Метод предобработки изображения кисти руки в системе распознавания жестовых символов

Танцевов Г. М.[a]

[a] *МГТУ им. Н. Э. Баумана, Москва, Россия*

Аннотация

Рассмотрены методы обработки изображения с целью выявления наиболее применимого на этапе предобработки изображения в задаче распознавания жестовых символов. Были поставлены эксперименты для сравнения работы описанных алгоритмов, на основании результатов которых был проведен сравнительный анализ. Были выделены два метода: выделение силуэта и построение скелета по ключевым точкам. Предложены возможные пути их улучшения для дальнейшего построения метода распознавания жестовых символов.

Ключевые слова: жесты, выделение границ, скелет кисти, выделение контура

1. Введение

Одной из перспективных задач машинного зрения является распознавание жестовых символов. Актуальность данной темы обусловлена множеством сфер применения: от новых методов взаимодействия с ПК до систем распознавания жестовых языков. Как правило, такие системы состоят из трех частей:

1. Получение данных о жесте
2. Предобработка данных
3. Классификация

В качестве исходных данных можно использовать снимок с камеры, например, смартфона. В этом случае, для упрощения классификации, важен этап предобработки данных. На данном этапе необходимо выделить на изображении основные признаки исходного жеста. Алгоритмы, применимые для достижения данной цели, можно разделить на следующие группы:

- Выделение контура фигуры
- Выделение силуэта кисти руки
- Построение скелета кисти руки

Электронная почта: email_tantsevov@gmail.com (Танцевов Г. М.)

2. Выделение контура фигуры

Для выделения контура кисти руки можно использовать операторы преобразования изображения. К таким методам можно отнести:

- Оператор Собеля
- Оператор Прюитт
- Перекрестный оператор Робертса
- Оператор Кэнни

Рассмотрим каждый подробнее:

2.1. Оператор Собеля

Основная идея оператора Собеля[1] заключается в вычислении градиента освещенности каждой точки изображения. Вычисление производится примерное с помощью свертки изображения двумя сепарабельными целочисленными фильтрами размера 3x3 в вертикальном и горизонтальном направлениях. Благодаря этому вычисление работа данного оператора имеет низкие трудозатраты. В результате получаются два новых изображения Gx и Gy, в каждой точке которого записано приближенное значение производных по x и по y соответственно. Пусть A - исходное изображение, тогда вычисляются они следующим образом:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{1}$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{2}$$

Определение данных матриц на языке Python выглядит следующим образом:

```
sobelx = [[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]]
sobely = [[1, 2, 1], [0, 0, 0], [-1, -2, -1]]
```

Алгоритм 1: Определение матриц свертки оператора Собелья

В итоге значение градиента вычисляется как $G = \sqrt{G_x^2 + G_y^2}$, а его направление как $\theta = \arctan(\frac{G_x}{G_y})$.

Для получения результата применем матрицы к изображению:

```
for row in range(self.width-len(sobelx)):
    for col in range(self.height-len(sobelx)):
        gx = 0
        gy = 0
        for i in range(len(sobelx)):
            for j in range(len(sobely)):
                val = mat[row+i, col+j] * lin_scale
                gx += sobelx[i][j] * val
                gy += sobely[i][j] * val
        pixels[row+1, col+1] = int(math.sqrt(gx*gx + gy*gy))
```

Алгоритм 2: Свертка изображения оператором Собеля

Результат показывает скорость изменения яркости изображения в конкретной точке, т.е. вероятность ее нахождения на границе изображения.

2.2. Оператор Прюитт

Принцип работы[2] аналогичен оператору Собеля, используются только другие ядра свертки:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \tag{3}$$

$$G_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \tag{4}$$

Отсюда реализация данного оператора отличается от реализации оператора Собеля только определением матриц:

```
prewittx = [[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]]
prewitty = [[1, 1, 1], [0, 0, 0], [-1, -1, -1]]
```

Алгоритм 3: Определение матриц свертки оператора Прюитт

2.3. Перекрестный оператор Робертса

Данный алгоритм[3] для каждого пиксела вычисляет сумму квадратов разниц со смежным ему диагональным пикселем. Данную операцию можно представить в виде свертки изображения двумя ядрами размера 2x2:

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \tag{5}$$

Подход данного метода не сильно отличается от двух рассмотренных ранее, в следствии чего его реализация выглядит следующим образом:

```
robertsx = [[1,0],[0,-1]]
robertsy = [[0,1],[-1,0]]
lin_scale = .7
for row in range(self.width-len(robertsx)):
    for col in range(self.height-len(robertsy)):
        Gx = 0
        Gy = 0
        for i in range(len(robertsx)):
            for j in range(len(robertsy)):
                val = mat[row+i, col+j] * lin_scale
                Gx += robertsx[i][j] * val
                Gy += robertsy[i][j] * val
        pixels[row+1,col+1] = int(math.sqrt(Gx*Gx + Gy*Gy))
```

Алгоритм 4: Реализация перекрестного оператора Робертса

В результате получается изображение пространственного градиента исходного изображения, где точки с наибольшим значением соответствуют границе.

2.4. Оператор Кэнни

Данный фильтр[4] был разработан с учетом удовлетворения следующих свойств:

- хорошее обнаружение (Кэнни трактовал это свойство как повышение отношения сигнал/шум);
- хорошая локализация (правильное определение положения границы);
- единственный отклик на одну границу.

Алгоритм состоит из пяти последовательных шагов:

1. Размытие изображения для удаления лишнего шума.
2. Поиск градиентов, для определения границ с максимальным значением градиента.
3. Подавление не-максимумов, т.е. для границ берутся исключительно локальные максимумы.
4. Определение потенциальных границ с помощью двойной пороговой фильтрации.
5. Трассировка области неоднозначности

Для получения размытого изображения можно использовать стандартную функцию *convolve*() из пакета $scipy.ndimage.filters$. Для поиска градиентов подойдет оператор Собеля, описанный выше.

Реализация подавления не-максимумов:

```
def non_max_suppression(img, D):
    M, N = img.shape
    Z = np.zeros((M, N), dtype=np.int32)
    angle = D * 180. / np.pi
    angle[angle < 0] += 180
    for i in range(1, M - 1):
        for j in range(1, N - 1):
            try:
                q = 255
                r = 255
                # angle 0
                if (0 <= angle[i, j] < 22.5) or
                (157.5 <= angle[i, j] <= 180):
                    q = img[i, j + 1]
                    r = img[i, j - 1]
                # angle 45
                elif 22.5 <= angle[i, j] < 67.5:
                    q = img[i + 1, j - 1]
                    r = img[i - 1, j + 1]
                # angle 90
                elif 67.5 <= angle[i, j] < 112.5:
                    q = img[i + 1, j]
                    r = img[i - 1, j]
                # angle 135
                elif 112.5 <= angle[i, j] < 157.5:
                    q = img[i - 1, j - 1]
                    r = img[i + 1, j + 1]
                if (img[i, j] >= q) and (img[i, j] >= r):
                    Z[i, j] = img[i, j]
                else:
                    Z[i, j] = 0
            except IndexError as e:
                pass
    return Z
```

Алгоритм 5: Функция подавления не-максимумов

Реализация опредления потенциальных границ:

```
def threshold(self, img):
    highThreshold = img.max() * self.highThreshold
    lowThreshold = highThreshold * self.lowThreshold
    M, N = img.shape
    res = np.zeros((M, N), dtype=np.int32)
    weak = np.int32(self.weak_pixel)
    strong = np.int32(self.strong_pixel)
    strong_i, strong_j = np.where(img >= highThreshold)
    zeros_i, zeros_j = np.where(img < lowThreshold)
    weak_i, weak_j = np.where((img <= highThreshold)
        & (img >= lowThreshold))
    res[strong_i, strong_j] = strong
    res[weak_i, weak_j] = weak
    return res
```

Алгоритм 6: Функция определения потенциальных

Финальным шагом является трассировка областей неоднозначности:

```
def hysteresis(self, img):
    M, N = img.shape
    weak = self.weak_pixel
    strong = self.strong_pixel
    for i in range(1, M - 1):
        for j in range(1, N - 1):
            if img[i, j] == weak:
                try:
                    if ((img[i + 1, j - 1] == strong)
                        or (img[i + 1, j] == strong)
                        or (img[i + 1, j + 1] == strong)
                        or (img[i, j - 1] == strong)
                        or (img[i, j + 1] == strong)
                        or (img[i - 1, j - 1] == strong)
                        or (img[i - 1, j] == strong)
                        or (img[i - 1, j + 1] == strong)):
                        img[i, j] = strong
                    else:
                        img[i, j] = 0
                except IndexError as e:
                    pass
    return img
```

Алгоритм 7: Трассировка области неоднозначности

Стоит обратить внимание, что описанные выше методы принимают на вход изображение в серых тонах. То есть нулевым шагом данных методов можно указать преобразование изображения из цветного в черно-белое.

3. Выделение силуэта кисти руки

Помимо классических методов определения границ можно использовать сегментациб по цвету кожи [5]. Данный метод преобразует RGB изображение в бинарное с помощью фильтрации пикселей по цвету, близкому к цвету кожи. Для улучшения работы алгоритма перед фильтрацией изображение переводят в цветовое пространство YCrCb, в котором различные цвета кожи расположены близко друг к другу [6].

Пример фильтрации изображения:

```
import math

for i in range(0, hand.shape[0]):
    for j in range(0, hand.shape[1]):
        if (math.sqrt(
                (int(imgYCC[i, j, 0]) - avg_skin_color[0]) ** 2 +
                (int(imgYCC[i, j, 1]) - avg_skin_color[1]) ** 2 +
                (int(imgYCC[i, j, 2]) - avg_skin_color[2]) ** 2) <= error):
            imgYCC[i, j] = [255, 255, 255]
        else:
            imgYCC[i, j] = [0, 0, 0]
```

Алгоритм 8: Фильтрация изображения по цвету кожи

В большинстве случаев после бинаризации на изображении присутствуют шумы и артефакты, вызванные тем, что на фоновой части изображения находились пиксели, попадающие в ограничения фильтра. Для их устранения можно использовать морфологические операции "расширение"и "сужение"[7]:

```
kernel = np.ones((7, 7), np.uint8)
imgYCC = cv2.dilate(imgYCC, kernel, iterations=1)
imgYCC = cv2.erode(imgYCC, kernel, iterations=1)
```

Алгоритм 9: Применение к изображению операций расширение и сужение

4. Построение скелета кисти руки

Для ускорения процесса классификации жеста руки можно использовать скелетную модель. Данный тип входных данных в силу своей специфики может упростить вычисление признаков, необходимых классификатору.

Для построения скелета кисти можно использовать метод построения скелета выпуклой фигуры [7].

В качестве выпуклой фигуры можно использовать результат работы метода, описанного в разделе 3.

В данном методе предлагается поиск скелета с помощью морфологической операции "сужение". Данная операция применяется до тех пор, пока последующие применение не приведет к очищению изображения. Пример реализации:

```
img = cv2.cvtColor(orig_img, cv2.COLOR_BGR2GRAY)
size = np.size(img)
skel = np.zeros(img.shape, np.uint8)

ret, img = cv2.threshold(img, 127, 255, 0)
element = cv2.getStructuringElement(cv2.MORPH_CROSS, (3, 3))
done = False

while not done:
    eroded = cv2.erode(img, element)
    temp = cv2.dilate(eroded, element)
    temp = cv2.subtract(img, temp)
    skel = cv2.bitwise_or(skel, temp)
    img = eroded.copy()
    zeros = size - cv2.countNonZero(img)
    if zeros == size:
        done = True
skel = cv2.cvtColor(skel, cv2.COLOR_GRAY2BGR)
for i in range(0, skel.shape[0]):
# looping through the rows( height)
    for j in range(0, skel.shape[1]):
# looping through the columns(width)
        if (skel[i, j] == [255, 255, 255]).all():
            skel[i, j] = [255, 0, 0]
            orig_img[i, j] = [255, 0, 0]
```

Алгоритм 10: Фильтрация изображения по цвету кожи

Проблемой данного метода являются побочные ветви скелета, образованные из-за возможной зашумленности или неточности фигуры. Другим недостатком можно считать отсутствие гарантии обеспечения связного набора пикселей для всего скелета, или обеспечения одинаковой ширины ветвей во всем скелете.

Для решения данных проблем можно обратиться к технологиям машинного обучения. Скелет кисти можно построить на основании ключевых точек, получаемых с помощью нейронной сети [8]. Данная нейронная сеть определяет на изображении 22 ключевых точки, 21 из которых относятся к кисти руки, а 22 отмечает фон. Пример расположения точек представлен на рисунке 1.

Далее для построения скелета необходимо соединить полученные точки в последовательностях, описанной в таблице 1.

В ходе экспериментов было выявлена проблема с нахождением ключевых точек. На некоторых изображениях алгоритм либо не находил точки вообще, либо находил не полное их количество.

5. Данные

Для проверки работы описанных методов были проведены тесты на нескольких наборах данных, различающиеся разными форматами изображений, их размерами и людьми, чьи ру-

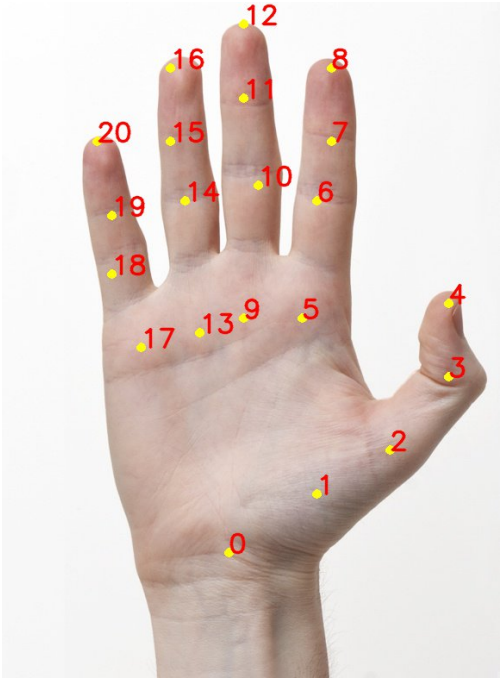

Рис. 1: Ключевые точки кисти руки

Таблица 1: Сравнение алгоритмов выделения источников

Ветвь скелета	Последовательность точек
Большой палец	$0 \to 1 \to 2 \to 3 \to 4$
Указательный палец	$0 \to 5 \to 6 \to 7 \to 8$
Средний палец	$0 \to 9 \to 10 \to 11 \to 12$
Безымянный палец	$0 \to 13 \to 14 \to 15 \to 16$
Мизинец	$0 \to 17 \to 18 \to 19 \to 20$

ки использовались для получения изображений кистей. Все данные находятся в открытом доступе:

- ASL Alphabet. Image data set for alphabets in the American Sign Language[1]. Данный датасет состоит из 87000 изображений американского дактиля в обучающей выборке и 29 проверочных изображений. Каждое изображение цветное, сохранено в формате JPG, имеет размерность 200x200 пикселей. Для проверки работы алгоритмов были использованы проверочные изображения.

- Hand Gesture of the Colombian sign language. Hand gestures, recognizing the numbers from 0 to 5 and the vowels[2]. Данный датасет состоит из фотографий жестов, изображающий

[1]https://www.kaggle.com/grassknoted/asl-alphabet
[2]https://www.kaggle.com/evernext10/hand-gesture-of-the-colombian-sign-language

гласные буквы колумбийского языка и цифры от 0 до 5. Приведены снимки как мужских, так и женских рук. Каждый жест имеет 3 разных фото с разных ракурсов. Каждое изображение цветное, сохранено в формате JPG, имеет размерность 4608 x 2592 пикселей. Для проверки работы алгоритмов были отобраны случайные изображения, по одному на каждый жест.

- ASL Fingerspelling Images (RGB & Depth) [3]. Данная выборка состоит из изображений американского дактиля. В выборке участвуют 5 разных людей, у каждого человека на каждый символ приходится более 1000 изображений. Все изображения цветные, формата PNG, имеют различную размерность. Для проверки работы алгоритмов были отобраны случайным образом по одному изображению для каждой буквы, то есть в итоговой выборке участвовали снимки разных людей под разными ракурсами.

- sign language between 0 9[4]. Данная выборка состоит из изображений жестов, обозначающих цифры от 0 до 10. Все изображения цветные, формата JPG, размерности 300x300. Данные разделены на обучающие, где на каждую цифру приходится более 100 различных изображений, и проверочные, где на каждую цифру представлено одно изображение. Для проверки работы алгоритмов были выбраны все изображения из проверочной выборки.

6. Результаты

Целью экспериментов было определение наиболее оптимального метода преобразования исходного изображения с целью упрощения процесса классификации. Для этого необходимо сравнить визуально результаты работы перечисленных выше алгоритмов и время их работы. Результаты экспериментов представлены ниже, отдельно для каждого набора данных:

- Результаты для ASL Alphabet представлены на рисунке А.2 и в таблице А.2
- Результаты для Hand Gesture of the Colombian sign language представлены на рисунке А.3 и в таблице А.3
- Результаты для ASL Fingerspelling Images представлены на рисунке А.4 и в таблице А.4
- Результаты для sign language between 0 9 представлены на рисунке А.5 и в таблице А.5

В результате экспериментов установлено, что среди методов выделения контура по качеству работы лидирует оператор Кэнни. Учитывая небольшую разницу во времени их работы, можно отбросить из рассмотрения все остальные операторы.

Простое выделение силуэта показало наилучшие временные результаты. Так же при правильной предварительной настройке метода можно добиться удовлетворительной четкости выделения. Тем не менее, предварительная настройка является главной проблемой этого алгоритма.

[3] https://www.kaggle.com/mrgeislinger/asl-rgb-depth-fingerspelling-spelling-it-out
[4] https://www.kaggle.com/beyzance96/sign-language-between-0-9

Морфологическое построение скелета показало плохой результат. Как говорилось выше, в результате получаются побочные ветви, а так же скелет получается неполносвязным. Данные недостатки не позволят сильно упросить работу классификатора в силу зашумленности итоговых данных.

Алгоритм построения скелета по ключевым точкам не справился со своей задачей на большинстве результатов. Так же для любого типа данных он работает за одно и тоже время. Это одновременно и хорошо (результаты Hand Gesture of the Colombian sign language) и плохо (остальные результаты).

7. Заключение

В результате данной работы были исследованы возможные методы предобработки изображения в задачах классификации жестовых символов. Были проведены эксперименты с целью определения наиболее оптимального по скорости работы и качеству выделения основных признаков метода.

В результате сравнительного анализа можно выделить два метода:

- Выделение силуэта
- Построение скелета по ключевым точкам

Первый метод показал наилучшие результаты по скорости работы алгоритма, кроме тестов на широкоформатных изображениях. В дальнейшем можно предложить улучшение путем упрощения первичной конфигурации цвета кожи и оптимизации работы на больших изображениях.

Второй метод, несмотря на неудачные результаты тестов, расходящимся с результатами авторов[8], можно попробовать оптимизировать по качеству через переобучение модели. По скорости данный алгоритм можно оптимизировать путем реализации его на другом языке программирования.

Список литературы

[1] Irwin Sobel. An isotropic 3x3 image gradient operator. *Presentation at Stanford A.I. Project 1968*, 02 2014.

[2] Judith M. S. Prewitt. Object enhancement and extraction picture processing and psychopictorics. *Academic*, pages 75–149, 1970.

[3] Lawrence Roberts. *Machine Perception of Three-Dimensional Solids.* 01 1963.

[4] John Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-8:679 – 698, 12 1986.

[5] S.L. Phung, Abdesselam Bouzerdoum, and Douglas Chai. Skin segmentation using color and edge information. pages 525 – 528 vol.1, 08 2003.

[6] Joshi Siddharth and Srivastava Gaurav. Face detection. *E368: Digital Image Processing*, pages 101 – 112, 2003.

[7] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing.* Pearson Prentice Hall, third edition, 2008.

[8] Tomas Simon, Hanbyul Joo, Iain Matthews, and Yaser Sheikh. Hand keypoint detection in single images using multiview bootstrapping. 04 2017.

ПриложениеA. Название приложения

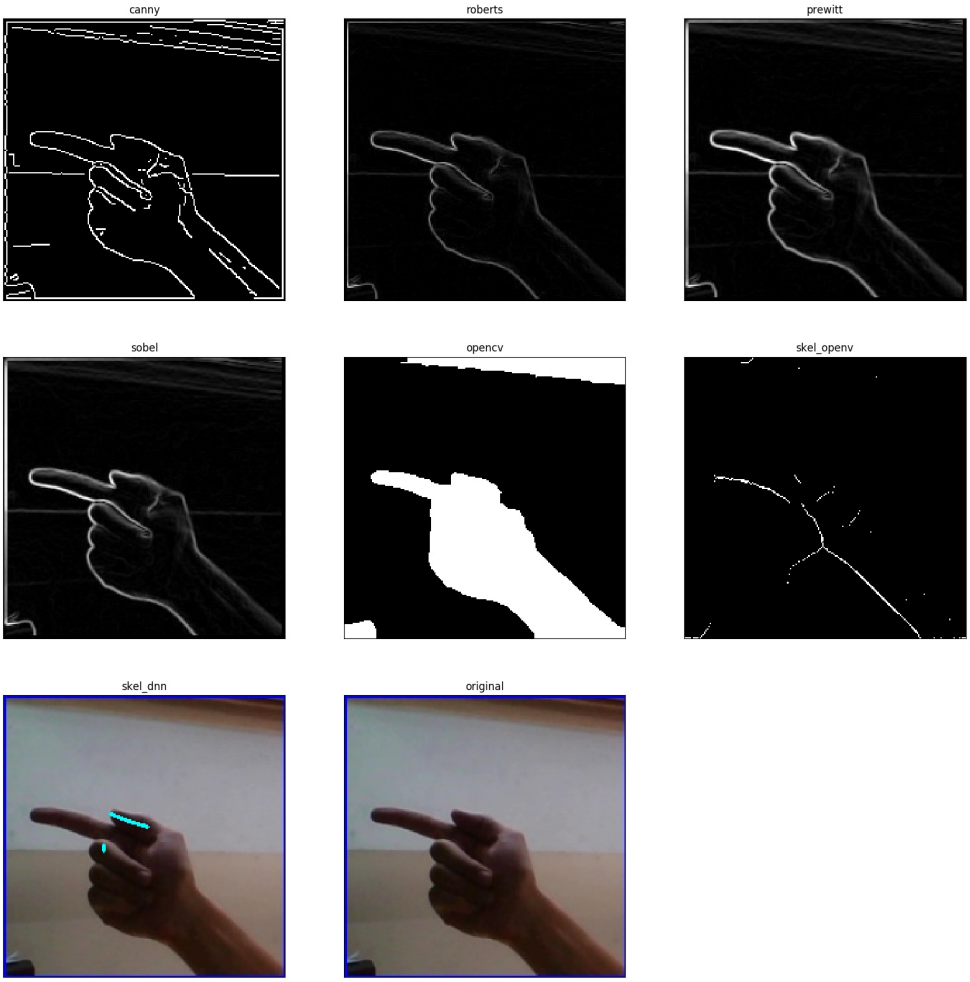

Рис. A.2: Результат работы алгоритмов (в секундах) на наборе данных ASL Alphabet

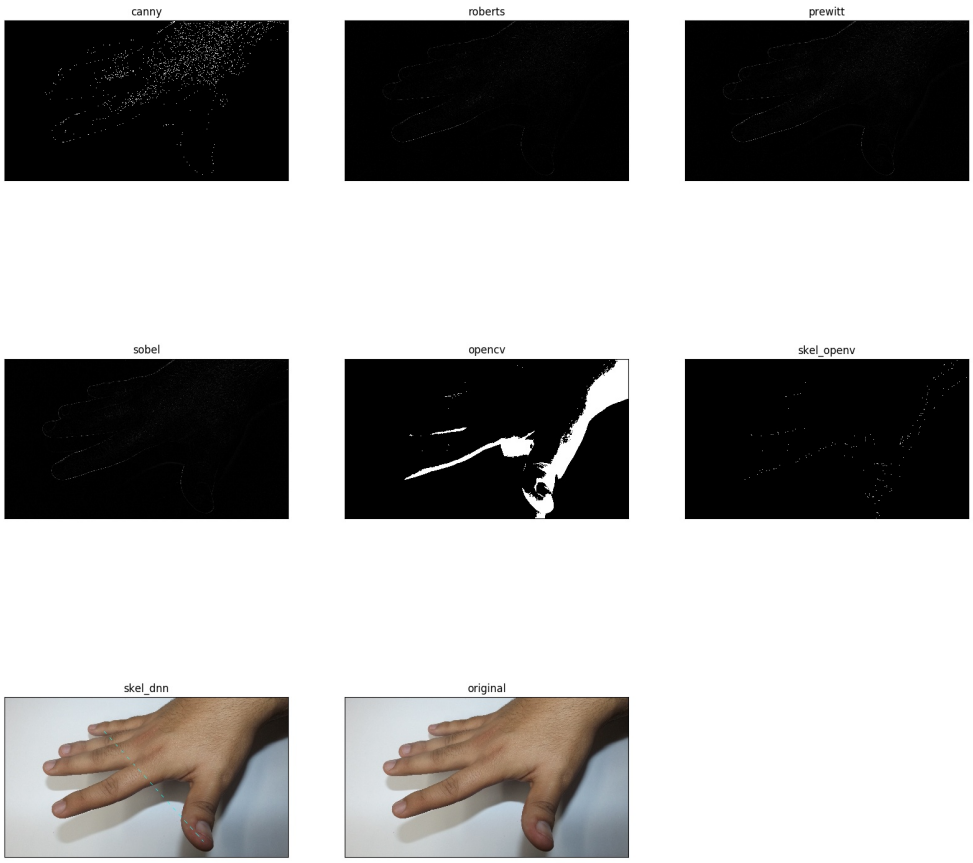

Рис. А.3: Результат работы алгоритмов (в секундах) на наборе данных Hand Gesture of the Colombian sign language

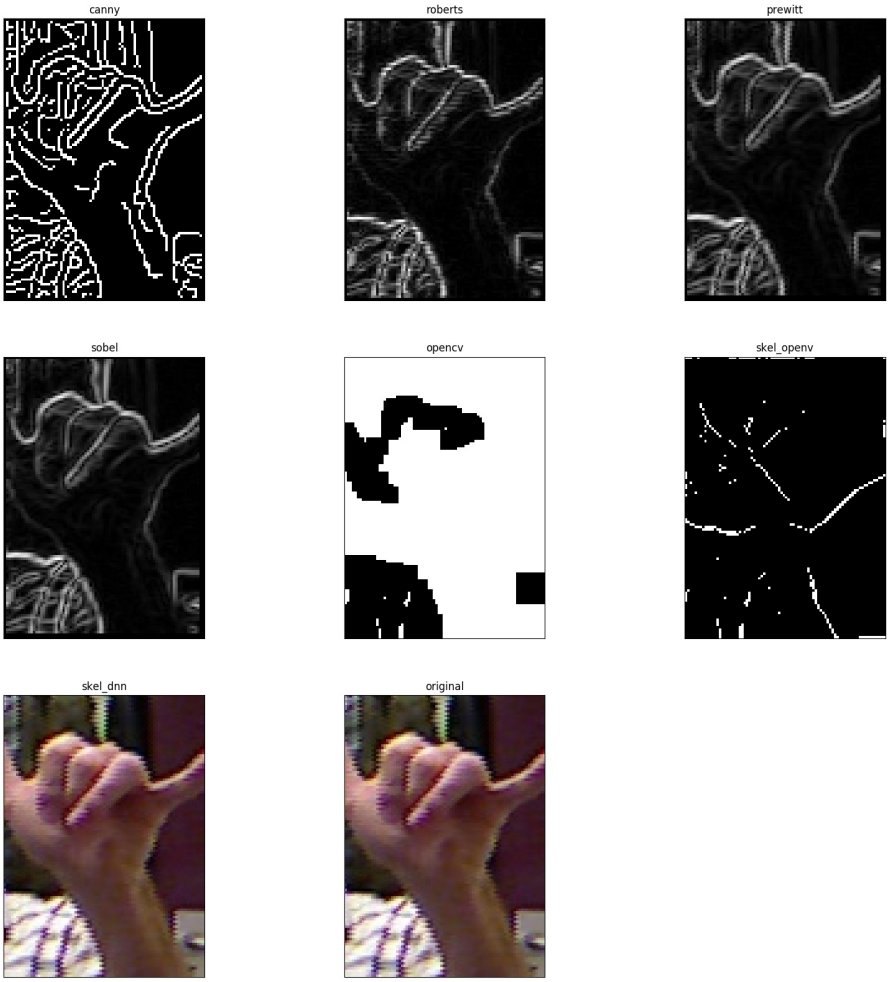

Рис. А.4: Результат работы алгоритмов (в секундах) на наборе данных ASL Fingerspelling Images

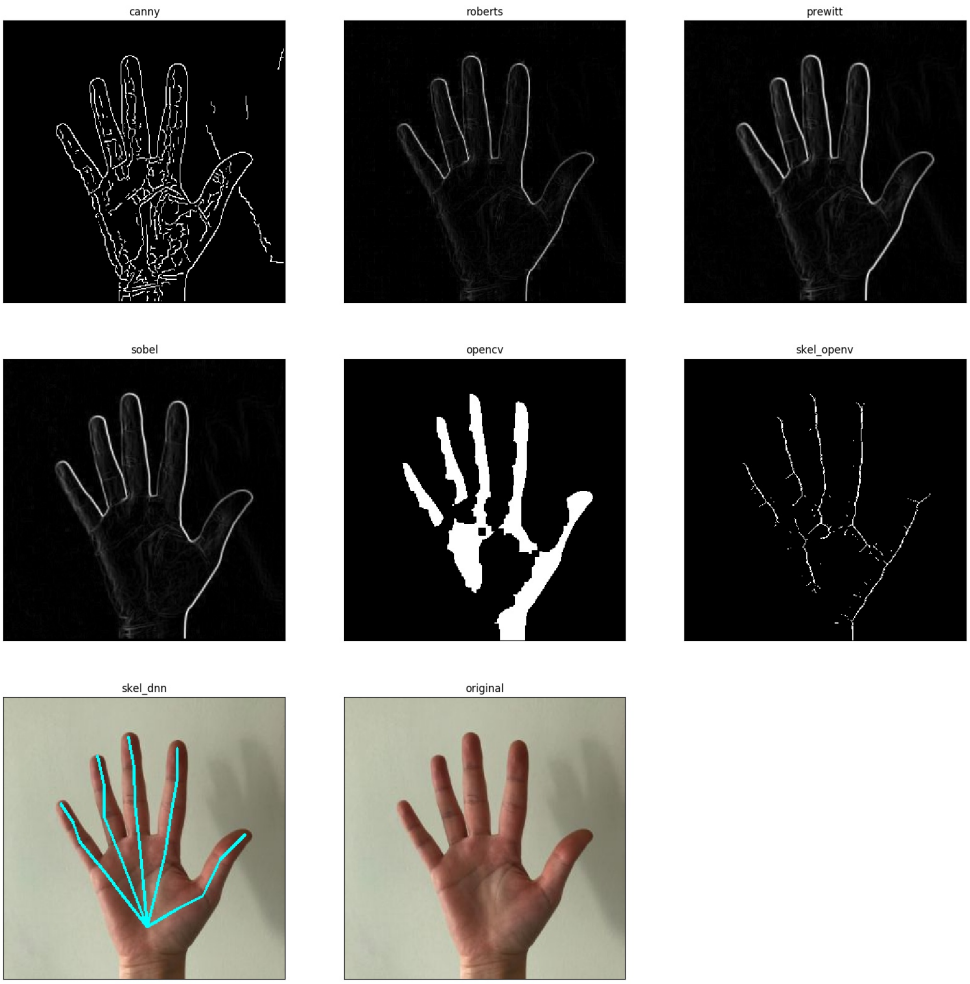

Рис. А.5: Результат работы алгоритмов (в секундах) на наборе данных sign language between 0 9

Таблица А.2: Время работы алгоритмов (в секундах) на наборе данных ASL Alphabet

Название алгоритма	Минимальное время	Максимальное время	Среднее время
Оператор Кэнни	0.1783	0.4642	0.2553
Оператор Робертса	0.0687	0.1252	0.0852
Оператор Прюитт	0.1175	0.2211	0.1424
Оператор Собеля	0.1177	0.3112	0.1527
Выделение силуэта	0.0668	0.2101	0.0901
Морфологическое построение скелета	0.2084	1.2112	0.3068
Построение скелета по ключевым точкам	1.408	3.4571	2.042

Таблица А.3: Время работы алгоритмов (в секундах) на наборе данных Hand Gesture of the Colombian sign language

Название алгоритма	Минимальное время	Максимальное время	Среднее время
Оператор Кэнни	53.5126	72.6076	62.971
Оператор Робертса	22.2837	54.2706	25.8842
Оператор Прюитт	38.7491	99.2961	46.5944
Оператор Собеля	38.8739	104.1867	46.9016
Выделение силуэта	22.3001	32.8930	23.5992
Морфологическое построение скелета	65.3335	88.3565	69.0014
Построение скелета по ключевым точкам	3.0844	4.2156	3.2775

Таблица А.4: Время работы алгоритмов (в секундах) на наборе данных ASL Fingerspelling Images

Название алгоритма	Минимальное время	Максимальное время	Среднее время
Оператор Кэнни	0.0193	0.0816	0.0545
Оператор Робертса	0.0111	0.0476	0.0286
Оператор Прюитт	0.0179	0.0864	0.0495
Оператор Собеля	0.0217	0.0847	0.0469
Выделение силуэта	0.0114	0.0506	0.0277
Морфологическое построение скелета	0.0343	0.1852	0.0884
Построение скелета по ключевым точкам	0.6251	3.3092	1.5665

Таблица A.5: Время работы алгоритмов (в секундах) на наборе данных sign language between 0 9

Название алгоритма	Минимальное время	Максимальное время	Среднее время
Оператор Кэнни	0.333	0.7686	0.4708
Оператор Робертса	0.1566	0.246	0.1778
Оператор Прюитт	0.271	0.3751	0.3027
Оператор Собеля	0.2718	0.655	0.3295
Выделение силуэта	0.1486	0.4709	0.2085
Морфологическое построение скелета	0.4471	1.637	0.6518
Построение скелета по ключевым точкам	1.4346	3.2976	2.0723