

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ
им. М.В. ЛОМОНОСОВА

факультет вычислительной математики и кибернетики
кафедра математических методов прогнозирования

Отчет
по лабораторной работе № 2:

«Изучение и освоение
методов классификации
формы изображений»

Мищустина Маргарита Владимировна
Обработка и распознавание изображений
3 курс, группа 317

Москва, 2020 г.

Содержание

1 Формулировка задания	2
2 Описание данных	2
3 Этапы решения	3
3.1 Этапы решения: определение границ	3
3.2 Этапы решения: построения выпуклой оболочки.	3
3.3 Этапы решения: использование геометрических соображений.	5
3.4 Этапы решения: создании линии ладони.	7
3.5 Предварительные выводы из первой части работы.	8
4 Построение признакового пространства и поиск ближайших соседей	8
5 Построение признакового пространства и кластеризация	9
6 Описание программной реализации	12
6.1 Нахождение точек ладони	12
6.2 Сортировка полученных точек для удобства визуализации . .	14
6.3 Нахождение центра ладони	15
6.4 Поиск ближайших соседей	16
6.5 Кластеризация	17
7 Выводы	18
8 Приложение: некоторые результаты работы	19
8.1 Нахождение ближайших соседей и кластеризация.	19

1 Формулировка задания

В данной лабораторной работе требовалось разработать и реализовать программу для работы с изображениями ладоней. Написанная программа должна была обеспечивать:

- * Ввод и отображение изображений на экране;
- * Сегментацию изображений на основе точечных и пространственных преобразований;
- * Генерацию признаковых описаний формы ладоней на изображениях;
- * Вычисление меры сходства ладоней;
- * Кластеризацию изображений.

В рамках предусмотренных уровней сложности задания, работа выполнялась для уровня «Expert», то есть основной целью было не только определить точки ладони, но и на основе полученных признаков - расстояний от кончиков пальцев до их оснований - построить модель, которая для каждой ладони находила бы 3 ближайших к ней, а также кластеризовала данные изображения, выясняя, какие руки принадлежат одному человеку.

Для решения задачи использовалась среда Jupyter Notebook с использованием языка программирования Python и библиотек numpy, scikit-learn и OpenCV. В настоящем отчете будут продемонстрированы все рассмотренные подходы к решению, а также их результаты.

2 Описание данных

Для тестирования программы был предоставлен набор фотографий ладоней, сделанных с использованием сканера. Примеры входных изображений представлены на рис.1.

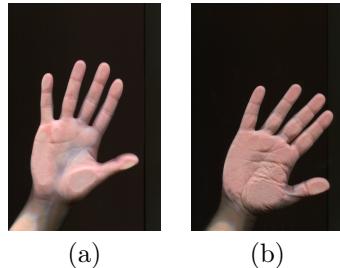


Рис. 1: Примеры изображений для сегментации и кластеризации.

Данная выборка имеет ряд особенностей: ладони расположены внутренней стороной вниз на черном фоне, при этом они имеют разный угол наклона на фотографиях, что значительно усложняет задачу их сегментации.

3 Этапы решения

3.1 Этапы решения: определение границ

Изначально изображение ладони было бинаризовано. Результаты бинаризации для ладоней на рис. 1 представлены на рис. 2.



Рис. 2: Результаты бинаризации для некоторых изображений ладони.

Для бинаризации использовался простейший порог THRESH_BINARY со значением 45. Данный выбор был обусловлен тем, что чаще всего светлая ладонь сильно контрастирует с черным фоном картинки, из-за чего отдельить руку от фона довольно легко. С этим отлично справляется выбранный порог.

3.2 Этапы решения: построения выпуклой оболочки.

После бинаризации на изображении легко получилось выделить контуры, а также найти среди них максимальный по площади и принять его за контур руки. Данного шага можно было бы избежать, однако тогда метод оказался бы сильно неустойчив к изображениям с шумом, поскольку на такого типа картинках выделялось бы несколько контуров.

Далее для полученного контура была построена выпуклая оболочка. Результаты данных действий представлены на рис.3

Выпуклая оболочка была построена для удобства дальнейшего нахождения кончиков пальцев и точек их основания. Библиотека OpenCV предоставляет ряд средств для работы с выпуклыми оболочками, которые могут быть использованы для решения задачи. В частности, она помогает найти так называемые «дефекты выпуклости», то есть те точки контура объекта (ладони), которые расположены на максимальном удалении от соответствующей

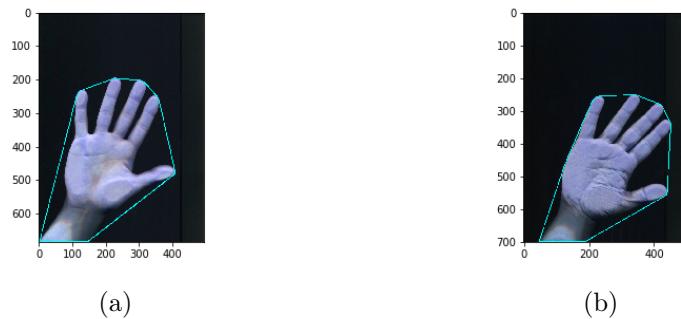


Рис. 3: Результаты построения выпуклой оболочки для некоторых изображений ладони.

выпуклой оболочки. Попутно вместе с данными «дефектами» находятся и те точки выпуклой оболочки, для которых данное расстояние максимально. Среди первой группы точек на контуре ладони следует искать основания пальцев, а в число точек на оболочке входят координаты кончиков.

Пример, поясняющий значение понятия «дефекты выпуклости» представлен на рис. 4.



Рис. 4: Пояснение термина «дефекты выпуклости».

3.3 Этапы решения: использование геометрических соображений.

Стоит отметить, что, как видно из рис. 5, не все полученные в результате работы алгоритма точки являются нужными. Поэтому стоит разобраться, какие из точек подойдут для решения, а какие - нет.

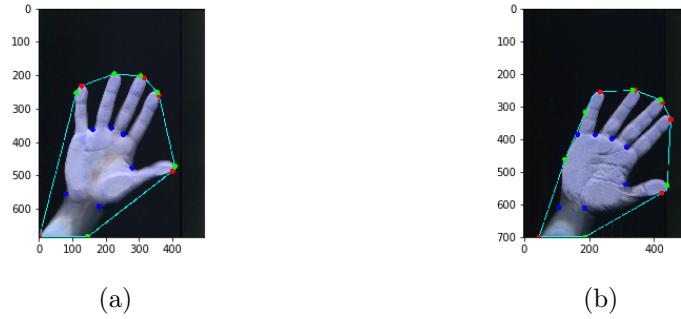


Рис. 5: Результаты определения дефектов выпуклой оболочки.

На рис. 5 различными цветами обозначены параметры каждого «дефекта» (начало и конец - то есть точки выпуклой оболочки, имеющие пересечение с ладонью, от которых предполагаемое основание пальца располагается на максимальном расстоянии, а также само основание). Из данного рисунка видно, что осталось лишь отобрать нужные точки. Это будет сделано с помощью геометрических соображений.

Для начала будет рассмотрен треугольник со следующими вершинами: начало (start), конец (end) и предполагаемое основание пальца (far) - одна такая тройка изображена на рис. 6.

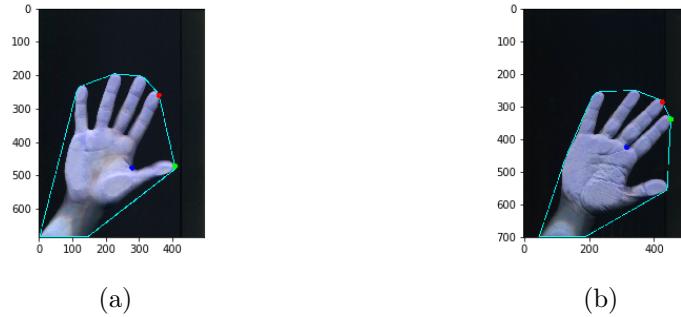


Рис. 6: Результаты определения дефектов выпуклой оболочки.

Зная координаты вершин треугольника, найдем его стороны a, b, c, как:

$$a = \sqrt{((end[0] - start[0])^2 + (end[1] - start[1])^2)}$$

$$b = \sqrt{((far[0] - start[0])^2 + (far[1] - start[1])^2)}$$

$$c = \sqrt{((end[0] - far[0])^2 + (end[1] - far[1])^2)}$$

Далее с помощью формулы Герона можно вычислить площадь данного треугольника:

$$s = \sqrt{(p * (p - a) * (p - b) * (p - c))}$$

где

$$p = \frac{(a + b + c)}{2}$$

- полупериметр

Итак, мы получили треугольник, его площадь и периметр. Как теперь понять, являются ли его вершины теми точками ладони, которые нам нужны? Для начала с использованием теоремы косинусов вычислим угол между предполагаемыми пальцами (то есть сторона b и c треугольника):

$$angle = \arccos(b^2 + c^2 - a^2) / (2 * b * c)$$

Если данный угол окажется больше 90 градусов, то такая тройка точек нам не подойдет: это будет означать, что искомые точки расположены где-то на ребре, либо в основании ладони. Данный факт следует из очевидных соображений о том, что угол между двумя пальцами человека не может превысить 90 (в крайнем случае 100) градусов.

Кроме того вычислим (из соображения того, что площадь треугольника равна половине произведения высоты на основание) следующую величину, которая означает длину перпендикуляра, проведенного от точки, подозрительной на основание пальца, к противоположной стороне построенного на-ми треугольника:

$$d = \frac{(2 * ar)}{a}$$

Если ее величина будет меньше значения некоторого порога, то это означает, что точка основания пальца расположена слишком близко к выпуклой оболочке. Тогда она не может являться основанием пальца.

В ходе проведенных рассуждений в силу особенности реализации (на каждом шаге отмечались только основание и start, чтобы не произошло наложение start и end из разных дефектов друг на друга) были правильно отмечены все основания и кончики четырех пальцев. Оставшийся палец (мизинец) отметим следующим образом: рассмотрим все точки end, которые мы не отмечали на протяжении работы алгоритма, однако сохраняли. Из данных точек выберем ту, которая находится на наибольшем расстоянии от всех остальных пальцев. Это и будет оставшийся пятый кончик пальца, который и завершит разметку точек ладони. На самом деле, если бы все ладони были одинаково ориентированы, то можно было бы просто рассмотреть точку из последнего дефекта (удовлетворяющего, конечно, условиям, вытекающим из геометрических соображений). Однако, поскольку это не так, стоит проверять, какая именно точка является кончиком нужного пальца.

Результаты работы алгоритма см. на рис.7.



Рис. 7: Результаты определения кончиков и оснований пальцев.

3.4 Этапы решения: создании линии ладони.

Итак, точки были получены корректно. Далее следовало соединить их ломаной так, чтобы они образовывали линию ладони (кроме того, данное действие нужно было произвести и для того, чтобы точки оказались расположены подряд по отношению к ладони, а также начинались с одной точки - кончика мизинца; очевидно, что обратное привело бы к некорректности признаков). Из-за, опять же, того, что ладони могут быть повернуты на любой градус, делать это, последовательно соединяя отмеченные точки, оказалось неверным решением, поэтому был придуман следующий алгоритм:

Поскольку последним пальцем всегда на всех ладонях являлся мизинец (он добавлялся в конце вручную), то оставшиеся пальцы и их основания сортировались по увеличению расстояния до мизинца. Таким образом, начиная с мизинца, и строилась линия, соединявшая упорядоченные точки, каждые из которых соответствовали друг другу. Результаты представлены на рис.8.



Рис. 8: Результаты нахождения линии ладони.

3.5 Предварительные выводы из первой части работы.

В ходе проведенной работы задача была полностью выполнена для уровня «Intermediate». Алгоритм корректно работает на 98/99 изображений и на оставшемся изображении требуется немного изменить порог значения угла (с 90 до 100 градусов) для правильного определения точек и линии.

4 Построение признакового пространства и поиск ближайших соседей

В ходе проведенных действий были получены точки кончиков и оснований пальцев. В признаковое пространство были добавлены 8 длин соответствующих линий, соединяющих соответствующие точки (обозначены розовым цветом на рис. 8).

Далее были посчитаны расстояния между основаниями пальцев, а также выделен центр ладони и произведены вычисления расстояний от кончиков и оснований пальцев до данной середины. Кроме того, благодаря средствам библиотеки OpenCV, были получены площадь ладони, ее периметр, а также средний цвет. В дальнейшем рассматривались различные комбинации полученных признаков для получения наилучшего результата. Кроме того, при использовании только признаков, обозначающих линии, значения не нормировались, в то время как при добавлении площади, периметра и цвета все признаки были нормированы для приведения к одному масштабу.

В ходе экспериментов было выяснено, что лучший результат поиска соседей достигался при использовании исходных длин ломаной линии ладони, расстояний от центра ладони до кончиков пальцев, а также площади, периметра и цвета ладони.

Пример вычисления ближайших соседей показан на рисунке 9.

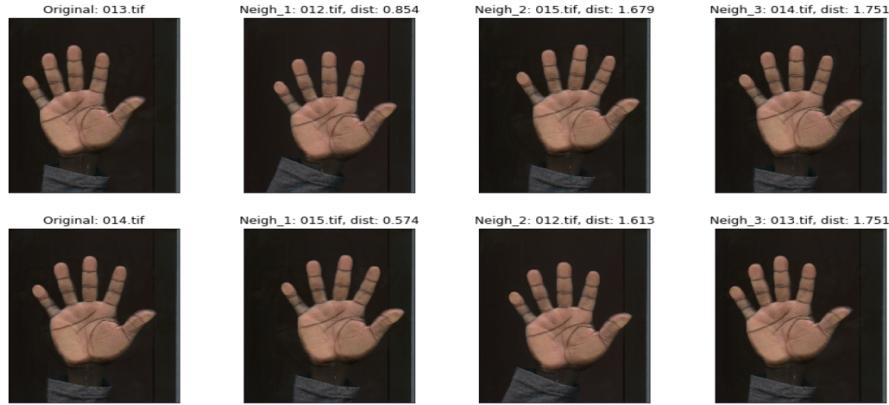


Рис. 9: Пример поиска ближайших соседей

Отметим, что полученные, но неиспользованные признаки оказывались либо излишними, либо необъективными: поскольку если измерить расстояние от центра ладони до кончиков пальцев и от кончиков до оснований, то при добавлении расстояний от центра до оснований ничего не изменится, а что касается расстояний между основаниями - они могут меняться в зависимости от расположения пальцев. Значения периметра, площади и цвета ладони могут так же быть неподходящими, поскольку рука может быть рассмотрена под разным освещением, помимо того содержать запястье или не содержать. Однако на конкретных примерах модель, учитывавшая данные признаки, показала себя лучше других.

5 Построение признакового пространства и кластеризация

Для кластеризации были рассмотрены следующие стандартные средства библиотеки scikit-learn:

- * `sklearn.cluster.OPTICS`;
- * `sklearn.cluster.AffinityPropagation`;
- * `sklearn.cluster.KMeans`;

Первые 2 метода отличаются тем, что для них не нужно указывать число кластеров, а лишь следует настроить параметры. Что касается третьего метода, то он наиболее интуитивно понятен, однако для его успешной работы следует указывать потенциальное число кластеров, которое мы хотим получить.

Метод AffinityPropagation в общем неплохо справляется с кластеризацией и кажется в данном случае наиболее подходящим, поскольку не оставляет выбросов и распределяет все объекты по кластерам. Однако при любом выборе параметров данный метод выдает число кластеров 12 и при дальнейшем рассмотрении выясняется, что полученные кластеры не совсем адекватны. Метод OPTICS выдает гораздо лучшие по комплектации классы, однако оставляет выбросы. Производить повторную обработку выбросов тем же методом не вполне корректно, поскольку, во-первых, выбросы не получат возможность оказаться в одном кластере с уже распределенными объектами и, во-вторых, новая итерация метода повлечет за собой создание еще объектов-выбросов. Поэтому возникла идея взять количество кластеров, выдаваемых данным методом (21 - 25) при разном подборе параметров и рассмотреть значение количества кластеров 22 (в таком случае результат получался наилучшим), поскольку некоторые объекты-выбросы породят новые кластеры, а некоторые - отнесутся к старым. Далее был применен метод KMeans с уже указанным параметром числа кластеров и результат оказался хорошим. Пример некоторых кластеров см. на рис. 5. Стоит отметить, что для

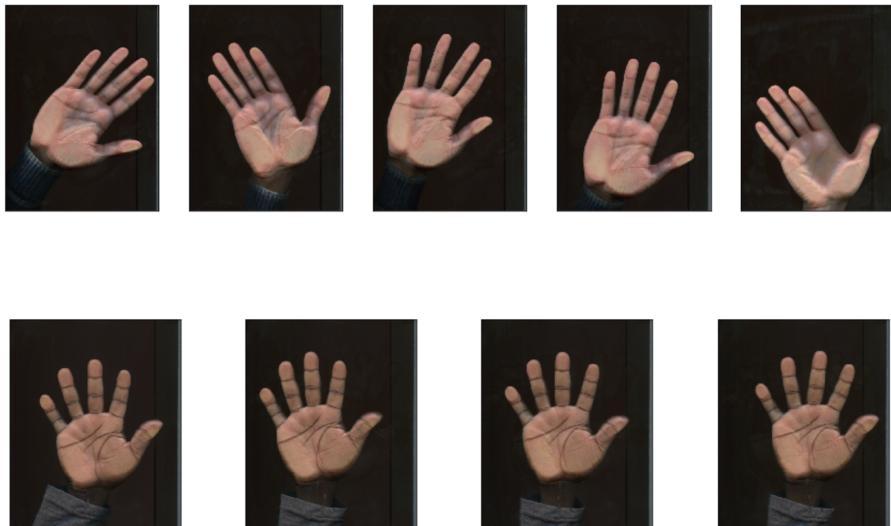


Рис. 10: Результаты кластеризации.

выбранного числа кластеров получилось так, что в каждом кластере ладони максимально похожи между собой, однако некоторые кластеры требу-

ют объединения. Но при рассмотрении меньшего числа классов результат ухудшается, а что касается метода AffinityPropagation, то выдаваемых им кластеров - напротив - недостаточно. Из-за того что для данных нет меток о принадлежности тому или иному классу, а зрительно довольно трудно определить, сколько людей принимало участие в создании выборки, нет точного критерия качества, способного определить корректность классификации. Однако что касается выполненного метода, результаты которого можно видеть на рис. 5 и в приложении на рис. 8.1, он кажется довольно успешным и подходящим для данной задачи.

6 Описание программной реализации

6.1 Нахождение точек ладони

Рассмотрим программную реализацию алгоритма, описанного ранее. Код функции представлен без учета визуализации и с опущением некоторых моментов, не обязательных для понимания работы алгоритма. Комментарии к действиям представлены непосредственно в коде.

```
1 def find_fingers(image):
2     # делаем изображение серым для бинаризации
3     gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
4
5     # бинаризация
6     _, thresh = cv2.threshold(gray, 45, 255, cv2.THRESH_BINARY)
7     # убираем шумы с помощью эрозии и дилатации
8     thresh = cv2.erode(thresh, None, iterations=2)
9     thresh = cv2.dilate(thresh, None, iterations=2)
10
11    # ищем контуры (для сокращения записи уберем параметры)
12    cnts = cv2.findContours(thresh.copy(), *args)
13    cnts = imutils.grab_contours(cnts)
14
15    # выбираем контур, соответствующий ладони
16    hand = max(cnts, key=cv2.contourArea)
17
18    n = hand.copy()
19
20    # параметр для approx
21    epsilon = 0.0005 * cv2.arcLength(n,True)
22
23    # строим описанный многоугольник
24    # с учетом особенностей формы
25    approx = cv2.approxPolyDP(n,epsilon,True)
26
27    # строим выпуклую оболочку по ранее
28    # построенному многоугольнику
29    hull = cv2.convexHull(approx, returnPoints=False)
30    # вычисляем дефекты выпуклости
31    defects = cv2.convexityDefects(approx, hull)
32    l = 0
33
34    point_between = []
35    end_arr = []
36    fingers = []
37    # смотрим по всем дефектам (лок максимумам)
38    for i in range(defects.shape[0]):
39        s,e,f,d = defects[i,0]
```

```

40     start = tuple(approx[s][0])
41     end = tuple(approx[e][0])
42     far = tuple(approx[f][0])
43
44     # поиск длин векторов по координатам
45     a = math.sqrt((end[0] - start[0])**2 + (end[1] - start[1])**2)
46     b = math.sqrt((far[0] - start[0])**2 + (far[1] - start[1])**2)
47     c = math.sqrt((end[0] - far[0])**2 + (end[1] - far[1])**2)
48     # полупериметр
49     p_p = (a + b + c) / 2
50     # вычисление площади
51     ar = math.sqrt(p_p * (p_p - a) * (p_p - b) * (p_p - c))
52
53     # расстояние между точкой и выпуклой оболочкой
54     d = (2 * ar) / a
55
56     # устанавливаем порог
57     porog = 87
58
59     # теорема косинусов
60     angle = math.acos((b**2 + c**2 - a**2) / (2 * b * c)) * 57
61
62     # если не нашли еще нужные 4 тройки точек
63     if l < 4:
64         if angle <= porog and d > 30:
65             l += 1
66             point_between.append(far)
67             end_arr.append(end)
68             fingers.append(start)
69
70     # нашли 4 пальца: осталось добавить мизинец
71     min_dist = 1000000
72     for e in end_arr:
73         k = 0
74         for s in fingers:
75             if abs(s[0] - e[0]) > 30 or abs(s[1] - e[1]) > 30:
76                 k += 1
77         if k == 4:
78             cv2.circle(image, e, 8, [255,127,0], -1)
79             fingers.append(e)
80             break
81     return fingers, point_between, hull, end_arr

```

6.2 Сортировка полученных точек для удобства визуализации

```
1  def draw_cont(fingers, point_between):
2      dist_arr = {}
3      dist_p = {}
4      # расстояния от мизинца до остальных пальцев
5      for finger in fingers[:4]:
6          dist_arr[finger] = dist(finger, fingers[4])
7
8      # расстояния от мизинца до оснований пальцев
9      for point in point_between:
10         dist_p[point] = dist(point, fingers[4])
11
12     new_f = fingers[:4]
13     new_d = point_between
14     # сортируем по удалению от мизинца
15     new_f.sort(key=lambda x: dist_arr[x])
16     new_d.sort(key=lambda x: dist_p[x])
17
18     return new_f, new_d, fingers[4]
```

6.3 Нахождение центра ладони

Поскольку в качестве дополнительного признака рассматривались расстояния от кончиков пальцев до центра ладони, то нужно было решить подзадачу, которая заключалась в нахождении центра ладони. Для того чтобы центр определялся более корректно, чем просто центр контура, к изображению последовательно применялись эрозии, которые убирали пальцы и запястье, тем самым оставляя только среднюю часть руки, по которой и считался центр.

```
1  def center(image):
2      # все операции, кроме сильной эрозии, были рассмотрены ранее
3      gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
4      _, thresh = cv2.threshold(gray, 45, 255, cv2.THRESH_BINARY)
5      thresh = cv2.erode(thresh, None, iterations=60)
6
7      cnts = cv2.findContours(thresh.copy(), *args)
8      cnts = imutils.grab_contours(cnts)
9
10     hand = max(cnts, key=cv2.contourArea)
11     rot_rect = cv2.minAreaRect(hand)
12
13     return rot_rect[0]
```

6.4 Поиск ближайших соседей

Для поиска ближайших соседей использовались собственноручно написанные функции для подсчета расстояний между матрицами, а так же ее сортировки и дальнейшей визуализации.

```
1      # подсчет матрицы попарных расстояний
2      def euclidean_distance(X, Y):
3          X_sum = np.sum(X * X, axis=1)
4          Y_sum = np.sum(Y * Y, axis=1)
5          XY_sum = np.dot(X, Y.T)
6
7          X_new = np.tile(X_sum[:, np.newaxis], (1, Y.shape[0]))
8          Y_new = np.tile(Y_sum[:, np.newaxis], (1, X.shape[0])).T
9
10         return (X_new - 2 * XY_sum + Y_new) ** 0.5
11
12     # нахождение расстояний и ближайших соседей
13     def count_euclidean(m, idx=np.arange(0, 23), norm=False):
14         m = m[:,idx]
15         if norm:
16             m = (m - np.mean(m, axis=0)) / np.std(m, axis=0)
17
18         dist_1 = euclidean_distance(m, m)
19         # на диагональ устанавливаем большие числа, чтобы
20         # элемент не был своим ближайшим соседом
21         for j in range(99):
22             dist_1[j, j] = 1000000
23
24         dist_3 = np.argsort(dist_1, axis=1)[:, :3]
25         return dist_1, dist_3
```

6.5 Кластеризация

Для кластеризации использовались упомянутые выше стандартные средства библиотеки sklearn.

```
1 def cluster(for_neigh):
2     model = OPTICS(min_samples = 2, metric = euclidean)
3     model = model.fit(for_neigh)
4     labels = model.labels_
5     num_classes = np.unique(labels).shape[0]
6
7     model = KMeans(n_clusters=num_classes + 2)
8     model = model.fit(for_neigh)
9     predicted_label = model.predict(for_neigh)
10    return predicted_label
```

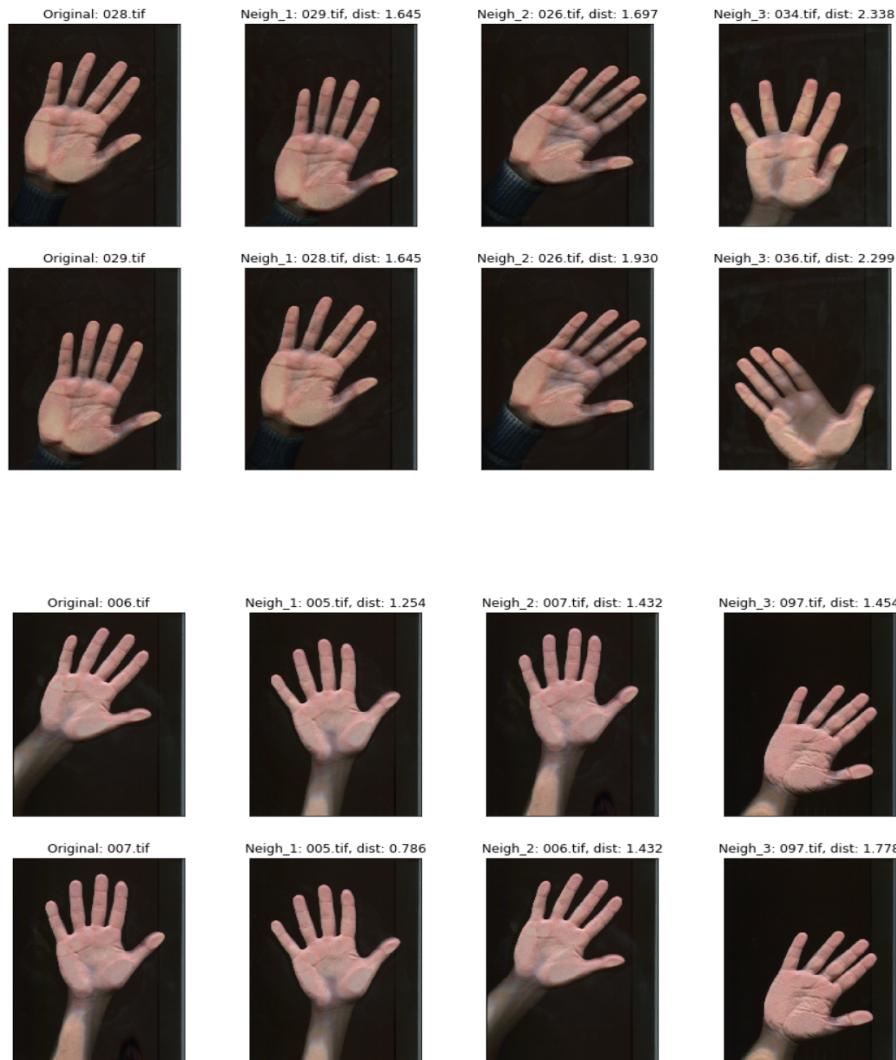
7 Выводы

В ходе изучения задачи, а также возможных методов ее решения, были сделаны следующие выводы:

1. Для работы с изображениями лучше всего подходят средства библиотеки OpenCV. Она позволяет решать многие подзадачи в ходе рассмотрения большого проекта.
2. Для работы с задачами подобного рода очень важен математический аппарат: выпуклые оболочки, простейшие теоремы из геометрии.
3. Расстояния между особыми точками ладони являются объективным, но не достаточным признаком для однозначного определения принадлежности руки тому или иному человеку. Поэтому данный признак следует рассматривать лишь в совокупности с некоторыми другими для достижения более качественного результата.
4. Задача кластеризации является довольно сложной, особенно когда речь идет о распределении объектов на неизвестное число классов и отсутствии некоторой метрики, подтверждающей высокое качество работы программы.
5. Даже данная версия программы, проверенная на датасете из небольшого числа примеров, может быть использована для определения, присутствует, к примеру, человек в некоторой базе или нет. Это утверждение сделано на основе успешных результатах поиска ближайших соседей, поскольку как минимум 2 из них определяются верно во всех экспериментах.

8 Приложение: некоторые результаты работы

8.1 Нахождение ближайших соседей и кластеризация.



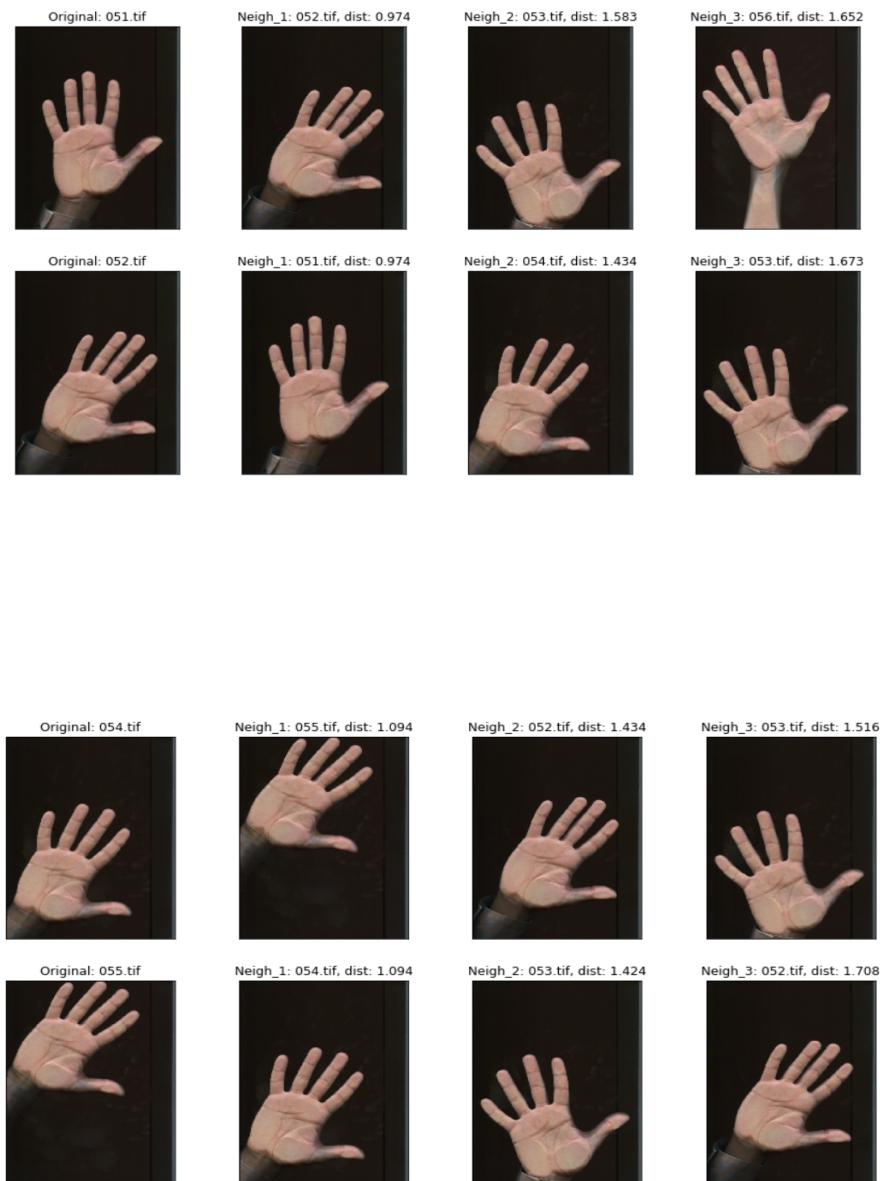
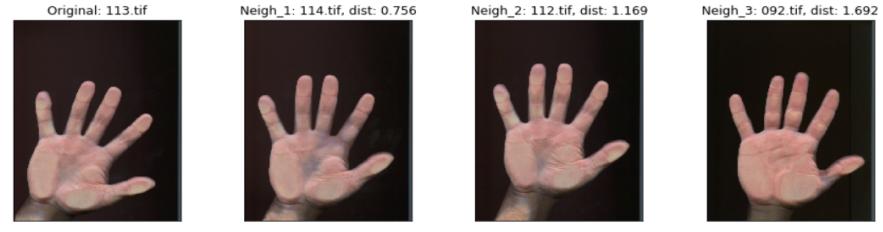
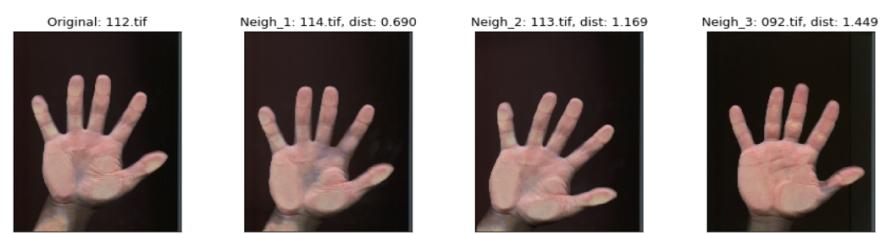
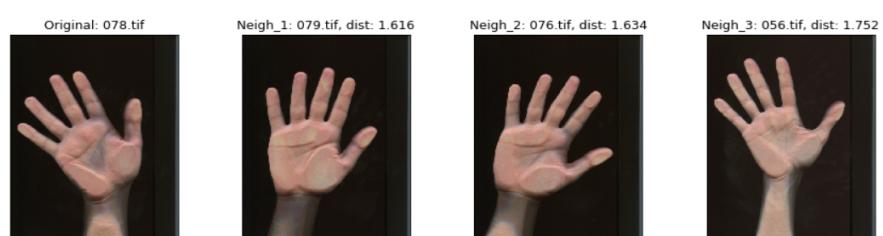


Рис. 11: Результаты нахождения ближайших соседей.



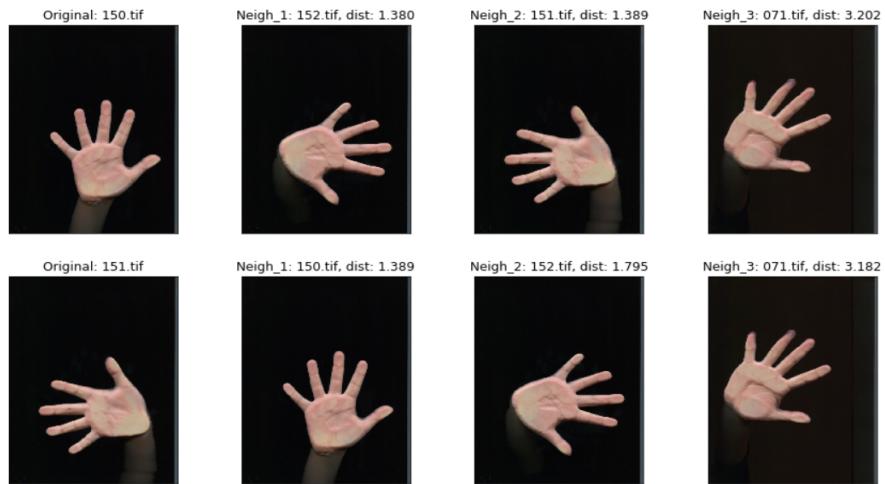
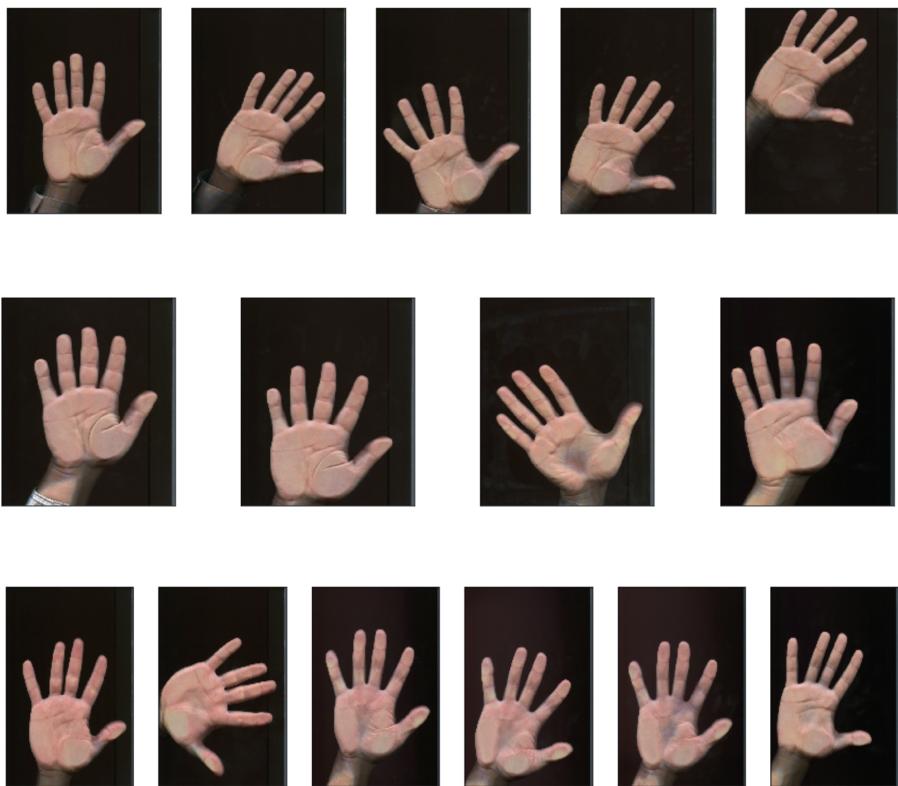


Рис. 12: Результаты нахождения ближайших соседей



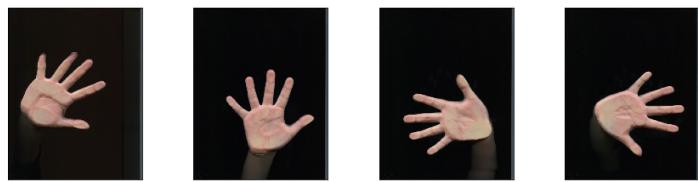


Рис. 13: Результаты кластеризации.