 = 0
        same_person = None
        for j in range(0,len(features)):
            if i == j:
                continue

            score_1 = round(abs((features[i][0]/(features[i][0]+features[j][0]))-0.5),5)

            score_2 = round(abs((features[i][1]/(features[i][1]+features[j][1]))-0.5),5)

            score_3 = round(abs((features[i][2]/(features[i][2]+features[j][2]))-0.5),5)

            score_4 = round(abs((features[i][3]/(features[i][3]+features[j][3]))-0.5),5)

            score_5 = round(abs((features[i][4]/(features[i][4]+features[j][4]))-0.5),5)

            score_6 = round(abs((features[i][5]/(features[i][5]+features[j][5]))-0.5),5)
            metric = round((((score_1 + score_2 + score_3 + score_4 + score_5 + score_6)/6)*100),3)
            if metric <= 2:
                same_person = 1
            else:
                same_person = None
            lst_metrics.append([i,j,metric,same_person])
    return lst_metrics
```

In [84]:

```
m_test = feature_comp(ft_lst)
```

In [85]:

```
m_test
```

Out[85]:

```
[[0, 1, 0.096, 1],
 [0, 2, 0.038, 1],
 [0, 3, 2.345, None],
 [0, 4, 2.418, None],
 [1, 0, 0.096, 1],
 [1, 2, 0.057, 1],
 [1, 3, 2.27, None],
 [1, 4, 2.343, None],
 [2, 0, 0.038, 1],
 [2, 1, 0.057, 1],
 [2, 3, 2.312, None],
 [2, 4, 2.385, None],
 [3, 0, 2.345, None],
 [3, 1, 2.27, None],
 [3, 2, 2.312, None],
 [3, 4, 0.074, 1],
 [4, 0, 2.418, None],
 [4, 1, 2.343, None],
 [4, 2, 2.385, None],
 [4, 3, 0.074, 1]]
```

Как можно видеть, метрика справилась с идентификацией одинаковых рук.

Теперь провожу тест на 21 изображении (все выбранные для работы изображения + их повороты):

In [106]:

```
array_of_images = []
array_of_paths = []
array_of_filenames = []
```

In [107]:

```
import os
dir_path = 'C:/Users/Alexey/Desktop/MAI/2semester/Imaging_de/var_2/'
for element in os.listdir(dir_path):
    im_s = cv2.imread(element, 1)
    filename, file_type = os.path.splitext(element)
    fln = filename+".png"
    path = os.path.join(dir_path,fln)
    if cv2.imread(path) is None:
        cv2.imwrite(path, im_s)
        array_of_paths.append(path)
        array_of_filenames.append(fln)
        array_of_images.append(array(Image.open(path)))
```

In [88]:

```
all_features = []
for i in array_of_images:
    featur = get_features(i)
    all_features.append(featur)
```

In [89]:

```
all_scores = feature_comp(all_features)
all_scores
```

Out[89]:

```
[[0, 1, 0.096, 1],
 [0, 2, 0.038, 1],
 [0, 3, 7.22, None],
 [0, 4, 6.938, None],
 [0, 5, 7.282, None],
 [0, 6, 4.465, None],
 [0, 7, 4.354, None],
 [0, 8, 4.469, None],
 [0, 9, 10.557, None],
 [0, 10, 10.72, None],
 [0, 11, 10.543, None],
 [0, 12, 2.776, None],
 [0, 13, 2.856, None],
 [0, 14, 2.816, None],
 [0, 15, 2.345, None],
 [0, 16, 2.418, None],
 [0, 17, 2.436, None],
 [0, 18, 10.171, None],
 [0, 19, 11.405, None],
 [0, 20, 10.087, None],
 [1, 0, 0.096, 1],
 [1, 2, 0.057, 1],
 [1, 3, 7.17, None],
 [1, 4, 6.886, None],
 [1, 5, 7.232, None],
 [1, 6, 4.392, None],
 [1, 7, 4.28, None],
 [1, 8, 4.396, None],
 [1, 9, 10.506, None],
 [1, 10, 10.669, None],
 [1, 11, 10.491, None],
 [1, 12, 2.7, None],
 [1, 13, 2.78, None],
 [1, 14, 2.74, None],
 [1, 15, 2.27, None],
 [1, 16, 2.343, None],
 [1, 17, 2.36, None],
 [1, 18, 10.1, None],
 [1, 19, 11.34, None],
 [1, 20, 10.016, None],
 [2, 0, 0.038, 1],
 [2, 1, 0.057, 1],
 [2, 3, 7.198, None],
 [2, 4, 6.914, None],
 [2, 5, 7.26, None],
 [2, 6, 4.433, None],
 [2, 7, 4.321, None],
 [2, 8, 4.437, None],
 [2, 9, 10.532, None],
 [2, 10, 10.696, None],
 [2, 11, 10.518, None],
 [2, 12, 2.742, None],
 [2, 13, 2.822, None],
 [2, 14, 2.783, None],
 [2, 15, 2.312, None],
 [2, 16, 2.385, None],
 [2, 17, 2.403, None],
 [2, 18, 10.144, None],
 [2, 19, 11.381, None],
 [2, 20, 10.06, None],
 [3, 0, 7.22, None],
 [3, 1, 7.17, None],
 [3, 2, 7.198, None],
 [3, 4, 0.377, 1],
 [3, 5, 0.07, 1],
 [3, 6, 5.809, None],
 [3, 7, 5.888, None],
 [3, 8, 5.78, None],
 [3, 9, 4.34, None],
 [3, 10, 4.508, None],
 [3, 11, 4.302, None],
 [3, 12, 6.282, None],
 [3, 13, 6.315, None],
 [3, 14, 6.298, None],
 [3, 15, 5.971, None],
 [3, 16, 5.91, None],
 [3, 17, 5.888, None],
 [3, 18, 4.288, None],
 [3, 19, 4.805, None],
 [3, 20, 4.233, None],
 [4, 0, 6.938, None],
 [4, 1, 6.886, None],
 [4, 2, 6.914, None],
 [4, 3, 0.377, 1],
 [4, 5, 0.442, 1],
 [4, 6, 5.472, None],
```

[4, 7, 5.551, None],
[4, 8, 5.442, None],
[4, 9, 4.696, None],
[4, 10, 4.863, None],
[4, 11, 4.658, None],
[4, 12, 5.958, None],
[4, 13, 5.99, None],
[4, 14, 5.974, None],
[4, 15, 5.67, None],
[4, 16, 5.608, None],
[4, 17, 5.586, None],
[4, 18, 4.195, None],
[4, 19, 5.176, None],
[4, 20, 4.14, None],
[5, 0, 7.282, None],
[5, 1, 7.232, None],
[5, 2, 7.26, None],
[5, 3, 0.07, 1],
[5, 4, 0.442, 1],
[5, 6, 5.874, None],
[5, 7, 5.953, None],
[5, 8, 5.845, None],
[5, 9, 4.281, None],
[5, 10, 4.45, None],
[5, 11, 4.244, None],
[5, 12, 6.344, None],
[5, 13, 6.378, None],
[5, 14, 6.361, None],
[5, 15, 6.037, None],
[5, 16, 5.976, None],
[5, 17, 5.954, None],
[5, 18, 4.247, None],
[5, 19, 4.742, None],
[5, 20, 4.192, None],
[6, 0, 4.465, None],
[6, 1, 4.392, None],
[6, 2, 4.433, None],
[6, 3, 5.809, None],
[6, 4, 5.472, None],
[6, 5, 5.874, None],
[6, 7, 0.116, 1],
[6, 8, 0.032, 1],
[6, 9, 8.693, None],
[6, 10, 8.597, None],
[6, 11, 8.601, None],
[6, 12, 2.443, None],
[6, 13, 2.364, None],
[6, 14, 2.403, None],
[6, 15, 4.01, None],
[6, 16, 3.995, None],
[6, 17, 4.043, None],
[6, 18, 8.7, None],
[6, 19, 10.146, None],
[6, 20, 8.607, None],
[7, 0, 4.354, None],
[7, 1, 4.28, None],
[7, 2, 4.321, None],
[7, 3, 5.888, None],
[7, 4, 5.551, None],
[7, 5, 5.953, None],
[7, 6, 0.116, 1],
[7, 8, 0.118, 1],
[7, 9, 8.751, None],
[7, 10, 8.655, None],
[7, 11, 8.659, None],
[7, 12, 2.329, None],
[7, 13, 2.25, None],
[7, 14, 2.288, None],
[7, 15, 4.088, None],
[7, 16, 4.073, None],
[7, 17, 4.12, None],
[7, 18, 8.775, None],
[7, 19, 10.216, None],
[7, 20, 8.682, None],
[8, 0, 4.469, None],
[8, 1, 4.396, None],
[8, 2, 4.437, None],
[8, 3, 5.78, None],
[8, 4, 5.442, None],
[8, 5, 5.845, None],
[8, 6, 0.032, 1],
[8, 7, 0.118, 1],
[8, 9, 8.667, None],
[8, 10, 8.571, None],
[8, 11, 8.575, None],
[8, 12, 2.446, None],
[8, 13, 2.367, None],
[8, 14, 2.405, None],
[8, 15, 3.997, None],
[8, 16, 3.983, None],
[8, 17, 4.03, None],
[8, 18, 8.671, None],
[8, 19, 10.118, None],
[8, 20, 8.577, None],
[9, 0, 10.557, None],
[9, 1, 10.506, None],
[9, 2, 10.532, None],
[9, 3, 4.34, None],
[9, 4, 4.696, None],
[9, 5, 4.281, None],
[9, 6, 8.693, None],

```
[9, 7, 8.751, None],
[9, 8, 8.667, None],
[9, 10, 0.225, 1],
[9, 11, 0.104, 1],
[9, 12, 9.092, None],
[9, 13, 9.114, None],
[9, 14, 9.092, None],
[9, 15, 9.618, None],
[9, 16, 9.578, None],
[9, 17, 9.567, None],
[9, 18, 6.261, None],
[9, 19, 4.71, None],
[9, 20, 6.355, None],
[10, 0, 10.72, None],
[10, 1, 10.669, None],
[10, 2, 10.696, None],
[10, 3, 4.508, None],
[10, 4, 4.863, None],
[10, 5, 4.45, None],
[10, 6, 8.597, None],
[10, 7, 8.655, None],
[10, 8, 8.571, None],
[10, 9, 0.225, 1],
[10, 11, 0.218, 1],
[10, 12, 9.251, None],
[10, 13, 9.178, None],
[10, 14, 9.212, None],
[10, 15, 9.786, None],
[10, 16, 9.747, None],
[10, 17, 9.735, None],
[10, 18, 6.435, None],
[10, 19, 4.93, None],
[10, 20, 6.528, None],
[11, 0, 10.543, None],
[11, 1, 10.491, None],
[11, 2, 10.518, None],
[11, 3, 4.302, None],
[11, 4, 4.658, None],
[11, 5, 4.244, None],
[11, 6, 8.601, None],
[11, 7, 8.659, None],
[11, 8, 8.575, None],
[11, 9, 0.104, 1],
[11, 10, 0.218, 1],
[11, 12, 9.067, None],
[11, 13, 9.026, None],
[11, 14, 9.028, None],
[11, 15, 9.599, None],
[11, 16, 9.559, None],
[11, 17, 9.547, None],
[11, 18, 6.224, None],
[11, 19, 4.786, None],
[11, 20, 6.318, None],
[12, 0, 2.776, None],
[12, 1, 2.7, None],
[12, 2, 2.742, None],
[12, 3, 6.282, None],
[12, 4, 5.958, None],
[12, 5, 6.344, None],
[12, 6, 2.443, None],
[12, 7, 2.329, None],
[12, 8, 2.446, None],
[12, 9, 9.092, None],
[12, 10, 9.251, None],
[12, 11, 9.067, None],
[12, 13, 0.082, 1],
[12, 14, 0.06, 1],
[12, 15, 3.487, None],
[12, 16, 3.472, None],
[12, 17, 3.519, None],
[12, 18, 9.051, None],
[12, 19, 10.424, None],
[12, 20, 8.962, None],
[13, 0, 2.856, None],
[13, 1, 2.78, None],
[13, 2, 2.822, None],
[13, 3, 6.315, None],
[13, 4, 5.99, None],
[13, 5, 6.378, None],
[13, 6, 2.364, None],
[13, 7, 2.25, None],
[13, 8, 2.367, None],
[13, 9, 9.114, None],
[13, 10, 9.178, None],
[13, 11, 9.026, None],
[13, 12, 0.082, 1],
[13, 14, 0.066, 1],
[13, 15, 3.534, None],
[13, 16, 3.518, None],
[13, 17, 3.566, None],
[13, 18, 9.083, None],
[13, 19, 10.458, None],
[13, 20, 8.994, None],
[14, 0, 2.816, None],
[14, 1, 2.74, None],
[14, 2, 2.783, None],
[14, 3, 6.298, None],
[14, 4, 5.974, None],
[14, 5, 6.361, None],
[14, 6, 2.403, None],
```

[14, 7, 2.288, None],
[14, 8, 2.405, None],
[14, 9, 9.092, None],
[14, 10, 9.212, None],
[14, 11, 9.028, None],
[14, 12, 0.06, 1],
[14, 13, 0.066, 1],
[14, 15, 3.544, None],
[14, 16, 3.528, None],
[14, 17, 3.575, None],
[14, 18, 9.064, None],
[14, 19, 10.439, None],
[14, 20, 8.975, None],
[15, 0, 2.345, None],
[15, 1, 2.27, None],
[15, 2, 2.312, None],
[15, 3, 5.971, None],
[15, 4, 5.67, None],
[15, 5, 6.037, None],
[15, 6, 4.01, None],
[15, 7, 4.088, None],
[15, 8, 3.997, None],
[15, 9, 9.618, None],
[15, 10, 9.786, None],
[15, 11, 9.599, None],
[15, 12, 3.487, None],
[15, 13, 3.534, None],
[15, 14, 3.544, None],
[15, 16, 0.074, 1],
[15, 17, 0.092, 1],
[15, 18, 8.312, None],
[15, 19, 9.652, None],
[15, 20, 8.221, None],
[16, 0, 2.418, None],
[16, 1, 2.343, None],
[16, 2, 2.385, None],
[16, 3, 5.91, None],
[16, 4, 5.608, None],
[16, 5, 5.976, None],
[16, 6, 3.995, None],
[16, 7, 4.073, None],
[16, 8, 3.983, None],
[16, 9, 9.578, None],
[16, 10, 9.747, None],
[16, 11, 9.559, None],
[16, 12, 3.472, None],
[16, 13, 3.518, None],
[16, 14, 3.528, None],
[16, 15, 0.074, 1],
[16, 17, 0.048, 1],
[16, 18, 8.266, None],
[16, 19, 9.61, None],
[16, 20, 8.174, None],
[17, 0, 2.436, None],
[17, 1, 2.36, None],
[17, 2, 2.403, None],
[17, 3, 5.888, None],
[17, 4, 5.586, None],
[17, 5, 5.954, None],
[17, 6, 4.043, None],
[17, 7, 4.12, None],
[17, 8, 4.03, None],
[17, 9, 9.567, None],
[17, 10, 9.735, None],
[17, 11, 9.547, None],
[17, 12, 3.519, None],
[17, 13, 3.566, None],
[17, 14, 3.575, None],
[17, 15, 0.092, 1],
[17, 16, 0.048, 1],
[17, 18, 8.249, None],
[17, 19, 9.595, None],
[17, 20, 8.158, None],
[18, 0, 10.171, None],
[18, 1, 10.1, None],
[18, 2, 10.144, None],
[18, 3, 4.288, None],
[18, 4, 4.195, None],
[18, 5, 4.247, None],
[18, 6, 8.7, None],
[18, 7, 8.775, None],
[18, 8, 8.671, None],
[18, 9, 6.261, None],
[18, 10, 6.435, None],
[18, 11, 6.224, None],
[18, 12, 9.051, None],
[18, 13, 9.083, None],
[18, 14, 9.064, None],
[18, 15, 8.312, None],
[18, 16, 8.266, None],
[18, 17, 8.249, None],
[18, 19, 1.612, 1],
[18, 20, 0.102, 1],
[19, 0, 11.405, None],
[19, 1, 11.34, None],
[19, 2, 11.381, None],
[19, 3, 4.805, None],
[19, 4, 5.176, None],
[19, 5, 4.742, None],
[19, 6, 10.146, None],

```
[19, 7, 10.216, None],
[19, 8, 10.118, None],
[19, 9, 4.71, None],
[19, 10, 4.93, None],
[19, 11, 4.786, None],
[19, 12, 10.424, None],
[19, 13, 10.458, None],
[19, 14, 10.439, None],
[19, 15, 9.652, None],
[19, 16, 9.61, None],
[19, 17, 9.595, None],
[19, 18, 1.612, 1],
[19, 20, 1.712, 1],
[20, 0, 10.087, None],
[20, 1, 10.016, None],
[20, 2, 10.06, None],
[20, 3, 4.233, None],
[20, 4, 4.14, None],
[20, 5, 4.192, None],
[20, 6, 8.607, None],
[20, 7, 8.682, None],
[20, 8, 8.577, None],
[20, 9, 6.355, None],
[20, 10, 6.528, None],
[20, 11, 6.318, None],
[20, 12, 8.962, None],
[20, 13, 8.994, None],
[20, 14, 8.975, None],
[20, 15, 8.221, None],
[20, 16, 8.174, None],
[20, 17, 8.158, None],
[20, 18, 0.102, 1],
[20, 19, 1.712, 1]]
```

Как можно видеть по набору данных, алгоритм справился с определением одинаковых рук, и не дал ошибочную категорию для разных.

Теперь пишу ряд функций для визуализации данных в табличном виде:

In [90]:

```
import pandas as pd
pd_data = pd.DataFrame(all_scores,columns=['0','1','2','3'])
```

In [91]:

```
pd_data = pd_data.sort_values(by=['0','2'])
data_list = pd_data.values.tolist()
```

In [92]:

```
top_3_imgs = []
i=0
while i < len(data_list):
    temp = []
    try:
        for j in range(0,3):
            temp.append(data_list[i+j][1])
    except:
        print("Program needs minimum of 4 images to give top 3 closest.")
        raise
    top_3_imgs.append(temp)
    i+=(len(all_features)-1)
```

In [93]:

```
same_people = []
i=0
while i < len(data_list):
    temp = [data_list[i][0]]
    for j in range(0,(len(all_features)-1)):
        if data_list[i+j][3] == 1:
            temp.append(data_list[i+j][1])
    same_people.append(temp)
    i+=(len(all_features)-1)
```

In [94]:

```
same_people_n = np.full([len(same_people),len(max(same_people,key = lambda x: len(x)))],-1)
for i,j in enumerate(same_people):
    same_people_n[i][0:len(j)] = j
same_people_n.sort(axis=1)
_,idx = np.unique(same_people_n,axis=0,return_index=True)
same_people_n = same_people_n[idx]
```

In [95]:

```
top_3_for_print = []
for i in range(0,len(top_3_imgs)):
    temp = []
    temp.append(array_of_filenames[i])
    temp.append(array_of_filenames[int(top_3_imgs[i][0])])
    temp.append(array_of_filenames[int(top_3_imgs[i][1])])
    temp.append(array_of_filenames[int(top_3_imgs[i][2])])
    top_3_for_print.append(temp)
```

Демонстрирую таблицу топ 3 схожих рук. Некоторые из изображений подгружены ниже для наглядности (003 и 145, нулевая строка):

In [96]:

```
top_3_for_print_pd = pd.DataFrame(top_3_for_print,columns=['Picture_name','1st_closest','2nd_closest','3rd_closest'])
top_3_for_print_pd
```

Out[96]:

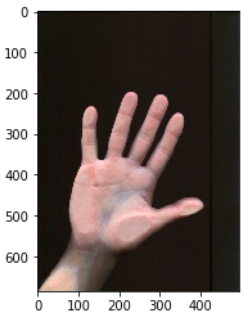
	Picture_name	1st_closest	2nd_closest	3rd_closest
0	003.png	003_2.png	003_1.png	145.png
1	003_1.png	003_2.png	003.png	145.png
2	003_2.png	003.png	003_1.png	145.png
3	006.png	006_2.png	006_1.png	152_2.png
4	006_1.png	006.png	006_2.png	152_2.png
5	006_2.png	006.png	006_1.png	152_2.png
6	031.png	031_2.png	031_1.png	097_1.png
7	031_1.png	031.png	031_2.png	097_1.png
8	031_2.png	031.png	031_1.png	097_1.png
9	057.png	057_2.png	057_1.png	006_2.png
10	057_1.png	057_2.png	057.png	006_2.png
11	057_2.png	057.png	057_1.png	006_2.png
12	097.png	097_2.png	097_1.png	031_1.png
13	097_1.png	097_2.png	097.png	031_1.png
14	097_2.png	097.png	097_1.png	031_1.png
15	145.png	145_1.png	145_2.png	003_1.png
16	145_1.png	145_2.png	145.png	003_1.png
17	145_2.png	145_1.png	145.png	003_1.png
18	152.png	152_2.png	152_1.png	006_1.png
19	152_1.png	152.png	152_2.png	057.png
20	152_2.png	152.png	152_1.png	006_1.png

In [97]:

```
imshow(array(Image.open(array_of_paths[0])))
```

Out[97]:

```
<matplotlib.image.AxesImage at 0x172502a5608>
```

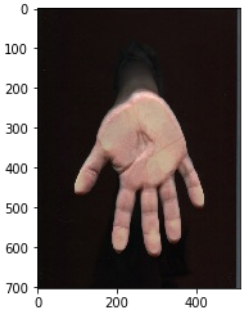


In [98]:

```
imshow(array(Image.open(array_of_paths[15])))
```

Out[98]:

```
<matplotlib.image.AxesImage at 0x172502a5d88>
```



Теперь генерирую таблицу персон с наименованиями изображений их рук (даются наименования файлов, а не полного пути):

In [99]:

```
persons_for_print = []
for i in range(0,len(same_people_n)):
    temp = []
    temp.append(i+1)
    for j in range(0,len(same_people_n[i])):
        if int(same_people_n[i][j]) != -1:
            temp.append(array_of_filenames[int(same_people_n[i][j])])
        else:
            temp.append(None)
    persons_for_print.append(temp)
```

In [100]:

```
cl_n = ['Person_num']
for i in range(1,len(persons_for_print[0])):
    cl_n.append(("Image_" + str((i))))
persons_for_print_pd = pd.DataFrame(persons_for_print,columns=cl_n)
persons_for_print_pd
```

Out[100]:

	Person_num	Image_1	Image_2	Image_3
0	1	003.png	003_1.png	003_2.png
1	2	006.png	006_1.png	006_2.png
2	3	031.png	031_1.png	031_2.png
3	4	057.png	057_1.png	057_2.png
4	5	097.png	097_1.png	097_2.png
5	6	145.png	145_1.png	145_2.png
6	7	152.png	152_1.png	152_2.png

При желании, данные таблицы можно сохранять на жесткий диск в нужном формате, но условие задания этого не требовало. Удаляю временные файлы:

In [108]:

```
for i in array_of_paths:
    os.remove(i)
```

Выводы

Данная работа демонстрирует, что классификация людей по руке в целом осуществимая задача, одна обладающая большим количеством "подводных камней." Использование в целом простых операций, таких как бинаризация, морфология, фильтрация, а также выделение контуров позволило получить удовлетворительный результат на выбранных для теста изображений даже не имея "обучающих" данных (то есть по сути решается задача создания классификатора без учителя), однако о точности работы алгоритма в целом можно судить только по результатам оценок минимум десятков тысяч различных изображений (что находится за пределами данной лабораторной работы). При этом, конкретные параметры операций были подобраны экспериментальным путем. Кроме того, скорость работы созданного алгоритма прямо зависит от количества изображений для анализа, т.к. каждое изображение сравнивается с каждым. Для облегчения работы с большими наборами изображений, можно реализовать функцию сохранения полученных векторов признаков и их последующего использования, но данная функциональность находится за пределами условий данной лабораторной работы.

Для адекватной классификации ладоней требуется соблюдения ряда условий, которые позволят значительно упростить данную задачу. Первое важное условие - все руки должны быть в фиксированном положении, чтобы не было вариаций в положении пальцев для одного и того же человека, и не нужно было компенсировать возможные "развернутые" изображения. Желательно, чтобы на изображении не было видно кисти руки и каких-либо "посторонних" объектов, для исключения дополнительной операции по фильтрации.

Для анализа рисунка ладони необходимо, чтобы все люди, сканирующие ладони, прикладывали только какую-либо одну руку, либо каждую поочередно, в противном случае изучать рисунок не имеет смысла, т.к. он будет разным на правой и левой руках. Также необходимо очень высокое разрешение изображения, т.к. выловить отдельные линии не представляется возможным. Следует учитывать, что сила давления на сенсор также будет влиять на видимость рисунка ладони. В целом можно сказать, что использовать рисунок ладони для классификации следует с осторожностью, т.к. слишком много факторов может оказать на него влияние.