Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

НА ТЕМУ:

«Метод распознавания жестовых символов»

Студент __ИУ7-42М_____ (Группа)	_____ (Подпись, дата)	**_Г.М. Танцевов_____** (И.О.Фамилия)
Руководитель ВКР	_____ (Подпись, дата)	**_К.А. Майков_____** (И.О.Фамилия)
Консультант	_____ (Подпись, дата)	_____ (И.О.Фамилия)
Консультант	_____ (Подпись, дата)	_____ (И.О.Фамилия)
Нормоконтролер	_____ (Подпись, дата)	**_Ю.В. Строганов____** (И.О.Фамилия)

2020 г.

тз

ТЗ

план

РЕФЕРАТ

Дипломная работа: 64 страницы, 24 рисунка, 8 таблиц, 30 источников.

В магистерской диссертации рассмотрена задача распознавания статических жестов. Проведен анализ предметной области, проанализированы известные методы решения данной задачи и обоснована необходимость построения метода, объединяющего преимущества существующих подходов. Разработан метод распознавания, основанны на использовании искусственных нейронных сетей, модифицированный путем добавления предобработки изображений с целью упрощения процесса классификации и использования в качестве классификатора капсульный нейронных сетей. Разработаны алгоритмические структуры для реализации комбинированного метода. Осуществлена программная реализация предложенного решения. Проведены вычислительные исследования, подтверждающие работоспособность метода и показывающие точность распознавания свыше 95%.

Результаты исследовательской деятельности были отражены в печатной работе.

В первом разделе приводится сравнительный анализ известных методов распознавания жестовых символов, на основании которого выбран прототип, для которого будут проводится исследования. Обоснована необходимость построение нового метода. Во втором разделе описан метод распознавания жестовых символов, описаны структуры, необходимые для его реализации. В третьем разделе описана архитектура программной реализации предложенного метода, обоснован выбор средств программной реализации. Описаны ресурсы, необходимые для сборки и запуска разработанного программного обеспечения, форматы входных и выходных файлов, а также интерфейс и руководство пользователя. В четвертом разделе описаны и проведены вычислительные экспери менты для исследования предложенного метода.

СОДЕРЖАНИЕ

ГЛОССАРИЙ

В настоящей работе используются следующие термины с соответствующими определениями.

Жест — осознанное движение человеческого тела или его части, которое несет информацию и совершается с целью её передачи.

Капсула — группа искусственных нейровон, инкапсулирующая в себе информацию в векторном виде.

Скелет — множество точек, равноудаленных от границы области.

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

СММ — Скрытая марковская модель

ASL — American Sign Language

СКК — Самоорганизующаяся карта Кохонена

СНС — Сверточные нейронные сети

КНС — Капсульные нейронные сети

ВВЕДЕНИЕ

Одной из актуальных задач в области компьютерного зрения является распознавание жестов. Качественный метод распознавания жестов позволит дать развитие многим системам, таким как интеллектуальные жестовые интерфейсы, системы перевода с жестовых языков, управление для систем виртуальной и дополненной реальностей. В настоящее время известен ряд практически применимых решений данной задачи [1], но все они имеют недостатки, например необходимость использования дополнительных источников данных (перчатки с датчиками).

Целью данной работы является разработка метода распознавания жестовых символов. Для решения поставленной задачи необходимо:

— проанализировать предметную область;

— проанализировать существующие методы распознавания жестов;

— на основе полученных во время анализа данных разработать собственный метод распознавания жестовых символов;

— реализовать разработанный метод в программном продукте;

— провести планирование и постановку экспериментов с целью выяснения качества работы разработанного метода.

1 Аналитический раздел

Известные на данный момент методы распознавания жестовых символов, как правило, реализуют три этапа обработки информации:

а) получение данных о жесте;
б) предобработка данных;
в) классификация жестов.

В качестве входных данных можно использовать кинематические трехмерные модели рук, получаемых с помощью специальных устройств ввода, таких как Microsoft Kinect[2] и Leap Motion Controller[3]. Такие методы обеспечивают достаточно высокую точность распознавания, позволяя обнаруживать различные жесты, но при этом требуют большого объема вычислений и наличия обширной базы данных, содержащей все жесты. Также данные методы получения информации о жесте требуют наличия дополнительных устройств ввода. Из-за этого, данную группу способов получения информации о жестах решено не рассматривать в данной работе.

В качестве другого источника данных можно использовать RGB изображения, полученные, например, с web-камеры ПК или камеры смартфона. Преимуществом данного подхода является распространенность данных устройств ввода. Web-камерой в наше время оснащен каждый ноутбук, а смартфонами владеет 45% населения Земли [4].

1.1 Алгоритмы предобработки изображения кисти руки, применимых к распознаванию жестовых символов

Скорость и качество работы алгоритмов классификации во многом зависит от исходных данных. Например, для классификации жестов с помощью скрытой марковской модели[5] основные признаки получаются из изображения рук в разноцветных перчатках. Тем самым, важно подобрать метод предобработки изображения таким образом, чтобы его применение в итоговом методе упрощало процесс классификации, не увеличивая при этом общее время работы. Известные алгоритмы, применимые для достижения данной цели, можно разделить на следующие группы:

— выделение контура фигуры;
— выделение силуэта кисти руки;
— построение скелета кисти руки.

Рассмотрим каждую из этих групп.

1.1.1 Выделение контура фигуры

Для выделения контура кисти руки можно использовать детекторы границ, определяющие градиент яркости черно-белого изображения. Поэтому предварительным этапом данных методов является преобразование изображения из цветного в черно-белое. К вышеуказанным методам относят:

— оператор Собеля[6];
— оператор Прюитт[7];
— перекрестный оператор Робертса[8];
— оператор Кэнни[9].

Операторы Собеля, Прюитт и Робертса

Принцип работы данных алгоритмов [6, 7, 8] заключается в свертке изображения двумя сепарабельными целочисленными фильтрами. Общая схема работы методов представлена на рис. 1.1.

Разница алгоритмов заключается в способе задания ядер свертки:

Оператор Собеля:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{1.1}$$

Оператор Прюитт:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} G_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \tag{1.2}$$

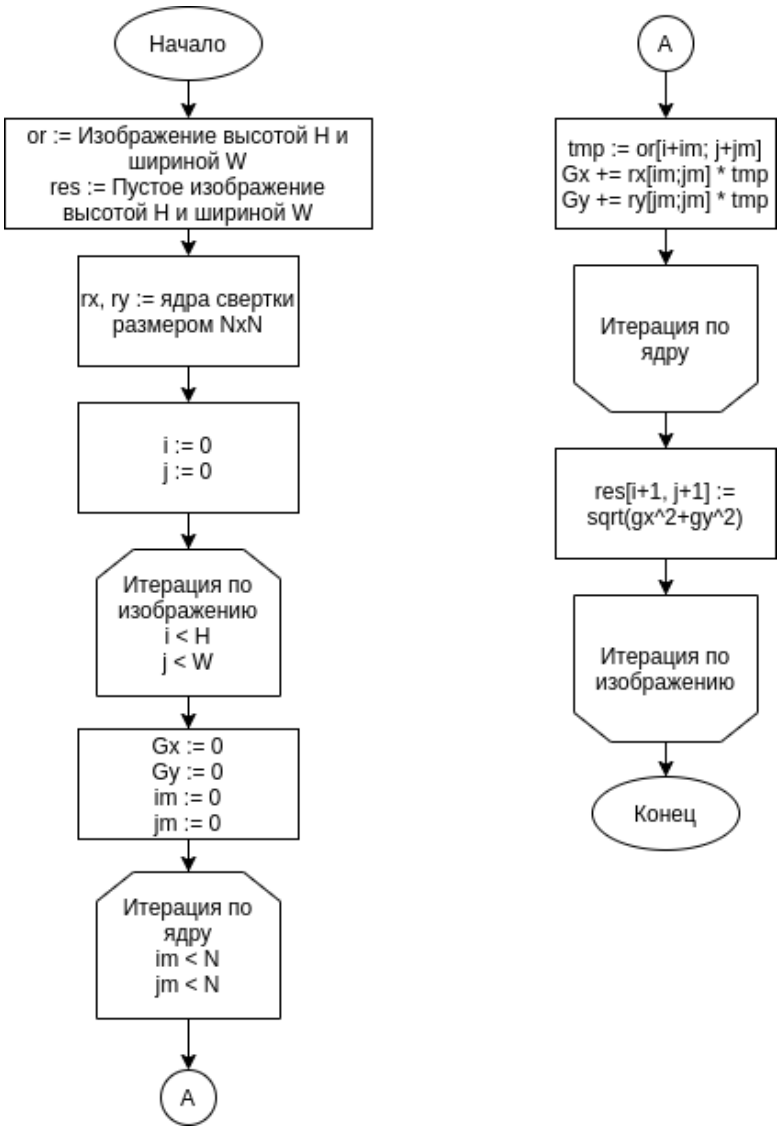

Рисунок 1.1 — Общая схема алгоритмов определения границ изображения

Из-за меньшего значения средних элементов итоговое изображение имеет более явный эффект сглаживания.

Перекрестный оператор Робертса:

$$G_x = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} G_y = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \tag{1.3}$$

Недостатком данного метода является отсутствие четко выраженного центрального элемента у ядра свертки. Но эта особенность алгоритма обуславливает высокую скорость обработки изображения.

В итоговом изображении в каждый пиксель записывается значение изменения яркости пикселя исходного изображения относительно соседних,

вычисляемое по формуле $G = \sqrt{G_x^2 + G_y^2}$, т. е. чем выше итоговое число, тем вероятнее, что данный пиксель находится на границе.

Оператор Кэнни

Метод[9] был разработан с целью удовлетворения следующим условиям:

— хорошее обнаружение (Кэнни трактовал это свойство как повышение отношения сигнал/шум);

— хорошая локализация (правильное определение положения границы);

— единственный отклик на одну границу.

Схема работы алгоритма представлена на рис. 1.2. Рассмотрим каждый этап подробнее с наглядной визуализацией обработки. Для этого применим данный оператор шаг за шагом к изображению, приведенному на рис. 1.3а.

Рисунок 1.2 — Схема алгоритма оператора Кэнни

а) Размытие изображения. Данный этап, как видно на рис. 1.3б, необходим для устранения лишних шумов, способных понизить качество последующих этапов выделения границ.

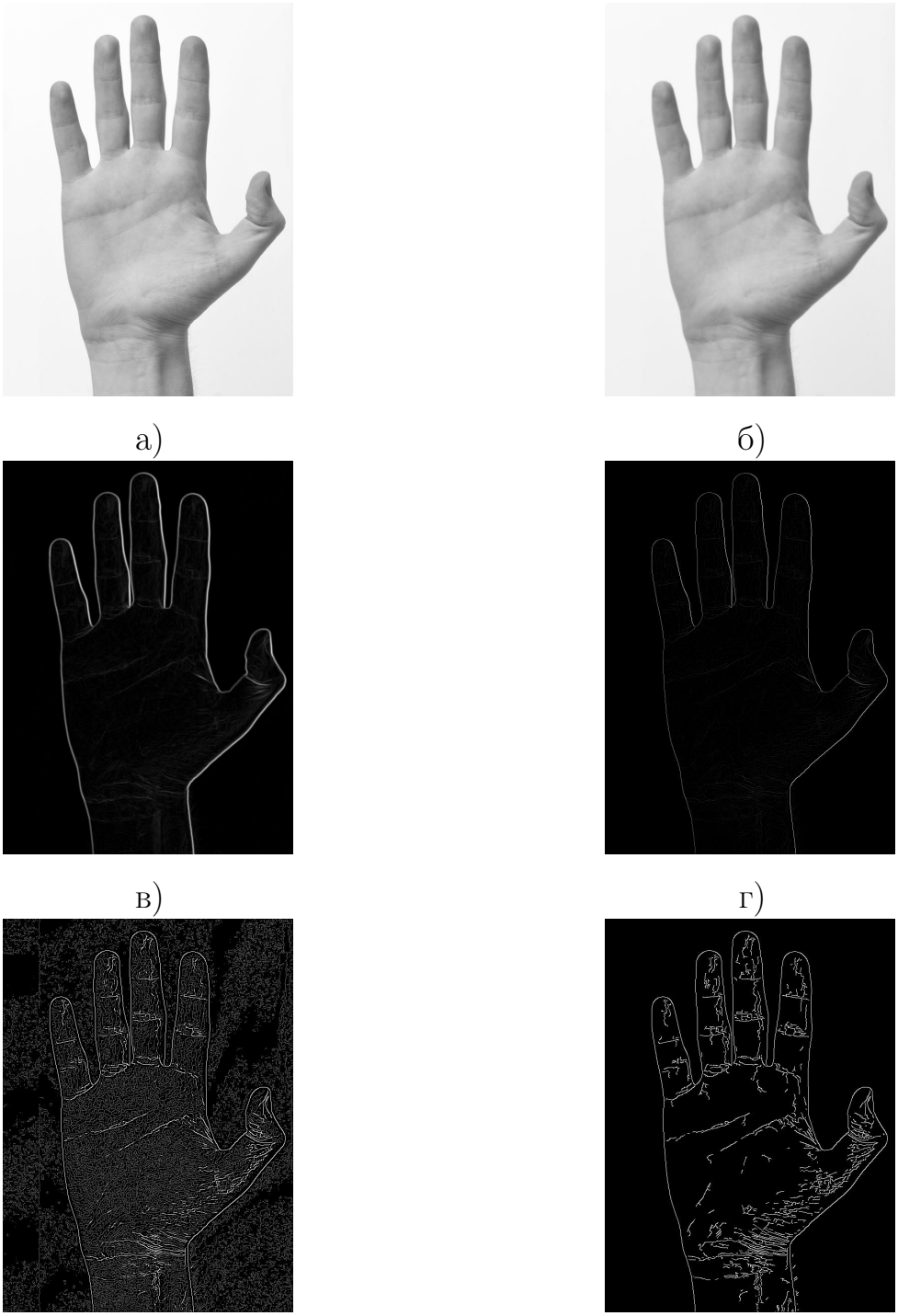

Рисунок 1.3 — Результаты экспериментального исследования этапов оператора Кэнни: а) исходное изображение; б) применение размытия; в) поиск градиентов; г) подавление не-максимумов; д) двойная пороговая фильтрация; е) трассировка областей неоднозначности

б) Поиск градиентов яркости. На данном этапе был применен оператор Собеля, описанный ранее. В результате на рис. 1.3в были получены точки, наиболее вероятно находящиеся на границах изображения[1].

в) Подавление «не-максимумов». Как видно на рис. 1.3г, на данном этапе отбрасываются точки, значение градиента которых не является локальным максимумом, т. е. такие точки являются ложными границами.

г) Определение потенциальных границ с помощью двойной пороговой фильтрации. При этом используется два порога фильтрации:

— Все пиксели со значением больше верхней границы принимают максимальное значение (достоверная граница).

— Все пиксели со значением меньше нижней границы подавляются.

— Все пиксели со значением в диапазоне границ принимают фиксированное среднее значение. Их уточнение происходит на следующем этапе.

На рис. 1.3д представлен результат применения данной фильтрации с порогами 0.03 и 0.07. В результирующую достоверную границу была добавлена часть контура тени кисти.

д) Трассировка области неоднозначности. После данного этапа, как видно на рис. 1.3е были отброшены все неопределенные границы, потому что они не были связаны с уже определенной границей.

1.1.2 Выделение силуэта кисти руки

Помимо классических методов определения границ, можно использовать сегментацию по цвету кожи[10]. 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[11]. Пример работы данного алгоримта представлен на рисунке 1.4.

Как правило, после бинаризации на изображении присутствуют шумы и артефакты, вызванные тем, что на фоновой части изображения находились

а)

б)

Рисунок 1.4 — Результат выделения кисти руки: а) исходное изображение; б) предобработанное изображение

пиксели, попадающие в ограничения фильтра. Для их устранения можно использовать морфологические операции: «наращивание» и «эрозия»[12].

Пусть имеется бинарное изображение A и структурный элемент B с началом координат в его центре (рис. 1.5).

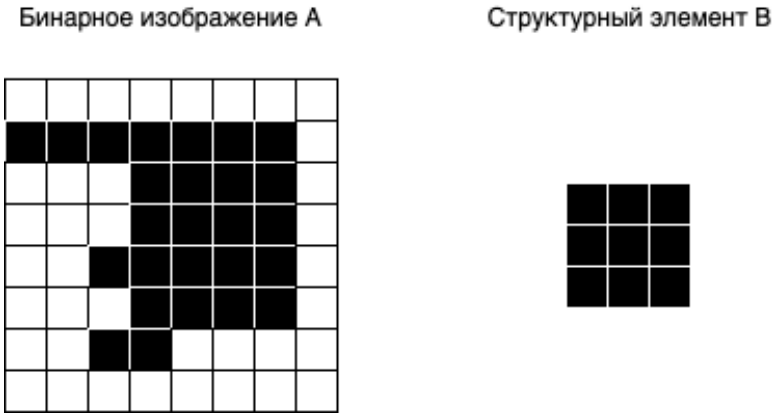

Рисунок 1.5 — Бинарное изображение и структурный элемент

Наращивание. Каждый раз, когда начало координат структурного элемента совмещается с единичным бинарным пикселем, ко всему структурному элементу применяется перенос и последующее логическое сложение с соответствующими пикселями бинарного изображения (рис. 1.6).

Эрозия. Если в некоторой позиции каждый единичный пиксель структурного элемента совпадает с единичным пикселем бинарного изображения, то выполняется логическое сложение центрального пикселя структурного элемента с соответствующим пикселем выходного изображения (рис. 1.7).

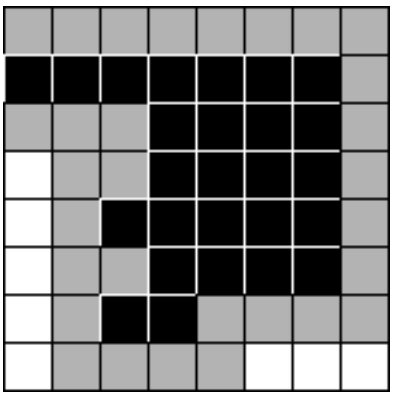

Рисунок 1.6 — Наращивание бинарного изображения A структурным элементом B

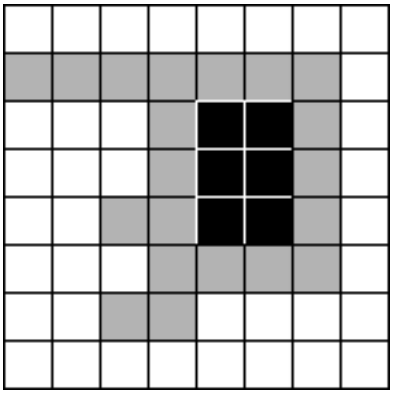

Рисунок 1.7 — Эрозия бинарного изображения A структурным элементом B

1.1.3 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[12].

В качестве выпуклой фигуры можно использовать результат работы метода выделения силуэта кисти руки, описанного в разделе 1.1.2.

В данном методе предлагается последовательное применение морфологической операции «эрозия» (рис. 1.7) до тех пор, пока результатом следующей итерации не станет пустое изображение.

Недостатком данного метода являются побочные ветви скелета, образованные из-за возможной зашумленности или неточности фигуры.

Другим недостатком можно считать высокую вероятность получения несвязного 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[13]. Данная нейронная сеть определяет на изображении 22 ключевые точки, 21 из которых относится к кисти руки, а 22 — отмечают фон. Пример расположения точек представлен на рис. 1.8.

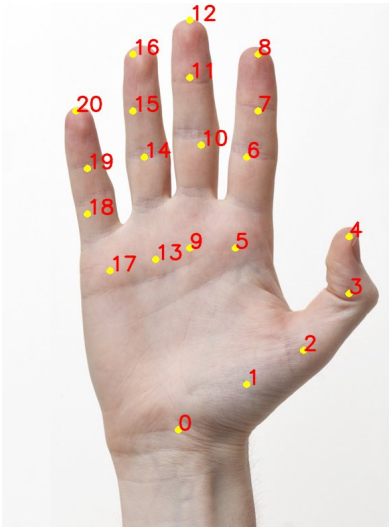

Рисунок 1.8 — Ключевые точки кисти руки

Далее для построения скелета необходимо соединить полученные точки в последовательностях, описанной в табл. 1.1.

Таблица 1.1 — Последовательность соединения ключевых точек

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

1.1.4 Сравнение алгоритмов предобработки изображений

Сравнительный анализ описанных выше методов был проведен в работе [14]. Каждый из алгоритмов был применен для обработки одинакового набора данных, состоящего из растровых изображений кистей рук; тестовая выборка составлялась из наборов данных, различающихся форматами изображений, их размерами и физиологическими особенностями кистей рук. Все данные находятся в открытом доступе:

— ASL Alphabet. Image data set for alphabets in the American Sign Language;

— Hand Gesture of the Colombian sign language. Hand gestures, recognizing the numbers from 0 to 5 and the vowels;

— ASL Fingerspelling Images (RGB & Depth);

— «sign language between 0 9».

Замеры времени обработки каждого изображения для получения статистики проводились по минимальному, максимальному и среднему времени работы алгоритма. Результаты экспериментов представлены для каждого набора данных: для ASL Alphabet представлены в табл. 1.2; для Hand Gesture of the Colombian sign language – в табл. 1.3; для ASL Fingerspelling Images — в табл. 1.4; для «sign language between 0 9» — в табл. 1.5.

Метод выделения силуэта показал наилучшие временные результаты (табл. 1.2). Также при правильной предварительной настройке метода можно добиться удовлетворительной четкости выделения. Тем не менее предварительная настройка является главной проблемой этого алгоритма. На рис. 1.9 видно, что при неудачном выборе начальных настроек алгоритм не справляется со своей задачей.

Морфологическое построение скелета показало время работы, сопоставимое с результатами оператора Кэнни (см. табл. 1.3, 1.4, 1.5). Итоговые результаты обработки изображения данным алгоритмом показали неудовлетворительное качество. Как говорилось выше, в результате получаются побочные ветви, а также скелет получается неполносвязным. В силу данных недостатков практически нецелесообразно использование данного алгоритма в построении метода классификации жестовых символов в силу зашумленности итоговых данных.

Таблица 1.2 — 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

Алгоритм построения скелета по ключевым точкам не справился со своей задачей на большинстве изображениях, как видно на рис. 8. Однако, как показано на рис. 1.10, в случае успешного определения ключевых точек алгоритм безупречно производит построение скелетной модели жеста. Также для любого типа данных он работает за одно и то же время. В одних случаях это является преимуществом (табл. 1.3), в других — недостатком (табл. 1.4).

1.1.5 Вывод

В результате проведенного сравнительного анализа были определены два метода.

Выделение силуэта. Данный метод показал наименьшее время работы. Кроме того, бинарное изображение руки содержит в себе необходимые признаки жеста для его обработки классификатором.

Построение скелета по ключевым точкам. Скелетная модель является наилучшим типом входных данных для классификатора, т. к. не несет в себе никаких лишних данных[15].

Таблица 1.3 — 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 1.4 — 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

Первый метод показал наилучшие результаты по скорости работы алгоритма, кроме тестов на широкоформатных изображениях. В неудачных

Таблица 1.5 — 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

случаях (рис. 1.9) второй метод не смог построить скелетную модель из-за неопределенных ключевых точек жеста.

1.2 Методы классификации

На данный момент известны следующие методы классификации жестовых символов:

— скрытая марковская модель[5, 16];
— самоорганизующаяся карта Кохонена[17, 18, 19];
— сверточные нейронные сети[20, 21].

1.2.1 Скрытая марковская модель

Одним из методов классификации, широкораспространенный в области распознавания жестов, является скрытая марковская модель (СММ). На ее основе построены системы распознавания китайского[5], английского[19] и польского[16] жестовых языков.

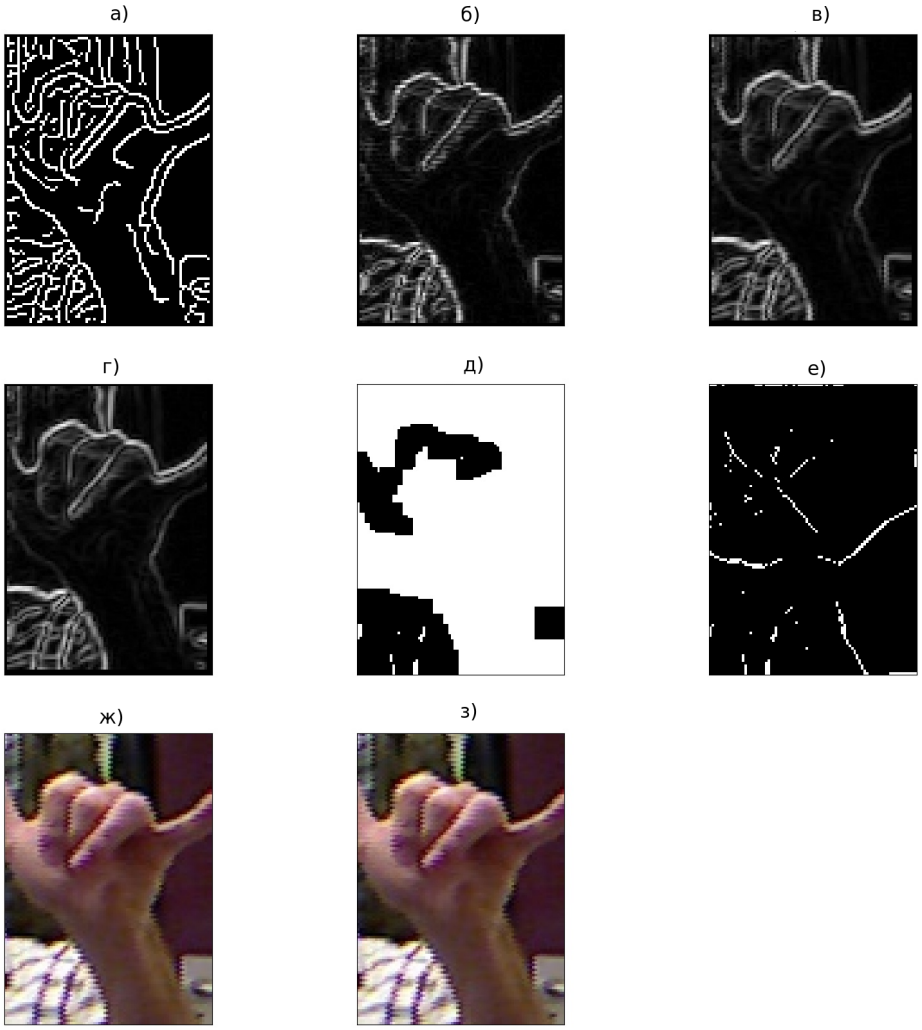

Рисунок 1.9 — Результат работы алгоритмов на наборе данных ASL Fingerspelling Images: а) оператор Кэнни; б) оператор Робертса; в) оператор Прюитт; г) оператор Собеля; д) выделение силуэта; е) морфологическое построение скелета; ж) построение скелета по ключевым точкам; з) оригинальное изображение

Скрытая марковская модель – модель процесса, считающегося Марковским. Система представляет собой марковскую цепь, которая имеет конечное множество скрытых состояний, т.е. заданный момент времени неизвестно, в каком состоянии s_i находится система. Каждое состояние s_i

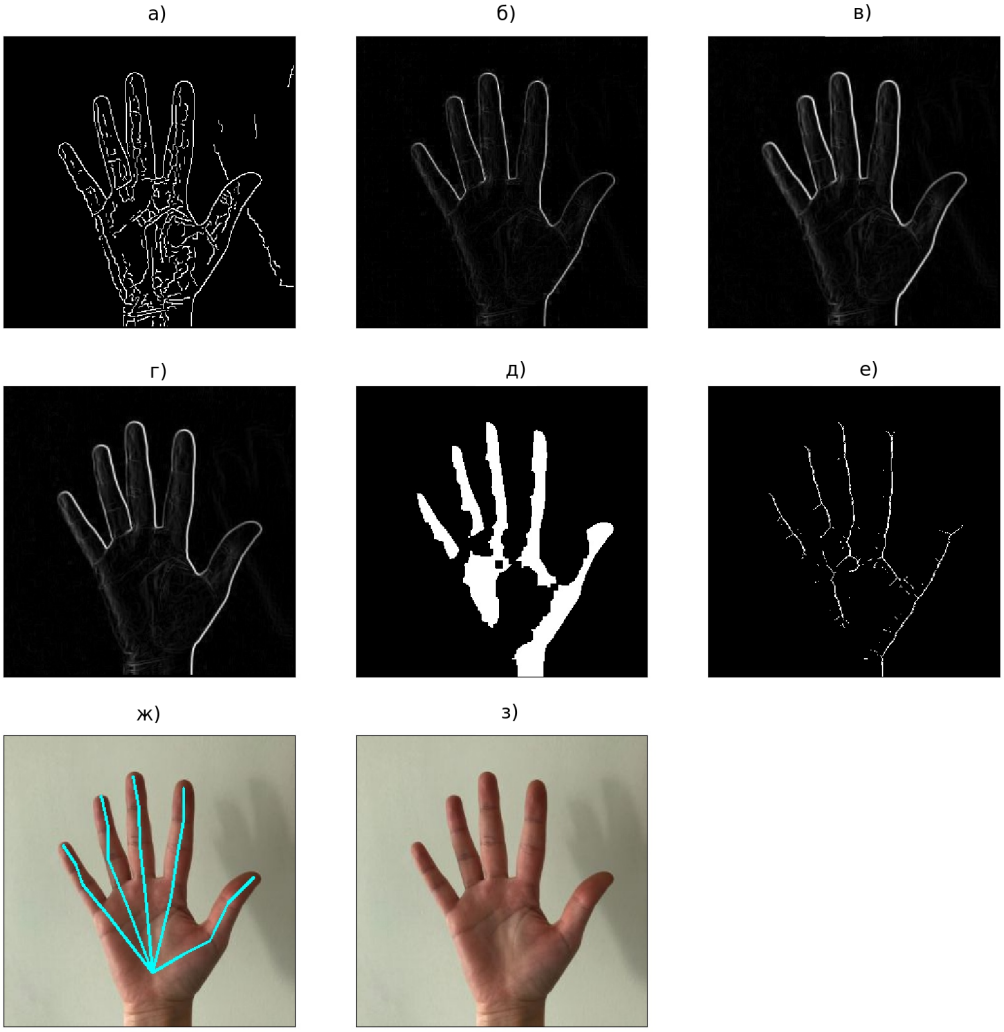

Рисунок 1.10 — Результат работы алгоритмов на наборе данных «sign language between 0 9»: а) оператор Кэнни; б) оператор Робертса; в) оператор Прюитт; г) оператор Собеля; д) выделение силуэта; е) морфологическое построение скелета; ж) построение скелета по ключевым точкам; з) оригинальное изображение

может с некоторой вероятностью b_{io_j} произвести событие o_j, которое можно наблюдать.

СММ λ задается как $\lambda = \{S, \Omega, \Pi, A, B\}$, где $S = \{s_1, ..., s_n\}$ – множество состояний, $\Omega = \{\omega_1, ..., \omega_m\}$ – множество возможных событий,

$\Pi = \{\pi_1, ..., \pi_n\}$ – множество начальных вероятностей, $A = \{a_{ij}\}$ – матрица вероятностей перехода из состояния s_i в состояние s_j, $B = \{b_{i\omega_k}\}$ – множество вероятностей наблюдения события ω_k после перехода системы в состояние s_i.

Задачи, решаемые с помощью СММ можно разделить на следующие группы:

— кластерный анализ[22] – упорядочивание объектов в сравнительно однородные группы;

— регрессионный анализ[23] – статистический метод исследования влияния одной или нескольких независимых переменных $x_1, ..., x_n$ на зависимую переменную y;

— задача классификации[24] – разделение множества объектов некоторым образом на классы.

Задачу классификации можно трактовать как поиск вероятности попадания в состояние s_n на шаге t. Для этого применяется алгоритм прямого-обратного хода. Обучение модели, происходящее с помощью алгоритма Витерби, заключается в подборе последовательности состояний, при которой вероятность заданной последовательности наблюдений является наибольшей. Алгоритм Баума-Велша меняет коэффициенты матрицы вероятности, максимизируя вероятность наблюдения последовательности событий O.

1.2.2 Самоорганизующаяся карта Кохонена

Самоорганизующиеся карты Кохонена(СКК) являются одной из разновидностей искусственных нейронных сетей. В области распознавания жестов нейросети данного типа могут применяться как для предобработки входных данных с целью извлечения признаков для классификатора[25], так и для распознавания самих жестов, например индийского[17] и американского[18] жестовых языков.

Основным отличием СКК от большинства известных нейронных сетей является обучение без учителя. При данном подходе процесс обучения не требует вмешательства со стороны и по этому результат будет зависеть только от структуры входных данных. Функцией СКК является кластеризация, то

есть нет необходимости заранее знать классы выходных данных из обучающей выборки.

Архитектура сети состоит из двух слоев: входного и выходного, при этом каждый нейрон входного слоя связан с каждым нейроном выходного. Нейроны выходного слоя упорядочены и имеют структуру сетки. При этом каждый нейрон представляет собой n-мертный вектор вида $w = [w_1, ..., w_n]^T$, где n равен размерности исходного пространства.

Процесс обучения разделяют на четыре основных этапа:

а) Инициализация. Первоначальные веса узлов задаются случайными числами.

б) Конкуренция. Нейроны вычисляют значения своей функции активации для каждого входного паттерна. Нейрон с наименьшим значением объявляется победителем.

в) Объединение. Активный нейрон определяет пространственное расположение топологической окрестности нейронов, которые будут участвовать в процессе обучения. Размер окрестности определяется радиусом обучения.

г) Подстройка весов. Выбранные нейроны уменьшают значения своих функций активации путем регулировки соответствующих весов узлов.

Модификация весовых коэффициентов происходит по формуле 1.4.

$$w_i(t+1) = w_i(t) + h(t) * [x(t) - w(t)], \tag{1.4}$$

где

— t – номер эпохи;
— $x(t)$ – некоторый вектор из обучающей выборки;
— $h(t)$ – функция соседства нейронов.

1.2.3 Сверточные нейронные сети

Создание сверточных нейронных сетей(СНС) было вдохновлено зрительной корой головного мозга человека[26]. Основное использование – обработка изображений. Отличительной особенностью данной сети,

подарившей ей название, является первый скрытый слой, работа которого похожа на процесс свертки двумерного изображения. В связи с этим, для большей наглядности входной слой вместо одномерного слоя нейронов можно рассматривать как двумерную матрицу, как в случае с файлом изображения. Для изображений значения этой двумерной матрицы представляют интенсивности пикселей.

Рассмотрим особенности данной сети: сверточный и субдискретизирующий слои.

Слой свертки

В отличии от обычной нейронной сети, в которой каждый нейрон входного слоя связан с каждым нейроном первого скрытого слоя, в СНС каждый нейрон в скрытом слое, называемом сверточным, связан только с нейронами, находящимися в определенной небольшой области (рис. 1.11), которая определяется ядром свертки. Для формирования скрытого слоя ядро перемещается построчно по всей входной области, и может происходить с различным шагом, например на рисунке 1.11 шаг смещения равен 1.

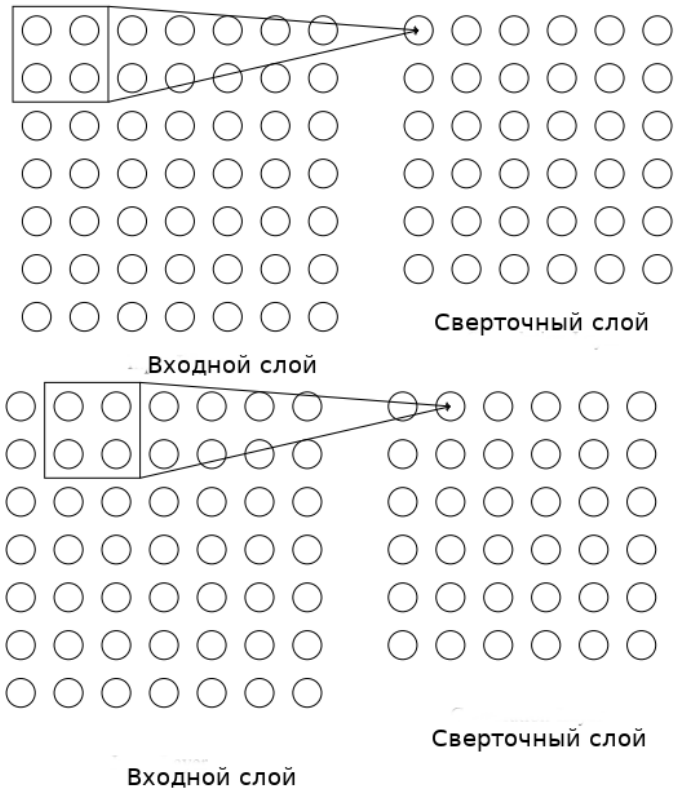

Рисунок 1.11 — Связывание входного слоя с первым скрытым слоем

Размерность сверточного слоя определяется формулой 1.5.

$$(((W - X)/s) + 1) \times ((H - Y)/s) + 1), \tag{1.5}$$

где

— $W \times H$ – размерность входного слоя;
— $X \times Y$ – размерность ядра свертки;
— s – шаг смещения.

Результат нейрона j,k сверточного слоя описывается формулой 1.6.

$$a_{j,k} = \sigma\left(b + \sum_{l=0}^{A-1}\sum_{m=0}^{B-1} w_{i,m}x_{j+l,k+m}\right) \tag{1.6}$$

Другими словами, сверточный слой выполняет функцию поиска первичных признаков входных данных, например границ изображения.

Слой субдискретизации

Слой субдискретизации (англ. Pooling) выполняет задачу уменьшения размерности данных через нелинейное уплотнение. Исходная область разбивается на области, к которым, независимо друг от друга, происходит уплотнение области до одного значения. Пример работы данного слоя предоставлен на рисунке 1.12.

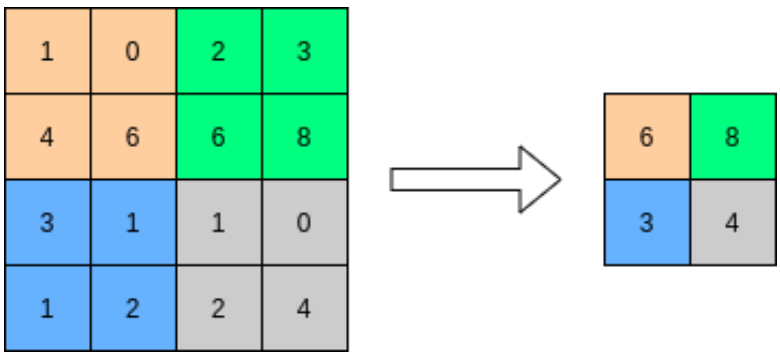

Рисунок 1.12 — Субдискретизация с функцией максимума и фильтром 2×2 с шагом 2

Размер области задается фильтром, который обычно равен 2×2. В качестве функции фильтра обычно используют функцию максимума.

Но так же применимы и другие, например функции среднего значения и L2-нормирования.

Другими словами работу слоя субдискретизации можно описать следующим образом: если при работе сверточного слоя уже были выявлены некоторые признаки, то в дальнейшем настолько подробные данные уже не нужны, то есть можно их сократить до менее подробных.

Общая архитектура CHC

В задачах распознавания жестовых символов CHC уже применялась для работы с русским[20] и итальянским[21] жестовыми языками. Не смотря на разницу в архитектурных особенностях методов, общий подход к построению сети можно разделить на следующие слои:

а) Сверточный слой.
б) Слой субдискретизации.
в) Полносвязный слой.

1.2.4 Сравнительный анализ выделенных методов классификации

В результате анализа предметной области были рассмотрены три метода классификации жестовых символов. Каждый класс обладает рядом функциональных преимуществ, которые выделяют его на фоне других подходов. В то же время имеется и ряд недостатков, который ограничивает область применимости данного класса методов.

Сравнение преимуществ и недостатков рассмотренных решений представлено в таблице 1.6.

Таблица 1.6 — Сравнительный анализ методов классификации

Название метода	Преимущества	Недостатки
СММ	Простая математическая структура	Каждая модель обучается только на экземплярах своего класса
	Возможность использования исходных данных без предобработки	Большое число неструктурированных параметров
		Максимизация отклика модели на свои классы без минимизации на другие
Самоорганизующаяся карта Кохонена	Обучение без учителя	Окончательный результат зависит от начальных установок
	Устойчивость к зашумленным данным	
	Скорость обучения	
СНС	Частичная инвариантность к масштабу	Склонность к переобучению
	Частичная инвариантность к повороту и сдвигу	Необходимость в большой обучающей выборке
	Созданы специально для обработки изображений	

Сравнительный анализ качества классификации на американском жестовом языке выделенных методов представлено в таблице 1.7.

Таблица 1.7 — Точность работы методов классификации

Название метода	Точность
СММ[19]	90,7 – 93,5%
Самоорганизующаяся карта Кохонена[18]	92%
СНС[27]	97,82%

Как видно из таблицы, использование СНС дает наилучший результат классификации в задачах распознавания жестовых символов. Данный тип нейронной сети изначально рассчитан на обработку изображений.

1.2.5 Капсульные нейронные сети

Капсульные нейронные сети (англ Capsule Neural Network) – предназначенная для распознавания изображений архитектура нейронных сетей. КНС были задуманы Джеффри Хинтоном в 1979 году, первые работы по ней опубликованы в 2017 году[28]. Идея данной сети является следствием критики сверточных нейронных сетей. Полученная в ходе свертки информация о входных данных частично теряется на этапе субдискретизации. Например в задачи распознавания лиц данная сеть учитывает наличие на изображении глаз, ушей, носа и губ, но игнорирует их взаимное расположение. Следствием этого может быть ложное распознавание деформированного лица, пример которого представлен на рисунке 1.13.

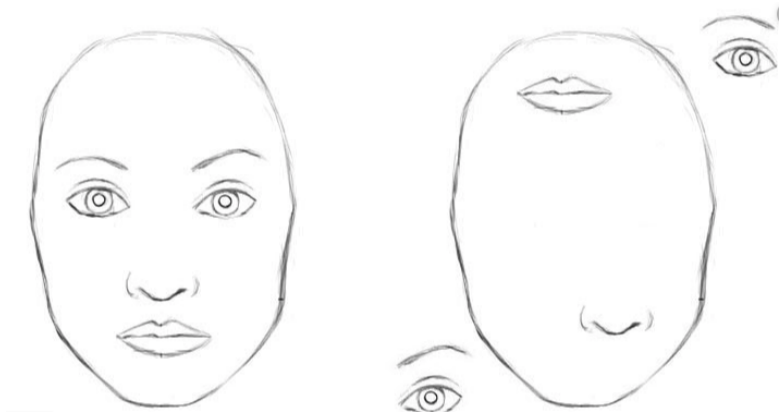

Рисунок 1.13 — Деформированное изображение лица

Особенностью КНС являются капсулы – группа нейронов, инкапсулирующая информацию о состоянии функции, которую обнаруживают в векторной форме.

В отличии от обычных нейронов, работающих со скалярными величинами, капсулы работают с векторами. Вычисление выходного значения капсулы происходит в соответствии с формулой 1.7, представляющее собой скалярное произведение векторов.

$$s_j = \sum_i c_{ij} \hat{u}_{i|j} \quad \hat{u}_{i|j} = W_{ij} u_i, \tag{1.7}$$

где

— u_i – вектор входных значений;
— W_{ij} – матрица аффинного преобразования;
— c_{ij} – коэффициент маршрутизации;
— s_j – выходное значение.

Заменой функции активизации, применяемой в нейронах для задания нелинейности данным, в капсулах является нормализация нормализация выходного вектора по формуле 1.8.

$$v_j = \frac{||s_j||^2}{1 + ||s_j||^2} \frac{s_j}{||s_j||} \tag{1.8}$$

Итоговые различия между капсулами и обычными нейронами представлена в таблице 1.8.

Капсулы являются расширением нейронов до векторной формы, что позволяет хранить, обрабатывать и передавать больше информации. Например в задачах распознавания лиц капсула может хранить не только признаки присутствия глаза на изображении, но и дополнительную информацию о его положении относительно других частей.

Благодаря этим особенностям данная архитектура инвариантна к поворотам и смещениям изображения, обучение происходит быстрее и требует меньший объем выборки. Применение данной архитектуры позволяет сократить ошибку классификации при поворотах на 43%, в сравнении с

Таблица 1.8 — Различия между капсулами и нейронами

Параметр	Капсула	Нейрон
Формат входных данных	вектор (u_i)	скаляр (x_i)
Преобразование входных данных	$\hat{u}_{i\|j} = W_{ij}u_i$	–
Сумматор	$s_j = \sum_i c_{ij}\hat{u}_{i\|j}$	$a_j = \sum_i w_i x_i + b$
Передаточная функция	$v_j = \frac{\|\|s_j\|\|^2}{1+\|\|s_j\|\|^2}\frac{s_j}{\|\|s_j\|\|}$	$h_j = f(a_j)$
Формат выходных значений	вектор (v_j)	скаляр (h_j)

CHC[29]. Эти утверждения были экспериментально доказаны на датасете рукописных цифр MNIST[28].

Автор предлагает использование КНС в качестве классификатора при построении метода классификации жестовых символов.

1.3 Вывод

Были представлены этапы работы известных методов распознавания жестовых символов. Рассмотрены возможные способы получения информации, их преимущества и недостатки.

Проведен сравнительный анализ методов предобработки изображений. Показано преимущество выделения силуэта кисти руки в сравнении с остальными методами с точки зрения скорости и качества выделения признаков с изображения.

Сравнительный анализ методов классификации показал, что CHC имеет наилучшее качество распознавания среди рассмотренных классификаторов. Принято решение использование данного решения в качестве базового при построении метода распознавания жестовых символов. Не смотря на высокую точность классификации, CHC имеют проблемы с инвариантностью к пространственным изменениям изображения, а также требуют большого объема обучающей выборки. В качестве альтернативы предложено использование КНС, архитектура которых направлена на устранение описанных недостатков сверточных сетей.

2 Конструкторский раздел

2.1 Архитектура программного продукта

Разрабатываемый метод состоит из 2 частей:

а) модуль предобработки входных данных;
б) модуль классификации жестового символа.

В качестве входных данных используются RGB изображения. Данный формат данных может быть получен с любого фото- и видео-устройства.

2.2 Предобработка изображений

В рассмотренном ранее методе выделения силуэта кисти руки предлагалась фильтрация пикселей изображения в цветовом пространстве YCbCr. Как показывает практика, данное решение имеет недостаток в виде артефактов выделения. Данная проблема возникает из-за наличия в фоновой части изображения пикселей, цвет которых близок к цвету кожи в данном цветовом пространстве.

Избежать подобные дефекты можно через дополнительную фильтрацию в цветовом пространстве HSV[30]. В итоговом методе пиксели исходного изображения проверяются одновременно и в обоих цветовых пространствах.

Для преобразования изображения из RGB в YCbCr используется формула 2.1.

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} + \begin{bmatrix} 65{,}81 & 128{,}551 & 24{,}966 \\ -37{,}797 & -74{,}203 & 112 \\ 112 & -93{,}786 & -18{,}214 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad 2.1$$

В HSV координатами цвета являются:

— H – цветовой тон, интервал значений $0 - 360°$;
— S – насыщенность, $0 - 1$;
— V – яркость, $0 - 1$;

Для получение значения координат пикселя в пространстве HSV, нормализуем его координаты в RGB по максимальному значению.

$$R' = R/255 \quad G' = G/255 \quad B' = B/255 \tag{2.2}$$

Зададим так же

$$C_{max} = max(R', G', B') \tag{2.3}$$

$$C_{min} = min(R', G', B') \tag{2.4}$$

$$\Delta = C_{max} - C_{min}. \tag{2.5}$$

Тогда координаты пикселя в пространстве HSV вычисляются следующим образом:

$$H = \begin{cases} 0^\circ & ,\Delta = 0 \\ 60^\circ \times (\frac{G'-B'}{\Delta} \mod 6) & ,C_{max} = R' \\ 60^\circ \times (\frac{B'-R'}{\Delta} + 2) & ,C_{max} = G' \\ 60^\circ \times (\frac{R'-G'}{\Delta} + 4) & ,C_{max} = B' \end{cases} \tag{2.6}$$

$$S = \begin{cases} 0 & ,C_{max} = 0 \\ \frac{\Delta}{C_{max}} & ,C_{max} \neq 0 \end{cases} \tag{2.7}$$

$$V = C_{max} \tag{2.8}$$

Алгоритм 1 описывает процесс получения силуэта кисти руки.
Входные данные: Изображение x

Выходные данные: Однотонное черно-белое изображение y

Получить изображение в YCbCr

$x_{ycbcr} \leftarrow YCbCr(x)$

Получить изображение в HSV

$x_{hsv} \leftarrow HSV(x)$

для каждого *пикселя* p *изображения* x *с координатами* i, j

выполнять

$Y =$ значение Y пикселя x_{ycbcr}

$Cb =$ значение Cb пикселя x_{ycbcr}

$Cr =$ значение Cr пикселя x_{ycbcr}

$H =$ значение H пикселя x_{hsv}

$S =$ значение S пикселя x_{hsv}

$V =$ значение V пикселя x_{hsv}

если $Y > 80$ *и* $135 \leq Cr \leq 180$ *и* $85 \leq Cb \leq 135$ *и* $H \leq 50°$ *и* $0{,}23 \leq S \leq 0{,}68$ **тогда**

$y[i][j] = 255$

конец

иначе

$y[i][j] = 0$

конец

конец

Алгоритм 1: Алгоритм выделения силуэта кисти руки

2.3 Классификация жестовых символов

Ранее было показано преимущество использование СНС в качестве классификатора в известных методах распознавания жестовых символов, выделены недостатки данного метода. Предложено использование КНС в качестве альтернативы с целью уменьшения обучающей выборки, ускорения процесса обучения и добавления методу инвариантности к аффинным преобразованиям изображения.

2.3.1 Алгоритм передачи данных между капсульными слоями

Передача информации между капсульными слоями происходит с помощью динамической маршрутизации. Целью данного алгоритма является передача выхода низкоуровневых капсул только тем капсулам, которые способны на основании полученных данных получить наилучший результат в контексте решаемой данной архитектурой задачи.

Все входные вектора капсулы после аффинного преобразования умножаются на коэффициенты маршрутизации, сумма которых равна единице. Значения данных коэффициентов задаются функцией softmax (формула 2.9).

$$c_{ij} = \frac{e^{b_{ij}}}{\sum_k e^{b_{ik}}} \tag{2.9}$$

Значение неизвестных b_{ij} вычисляются дискриминативно одновременно с остальными весовыми коэффициентами. Данный процесс зависит исключительно от взаимного расположения и типа капсул и не зависит от входных данных. Начальные коэффициенты связи затем итеративно уточняются на основании согласованности текущих выходов v_j низкоуровневых капсул j и предсказанием $\hat{u}_{i|j}$ высокоуровневой капсулы i.

Согласованность между капсулами определяется скалярным произведением $a_{ij} = v_j \hat{u}_{j|i}$. Оно рассматривается как логарифмическая вероятность и добавляется к исходному значению b_{ij} перед вычислением новых значений для всех коэффициентов связи, связывающих капсулу i с капсулами более низкого уровня.

Алгоритм 2 описывает итерационный процесс вычисления коэффициентов связи.

Входные данные: Вектор $\hat{u}_{j|i}$, количество итераций r, капсульный слой l

для каждого *индекса* i *капсулы слоя* l *и индекса* j *капсулы слоя* $l+1$ **выполнять**
  $b_{ij} \leftarrow 0$
конец
цикл r *итераций* **выполнять**
  для каждого *индекса* i *капсулы слоя* l **выполнять**
    $c_{ij} \leftarrow softmax(b_{ij})$
  конец
  для каждого *индекса* j *капсулы слоя* $l+1$ **выполнять**
    $s_j = \sum_i c_{ij}\hat{u}_{j|i}$
  конец
  для каждого *индекса* j *капсулы слоя* $l+1$ **выполнять**
    $v_j = \frac{||s_j||^2}{1+||s_j||^2}\frac{s_j}{||s_j||}$
  конец
  для каждого *индекса* i *капсулы слоя* l *и индекса* j *капсулы слоя* $l+1$ **выполнять**
    $b_{ij} \leftarrow b_{ij} + v_j\hat{u}_{j|i}$
  конец
конец

Алгоритм 2: Алгоритм динамической маршрутизации

2.3.2 Архитектура сети

Архитектура сети состоит из 4 слоев:

а) Входной слой. На вход сети подается черно-белое одноканальное изображение, где каждое значение пикселя означает интенсивность белого цвета.

б) Сверточный слой. Имеет 256 фильтров размерностью 9×9 со смещением 1. В качестве активационной функции используется ReLU (формула 2.10).

$$x_k = \max(u_k, 0), k = 1, ..., K \tag{2.10}$$

где

— x_k – выходное значение нейрона k-го слоя;

— u_k – результат работы сумматора нейрона k-го слоя.

в) Первый капсульный слой. Состоит из 32 капсул, каждая из которых представляет собой набор из 8 сверточных слоев с дает на выходе $N \times N$ векторов длины 8, где N - число строк или столбцов матрицы, полученной в результате свертки. Ядро свертки имеет размерность 9×9 со смещением 2. Условно, данную капсулу можно представить в виде группы подкапсул, объединенных одним набором весовых коэффициентов.

г) Выходной капсульный слой. Состоит из J капсул, где J – количество классов. Выходной вектор имеет длину 8. Каждая капсула принимает на вход все выходы с предыдущего слоя с применением алгоритма динамической маршрутизации. Для большей наглядности данный слой можно интерпретировать как полносвязный в СНС. Длина выходного вектора капсулы на данном слое соответствует вероятности принадлежности входных данных к классу, представленному конкретной капсулой, то есть итоговый класс изображения определяется по капсуле с выходным вектором наибольшей длины.

Схема нейронной сети представлена на рисунке 2.1.

Ошибка обучения вычисляется для каждой капсулы выходного слоя по следующей формуле:

$$L_k = T_k \max(0, m^+ - ||v_k||)^2 + \lambda(1 - T_k) \max(0, ||v_k|| - m^-)^2 \qquad (2.11)$$

где

— L_k – ошибка k - ой капсулы;

— T_k равен 1, если k-я капсула представляет текущий класс, иначе равен 0;

— m^+, m^- – варьируемые коэффициенты. В данной работе использовались значения 0,9 и 0,1 соответственно;

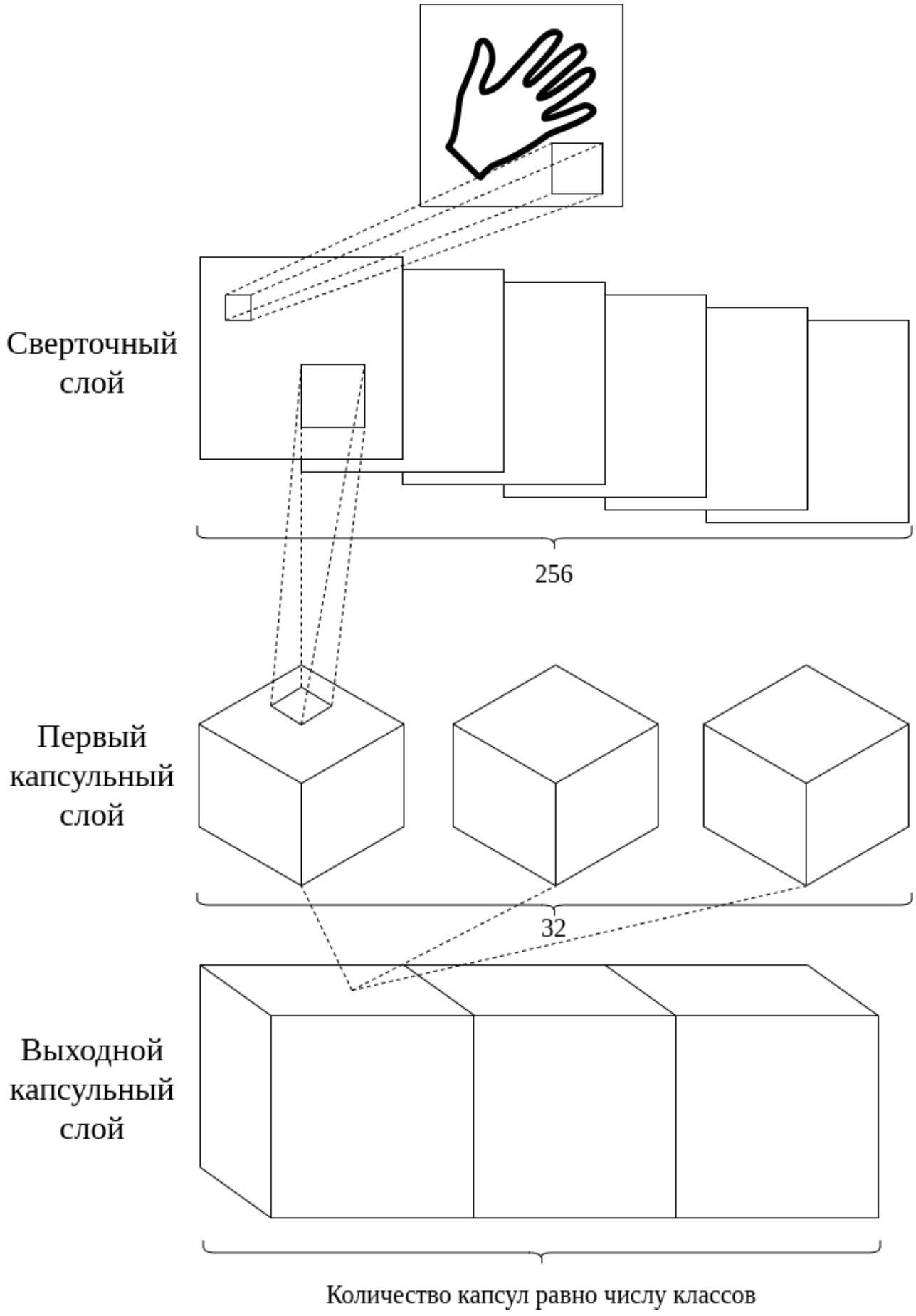

Рисунок 2.1 — Архитектура капсульной нейронной сети

— λ – коэффициент уменьшения исходных весов для отсутствующих классов, исключающий из процесса обучения сокращение длины вектора активности для всех сущностей. В данной работе используется значение 0,5.

Итоговая ошибка обучения вычисляется как сумма $L = \sum_k L_k$.

2.4 Вывод

В конструкторском разделе описывается построение метода распознавания жестовых символов. Дано подробное описание, построены схемы выбранных алгоритмов. При этом выделены основные этапы работы метода с указанием необходимых исходных данных для его работы и полученных результатов на каждом этапе.

3 Технологический раздел

В данном разделе описываются средства, используемые для разработки программного реализации построенного метода, требования для функционирования ПО, описываются результаты тестирования программного продукта.

3.1 Выбор средств разработки

3.1.1 Выбор языка программирования

Для программной реализации описанного метода был выбран язык программирования Python, так как он обладает следующими свойствами:

— большая база библиотек для работы с искусственными нейронными сетями, изображениями и математических расчетов;

— сочетание функционального, структурного и объектно-ориентированного подходов позволяет кратко описывать необходимые для решения поставленной задачи математические структуры;

— кроссплатформенность.

Описанные особенности позволяют при помощи Python одинаково удобно реализовывать как научно-исследовательские прототипы, так и коммерческие реализации программного продукта.

3.1.2 Выбор среды программирования и отладки

В качестве среды разработки для языка Python была выбрана кроссплатформенная IDE PyCharm, выбор которой обусловлен следующими предоставляемыми возможностями, упрощающими разработку приложения и способствующими повышению качества исходного кода:

— рефакторинг;

— навигация по проекту и исходному коду;

— встроенный отладчик;

— поддержка систем контроля версий;

— статический анализ кода.

3.1.3 Используемые библиотеки

В процессе реализации были использованы следующие библиотеки:

— OpenCV[31] – библиотека для обработки изображений. Использовалась для считывания, записи и реализации предобработки входных данных.

— scikit-learn[32] – библиотека для машинного обучения. Используется для форматирования входных данных.

— NumPy[33] – библиотека реализаций вычислительных алгоритмов, оптимизированных для работы с многомерными массивами. Используется для упрощения реализации математических операций.

— Tensorflow[34] – библиотека для машинного обучения. Используется для построения и обучения классификатора.

— Keras[35] – библиотека машинного обучения. Используется для построения модели классификатора, работающий на базе Tensorflow.

— Tkinter – графическая библиотека. Используется для создания пользовательского интерфейса на базе средств Tk.

3.2 Система контроля версий

В процессе разработки программы использовалась система контроля версий Git, позволяющая вносить в проект атомарные изменения, направленные на решения каких-либо задач. В случае обнаружения ошибок или изменения требований, внесенные изменения можно отменить. Кроме того, с помощью системы контроля версий решается вопрос резервного копирования.

Особенности Git:

— данная система контроля версий является децентрализованной, что позволяет иметь несколько независимых резервных копий проекта;

— поддерживается хостингом репозиториев GitHub;

— поддерживается средой разработки PyCharm;

— предоставляет широкие возможности для управления изменениями проекта и просмотра истории изменений.

3.3 Требования к вычислительной системе

Для запуска программы необходимо иметь установленный на ЭВМ интерпретатор для Python 3.6 с установленными библиотеками.

Так как выбранный язык программирования является кроссплатформенным, то требований к использованию операционной системы нет.

Обрабатываемые изображения не требуют большого объема оперативной памяти, но классификатор работает с большим числом параметров, поэтому рекомендуемый размер ОЗУ составляет не менее 256 Мб, желательна архитектура x64 (x86-64).

3.4 Формат данных

В качестве входных данных используются RGB изображения в следующих форматах:

— BMP – разработка компании Microsoft. В данном формате хранятся только однослойные растры. Значения пикселей могут иметь разрядности 1, 2, 4, 8, 16, 24, 32, 48 и 64 бит. При 8 и меньше бит в пикселе хранится индекс цвета в таблице цветов, а при большей – непосредственное значение в цветовой модели RGB.

— JPEG – формат хранения растровых изображений, способный сжимать изображения как с потерями, так и без потерь качества.

— PNG – графический формат, отличительной способностью которого является возможность хранения значений альфа-канала, отвечающую за прозрачность пикселя.

3.5 Проектирование архитектуры программного комплекса

Разрабатываемый программный комплекс состоит из следующих частей:

— модуль получения изображения жеста;
— модуль предобработки входных данных;
— модуль классификации жеста.

Формальная модель системы изображена на рисунке 3.1.

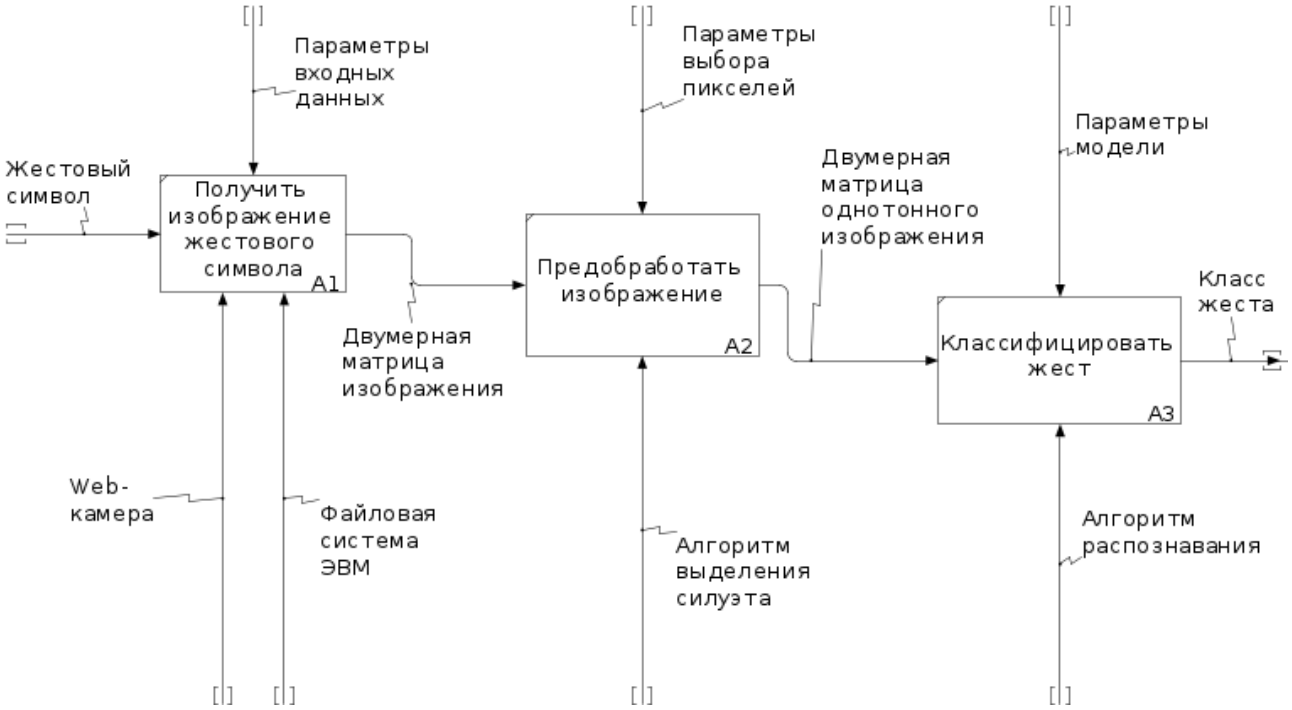

Рисунок 3.1 — Формальная модель системы классификации жестовых символов

Функционал обучения системы и распознавания жестов разделен на две отдельные подсистемы, объединенных единым хранилищем моделей, из-за разного подхода обработки данных. Итоговая архитектура программного комплекса представлена на рисунке 3.2.

Основной задачей системы обучения является подготовка моделей классификатора для определенного набора жестовых символов. Для исследования эффективности этапа предобработки входных данных, подсистема обучения способна создавать модели для изображений с предобработкой и без. Функциональные возможности подсистемы обучения представлены на рисунке 3.3.

Подсистема распознавания жестовых символов выполняет получение изображения с жесткого диска или с Web-камеры, предобработку данных и классификацию жеста. Функциональные возможности подсистемы распознавания жестовых символов изображены на рисунке 3.4.

Каждый этап функционирования системы был реализован в виде обособленных модулей с унифицированными API и форматами данных.

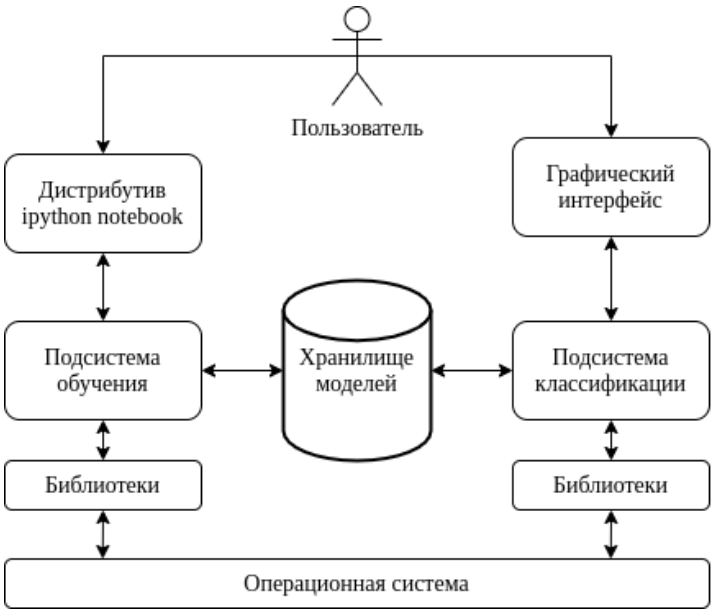

Рисунок 3.2 — Архитектура программного комплекса

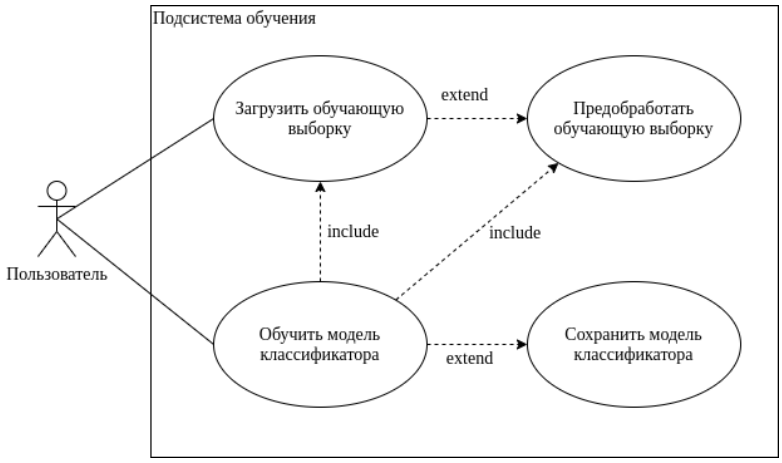

Рисунок 3.3 — Функциональные возможности подсистемы обучения

Функционал данного программного продукта может быть расширен путем добавления новых подсистем, а так же адаптирован под иные источники и форматы изображений жестовых символов.

3.6 Построение нейронной сети

На вход нейронной сети подается черно-белое изображение, полученное на этапе предобработки. Исходный код программной реализации выделения контура кисти руки представлен на алгоритме 3.

В представлении Tensorflow нейронная сеть является графом потока данных, в котором данные в виде многомерного массива переходят в

Рисунок 3.4 — Функциональные возможности подсистемы распознавания жестовых символов

разные узлы, в процессе чего происходят все необходимые вычисления. Из-за данного подхода процесс описания нейронных сетей с использованием нативного Tensorflow затруднителен. Для упрощения построения моделей можно использовать библиотеку Keras, которая предоставляет простой и удобный способ создания моделей глубокого обучения. Данная библиотека имеет большую базу широко используемых типов слоев искусственных нейронных сетей, а так же предоставляет возможность описывать собственные слои через наследование базового класса Layer.

Алгоритм 4 описывает построение КНС. Для преобразования входного изображения в тенсор используется слой layers.Input.

Капсулы первого капсульного слоя, как описано выше, представляют собой комбинацию нескольких сверточных слоев. В следствии этого, реализация данного слоя возможна стандартными слоями библиотеки Keras, как показано на алгоритме 5.

Второй капсульный слой нельзя реализовать стандартными средствами пакета layers библиотеки Keras из-за необходимости самостоятельного описания алгоритма динамической маршрутизации. Для этого был описан класс, CapluleLayer, наследованный от класса layers.Layer. Реализация динамической маршрутизации представлена в алгоритме 6. Исходный код

реализации вычисления формулы 1.7 и вычисление длин выходных векторов представлены на алгоритмах 7 и 8 соответственно.

Итоговый граф изображен на рисунке 3.5.

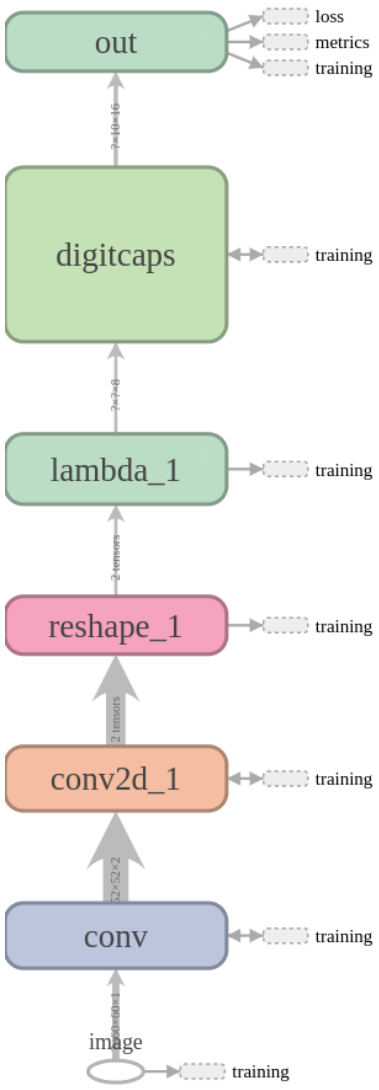

Рисунок 3.5 — Граф модели

3.7 Руководство пользователя

Установка программного обеспечения

Для запуска разработанного программного комплекса требуется установленный на ПК интерпретатор для Python 3. Все необходимые библиотеки указаны в файле requirements.txt, находящемся в корневом каталоге проекта. Средствами каталога программного обеспечения PyPI

(Python Package Index) все зависимости устанавливаются выполнением одной команды в терминале:

$ pip3 install -r requirements.txt

Подсистема обучения

Обучение классификаторов происходит с помощи Jupyter Notebook – интерактивной оболочки для языка Python. Данная технология позволяет объединяет код и вывод в окне одного документа, содержащего текст, математические уравнения и визуализации. Такой пошаговый подход обеспечивает быстрый, последовательный процесс разработки, поскольку вывод для каждого блока показывается сразу же.

Для работы и выполнения кода ноутбуков необходимо запустить сервер Jupiter, выполнив в корневом каталоге команду:

$ jupiter notebook

После этого в стандартном браузере системы откроется сайт с URL http://localhost:8888/tree с файловым менеджером, открытым в корневом каталоге. Для запуска ноутбуков обучения нужно открыть в этом же окне любой файл из папки notebooks и нажать кнопку Run All. Результаты обучения сохраняются в папку data/название выборки/tensorboard.

Подсистема распознавания жестовых символов

В подсистеме распознавания жестовых символов используется уже обученная нейронная сеть, файл с весовыми коэффициентами которой находится в папке data/название выборки/tensorboard. Для удобства пользователя программное обеспечение поставляется с набором уже обученных классификаторов. Для запуска программы используется команда

$ python3 main.py

После запуска программы появляется главное окно (рисунок 3.6).

Визуально область окна разделена на 3 зоны:

— Зона загрузки данных. Управляет методом получения изображения. Изначально указан метод «Загрузить с диска», отображающий кнопку «Открыть файл», при нажатии которого открывается диалоговое окно выбора

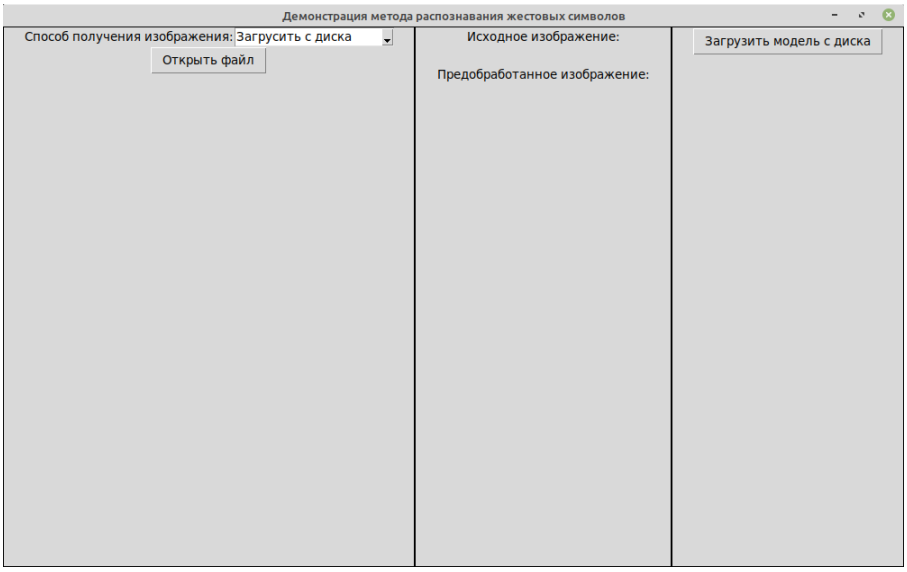

Рисунок 3.6 — Окно программы после запуска

файла изображения. При переключении на метод «Снять с web-камеры» открывается окно с демонстрацией видео-потока с web-камеры и кнопкой «Сделать снимок» (рисунок 3.7).

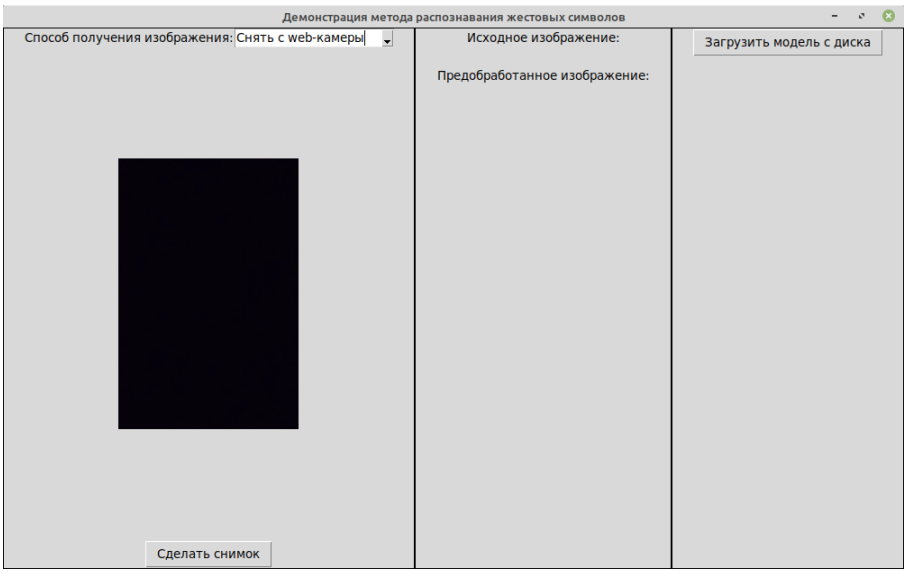

Рисунок 3.7 — Окно программы в режиме получения изображения с web-камеры

— Зона предварительно обработки. После получения входных данных отображает два изображения: оригинальное и результат предобработки (рисунок 3.8).

— Зона классификации. Кнопка «Загрузить модель с диска» открывает диалоговое окно выбора файла весовых коэффициентов КНС. Ниже

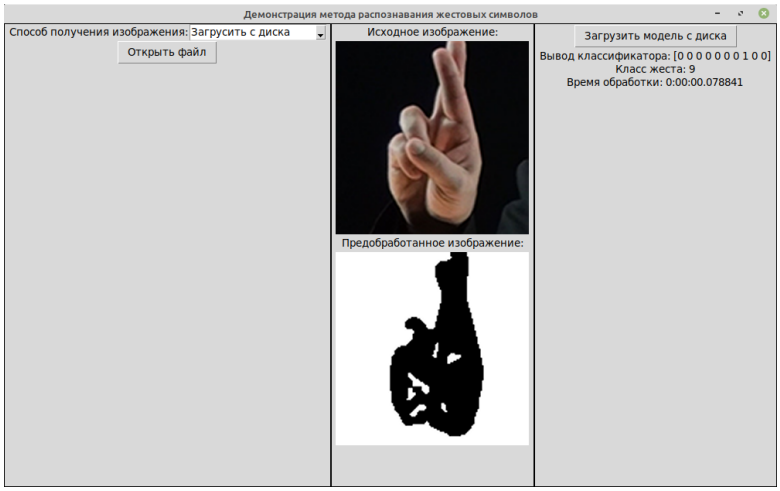

Рисунок 3.8 — Окно программы после классификации

отображаются результаты классификации: исходный вывод модели, интерпретированный результат классификации и время полной обработки изображения.

3.8 Вывод

Была разработана архитектура программного комплекса для демонстрации работы метода классификации жестовых символов. Процесс обучения и эксплуатации системы выделены в отдельные подсистемы для упрощения разработки. Проведена декомпозиция подсистем на модули, позволяющие расширять функционал программного продукта путем добавления новых модулей.

На основе спроектированной архитектуры был разработан программный комплекс на языке Python с использованием библиотек OpenCV, scikit-learn, NumPy, Tensorflow, Keras и Tkinter.

Продукт был разработан на ЭВМ со следующими характеристиками:

— Процессор: Intel© Core™ i5-8250U.
— Тактовая частота: 1.60ГГц × 4
— Объем оперативной памяти: 8 Гб.
— Графическая карта: Intel Corporation UHD Graphics 620.
— Операционная система: Linux Mint 19.3 Cinnamon.
— Версия ядра Linux: 5.3.0-53-generic.

4 Исследовательский раздел

В рамках дипломного проекта был проведен ряд экспериментов, направленных на исследование построенного метода распознавания жестовых интерфейсов. Целью проведенных исследований является выяснение зависимости качества классификации от этапа предобработки и количества итераций в алгоритме динамической маршрутизации.

В качестве входных данных использовались изображения жестов, выполненные как мужчинами, так и женщинами различной национальности.

4.1 Описание тестовых данных

Для проведения вычислительных экспериментов использовались следующие наборы данных:

— ASL Finger Spelling Dataset – набор изображений дактилей американского жестового языка. Использовался для оценки качества распознавания в описанных ранее методах [18, 19, 27]. На основании результатов данной выборке делается вывод о качестве построенного метода относительно конкурентов. Данный датасет состоит из двух частей: изображений 24 дактилических жестов (в данную выборку не входят буквы «j» и «z», так как являются динамическими) и карт глубин. В данной работе использовалась первая часть, которая состоит из 65000 изображений с непостоянным размером в цветовом пространстве RGB. Жесты демонстрируются пятью разными людьми.

— RSL by Oleg Potkin – набор данных русского дактиля. Содержит 1042 RGB изображения размером 128×128 пикселей. Разделен на 10 классов-букв: «а», «б», «в», «г», «е», «и», «о», «п», «с».

— Numbers – набор данных с изображением жестов цифр. Состоит из из 1125 RGB изображений.

4.2 Формальная модель и описание условий исследования

Для выявления зависимости качества распознавания от количества итераций в алгоритме динамической маршрутизации в рамках исследования для одного набора данных строились модели для двух, трех, четырех, пяти,

шести и семи итераций. Каждая модель обучалась на предобработанных и оригинальных наборах данных.

Для проведения вычислительных экспериментов была разработана формальная модель, представленная на рисунке 4.1.

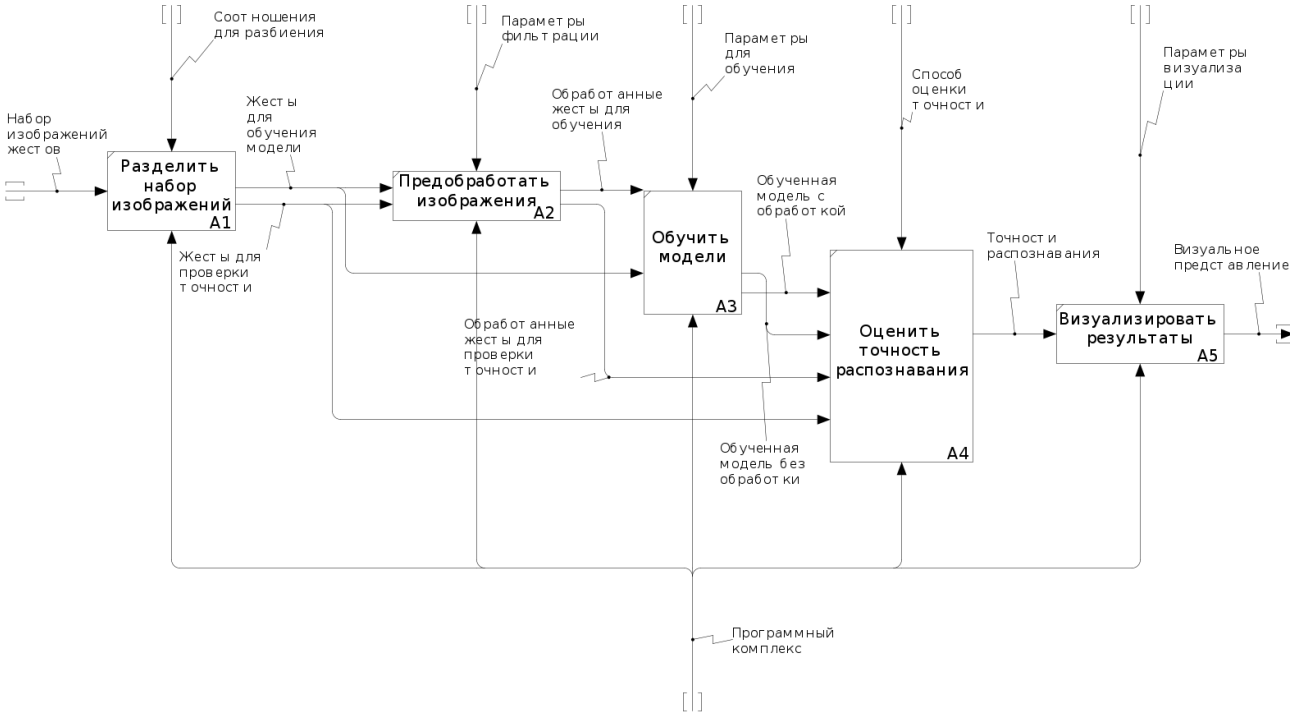

Рисунок 4.1 — Формальная модель эксперимента

В качестве метрики качества распознавания метода используется вероятность корректной классификации жестового символа (формула 4.1).

$$\text{Точность} = \frac{\text{Верные классификации}}{\text{Ложные классификации} + \text{Верные классификации}} \tag{4.1}$$

Каждый набор изображений был разделен в соотношении 20% для валидации, 64% для обучения и 16% для тестирования.

Исследования проводились с использованием платформы Google Colaboratory, предоставляющая бесплатное выполнение файлов ipython notebook с использованием GPU и TPU. В рамках одной сессии предоставляется 25,51 Гб ОЗУ и 68,40 дискового пространства.

4.3 Результаты исследований

Для оптимальной настройки описанного метода были проведены вычислительные эксперименты с целью определения зависимости точности распознавания от числа итераций динамичесой маршрутизации. Для выяснения влияния этапа предобработки на качество работы классификатора данные вычисления проводились для обработанных и исходных наборов изображений. Результаты экспериментов были обобщены и виде графиков, представленных на рисунках 4.2, 4.3 и 4.4

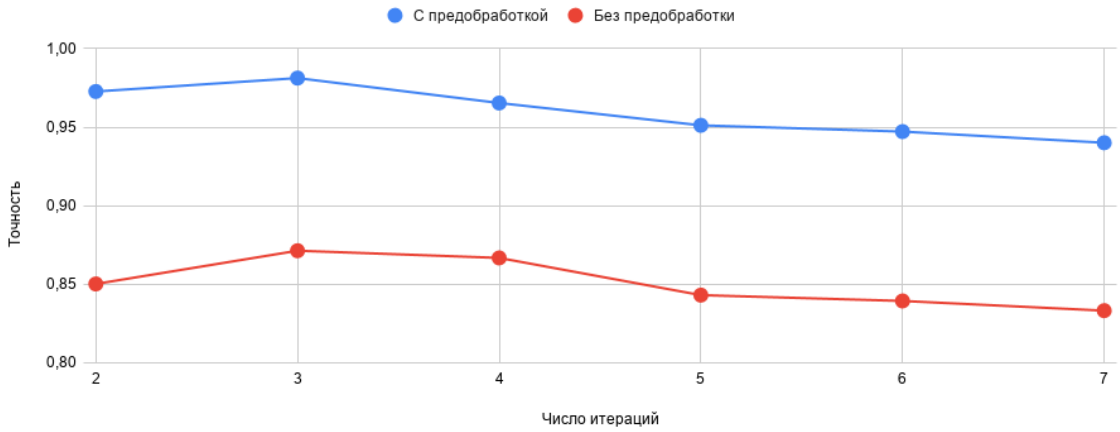

Рисунок 4.2 — Зависимость точности распознавания от предобработки входных данных и числе итераций на датасете ASL Finger Spelling Dataset

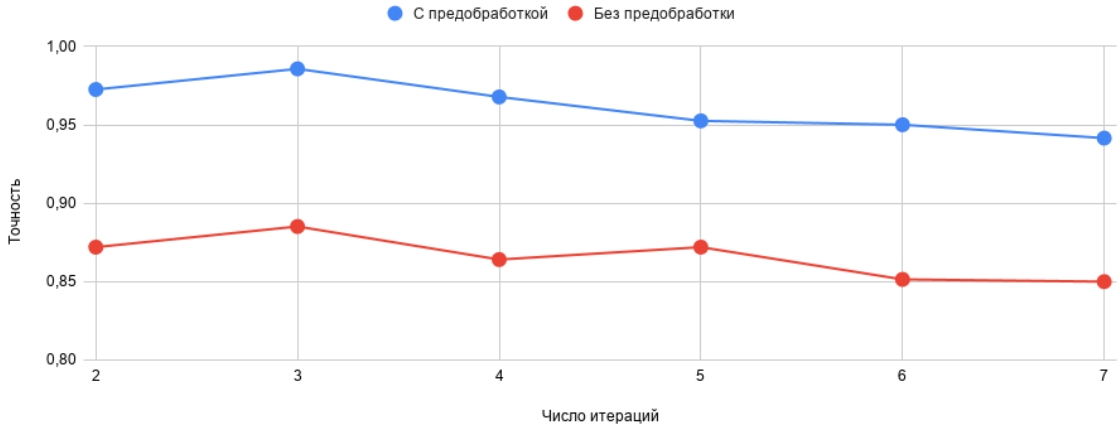

Рисунок 4.3 — Зависимость точности распознавания от предобработки входных данных и числе итераций на датасете RSL by Oleg Potkin

Исследование показало, что предобработка изображений позволяет увеличить точность распознавания в среднем на 10-15 %, как видно

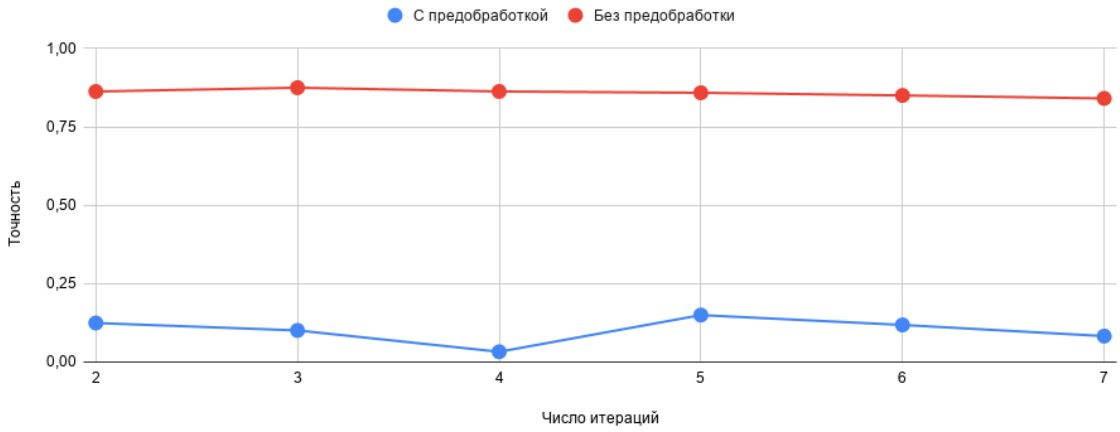

Рисунок 4.4 — Зависимость точности распознавания от предобработки входных данных и числе итераций на датасете Numbers

на рисунках 4.2 и 4.3. С другой стороны, есть вероятность получения зашумленных предобработанных изображений, следствием чего является потеря работоспособности классификатора (рисунок 4.4). Пример зашумленного предобработанного изображения представлен на рисунке 4.5.

а) б)

Рисунок 4.5 — Результат предобработки изображения из датасета Number с зашумлением: а) исходное изображение; б) предобработанное изображение

Наибольшая точность распознавания достигается при трех итерациях алгоритма маршрутизации, как видно на рисунках 4.2 и 4.3.

4.4 Анализ полученных результатов

В связи с тем, что набор ASL Finger Spelling Dataset использовался в ряде других исследований в области распознавания жестовых символов,

опубликованных в последние годы, стало возможным сравнить результаты распознавания предложенного метода с аналогичными методами.

Точность распознавания для других методов представлена на рисунке 4.6.

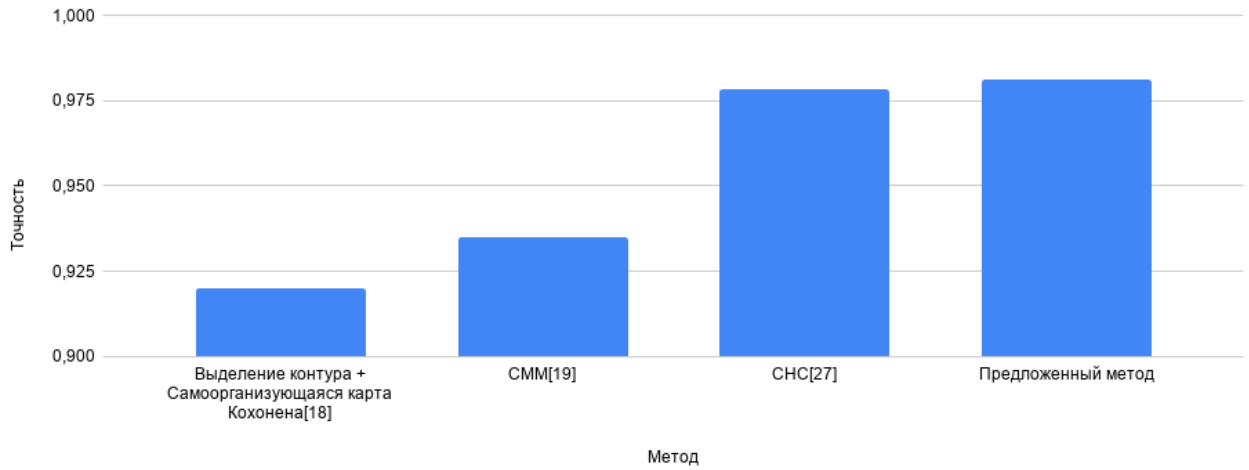

Рисунок 4.6 — Сравнение точности распознавания известных методов

Полученные результаты показывают, что разработанный метод на 3% позволяет повысить точность распознавания жестовых символов.

4.5 Вывод

Были выбраны три набора данных, на которых проводились исследования построенного метода.

Была разработана формальная модель для проведения экспериментов, нацеленных на исследование зависимости точности распознавания от этапа предобработки и числа итераций в алогритме динамической маршрутизации.

Показано повышение качества работы метода от этапа предоработки изображений, а так же выявлена зависимость работоспособности системы от зашумленности полученных данных.

Исследования показали, что лучшая точность распознавания достигается при трех итерациях алгоритма динамической маршрутизации.

Точность распознавания двух наборов данных из трех составила свыше 95%.

Полученные результаты распознавания показали увеличение точности в сравнении с известными аналогами.

ЗАКЛЮЧЕНИЕ

В результате проделанной работы были решены следующие задачи:

— был проведен анализ предметной области;

— были проанализированы существующие решения;

— на основе полученных во время анализа данных был разработан собственный метода распознавания жетовых символов;

— предложенный метод был реализован в программном продукте;

— был проведен анализ качества разработанного метода.

В результате тестирования и эксперимента было установлено, что разработанный метод имеет зависимость от качества изображения, полученного на этапе обработки: повышение точности при нормальном изображении и потеря работоспособности при зашумленном. Так же показано, что данный метод позволяет увеличить точность распознавания на 3% в сравнении с известными аналогами.

Результаты исследовательской деятельности были отражены в печатной работе[14].

Развитие разработанного метода можно осуществлять по следующим направлениям:

— модификация цветового фильтра путем автоматизации подбора конфигурации;

— увеличение скорости обучения и работы, а также качества работы классификатора путем оптимизации предложенной архитектуры, добавлением сверточных и капсульных слоев, применением другого алгоритма динамической маршрутизации.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Sign Language Recognition Application Systems for Deaf-Mute People: A Review Based on Input-Process-Output / Suharjito Suharjito, Ricky Anderson, Fanny Wiryana et al. // *Procedia Computer Science*. — 2017. — 12. — Vol. 116. — Pp. 441–448.

2. Fast sign language recognition benefited from low rank approximation / Hanjie Wang, Xiujuan Chai, Yu Zhou, Xilin Chen. — 2015. — 07.

3. *Mohandes, Mohamed*. Arabic Sign Language Recognition using the Leap Motion Controller / Mohamed Mohandes, Salihu Oladimeji, Mohamed Deriche. — 2014. — 06.

4. HOW MANY SMARTPHONES ARE IN THE WORLD? — `https://www.bankmycell.com/blog/how-many-phones-are-in-the-world`. — Дата обращения: 14-05-2020.

5. A vision-based sign language recognition system using tied-mixture density HMM / Liang-Guo Zhang, Yiqiang Chen, Gaolin Fang et al. — 2004. — 01. — Pp. 198–204.

6. *Sobel, Irwin*. An Isotropic 3x3 Image Gradient Operator / Irwin Sobel // *Presentation at Stanford A.I. Project 1968*. — 2014. — 02.

7. *Prewitt, Judith M. S.* Object Enhancement and Extraction Picture Processing and Psychopictorics / Judith M. S. Prewitt // *Academic*. — 1970. — Pp. 75–149.

8. *Roberts, Lawrence*. Machine Perception of Three-Dimensional Solids / Lawrence Roberts. — 1963. — 01.

9. *Canny, John*. A Computational Approach To Edge Detection / John Canny // *Pattern Analysis and Machine Intelligence, IEEE Transactions on*. — 1986. — 12. — Vol. PAMI-8. — Pp. 679 – 698.

10. *Phung, S.L.* Skin segmentation using color and edge information / S.L. Phung, Abdesselam Bouzerdoum, Douglas Chai. — 2003. — 08. — Pp. 525 – 528 vol.1.

11. *Siddharth, Joshi*. Face detection / Joshi Siddharth, Srivastava Gaurav // *E368: Digital Image Processing*. — 2003. — Pp. 101 – 112.

12. *Gonzalez, Rafael C.* Digital Image Processing / Rafael C. Gonzalez, Richard E. Woods. — Third edition. — Pearson Prentice Hall, 2008. — Pp. 631 – 635.

13. Hand Keypoint Detection in Single Images using Multiview Bootstrapping / Tomas Simon, Hanbyul Joo, Iain Matthews, Yaser Sheikh. — 2017. — 04.

14. *Танцевов, Григорий Михайлович.* Исследование алгоритмов предобработки изображения кисти руки, применимых к распознаванию жестовых символов / Григорий Михайлович Танцевов, Константин Анатольевич Майков. — 2020. — 01. — Pp. 61–74.

15. Hand Detection and Tracking Using the Skeleton of the Blob for Medical Rehabilitation Applications / Pedro Gil-Jiménez, Beatriz Losilla-López, Rafael Torres et al. — 2012. — 06. — Pp. 130–137.

16. *Kasprzak, Włodzimierz.* Hand Gesture Recognition in Image Sequences Using Active Contours and HMMs / Włodzimierz Kasprzak, Artur Wilkowski, Karol Czapnik. — 2009. — 01. — Pp. 248–255.

17. *Kharate, Gajanan.* Vision based multi-feature hand gesture recognition for indian sign language manual signs / Gajanan Kharate, Archana Ghotkar // *International Journal on Smart Sensing and Intelligent Systems.* — 2016. — 03. — Vol. 9. — Pp. 124–147.

18. Hand gesture recognition using self organizing map for Human Computer Interaction / Ujjwal Karn, Nagaraj Bhat, Y.V. Venkatesh, Dhruv Vig. — 2013. — 08.

19. *Starner, Thad.* Visual Recognition of American Sign Language Using Hidden Markov Models / Thad Starner, Massachusetts Group. — 1995. — 05.

20. *Potkin, Oleg.* Static gestures classification using Convolutional Neural Networks on the example of the Russian Sign Language / Oleg Potkin, Andrey Philippovich. — 2018. — 06.

21. Sign Language Recognition Using Convolutional Neural Networks / Lionel Pigou, Sander Dieleman, Pieter-Jan Kindermans, Benjamin Schrauwen // Computer Vision - ECCV 2014 Workshops / Ed. by Lourdes Agapito, Michael M. Bronstein, Carsten Rother. — Cham: Springer International Publishing, 2015. — Pp. 572–578.

22. *Helske, Jouni.* MINIMUM DESCRIPTION LENGTH BASED HIDDEN MARKOV MODEL CLUSTERING FOR LIFE SEQUENCE ANALYSIS / Jouni Helske, Mervi Eerola, Ioan Tabus. — 2010. — 08.

23. *Fridman, Moshe.* Hidden Markov Model Regression / Moshe Fridman. — 1997. — 01.

24. Classification with hidden markov model / B. Benyacoub, S. ElBernoussi, A. Zoglat, Ismail El Moudden // *Applied Mathematical Sciences.* — 2014. — 01. — Pp. 2483–2496.

25. A Chinese sign language recognition system based on SOFM/SRN/HMM / Wen Gao, Gaolin Fang, Debin Zhao, Yiqiang Chen // *Pattern Recognition.* — 2004. — 12. — Vol. 37. — Pp. 2389–2402.

26. *Hubel, D. H.* Receptive fields and functional architecture of monkey striate cortex / D. H. Hubel, T. N. Wiesel. — 1968.

27. *Garcia, Brandon.* Real-time American Sign Language Recognition with Convolutional Neural Networks / Brandon Garcia, Sigberto Alarcon Viesca. — 2016.

28. *Sabour, Sara.* Dynamic Routing Between Capsules. — 2017.

29. *Hinton, Geoffrey.* Matrix capsules with EM routing. — 2018.

30. Human Skin Detection Using RGB, HSV and YCbCr Color Models / S. Kolkur, Dhananjay Kalbande, P. Shimpi et al. — 2017. — 08.

31. OpenCV. — `https://opencv.org/`. — Дата обращения: 14-05-2020.

32. scikit-learn. — `https://scikit-learn.org/stable/index.html`. — Дата обращения: 14-05-2020.

33. NumPy. — `https://numpy.org/`. — Дата обращения: 14-05-2020.

34. TensorFlow. — `https://www.tensorflow.org/`. — Дата обращения: 14-05-2020.

35. Keras. — `https://keras.io/`. — Дата обращения: 14-05-2020.

ПРИЛОЖЕНИЕ А

```
import cv2
import numpy as np
def skin_detector(img):
    hsv_image = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    hsv_mask = cv2.inRange(hsv_image, (0, 15, 0), (17, 170,
        255))
    y_cr_cb_img = cv2.cvtColor(img, cv2.COLOR_BGR2YCrCb)
    y_cr_cb_mask = cv2.inRange(y_cr_cb_img, (0, 135, 85), (255,
        180, 135))
    mask = cv2.bitwise_and(y_cr_cb_mask, hsv_mask)
    result = cv2.bitwise_not(mask)
    kernel = np.ones((3, 3), np.uint8)
    result = cv2.erode(result, kernel, iterations=1)
    result = cv2.dilate(result, kernel, iterations=1)
    return result
```

Алгоритм 3: Исходный код предобработки изображения

```
from keras import layers, models
def CapsNet(input_shape, n_class, num_routing):
    x = layers.Input(shape=input_shape, name="image")
    conv5 = layers.Conv2D(filters=256, kernel_size=9,
        strides=1, padding='valid', activation='relu',
        name='conv')(x)
    primarycaps = PrimaryCap(conv5, dim_vector=8,
        n_channels=32, kernel_size=9, strides=2, padding='valid')
    digitcaps = CapsuleLayer(num_capsule=n_class,
        dim_vector=16, num_routing=num_routing,
        name='digitcaps')(primarycaps)
    out_caps = Length(name='out')(digitcaps)

    return models.Model(x, out_caps)
```

Алгоритм 4: Исходный код инициализации модели

```
from keras import layers
def PrimaryCap(inputs, dim_vector, n_channels, kernel_size,
    strides, padding):
    output = layers.Conv2D(filters=dim_vector*n_channels,
        kernel_size=kernel_size, strides=strides,
        padding=padding)(inputs)
    outputs = layers.Reshape(target_shape=[-1,
        dim_vector])(output)
    return layers.Lambda(squash)(outputs)
```

Алгоритм 5: Исходный код инициализации первого капсульного слоя

```
import tensorflow as tf
import keras.backend as K
class CapsuleLayer(layers.Layer):
  def call(self, inputs):
    inputs_expand = K.expand_dims(K.expand_dims(inputs, 2), 2)
    inputs_tiled = K.tile(inputs_expand, [1, 1,
        self.num_capsule, 1, 1])
    inputs_hat = tf.scan(lambda ac, x: K.batch_dot(x, self.W,
        [3, 2]), elems=inputs_tiled,
        initializer=K.zeros([self.input_num_capsule,
        self.num_capsule, 1, self.dim_vector]))
    assert self.num_routing>0, 'The num_routing should be >0.'
    for i in range(self.num_routing):
      c = tf.nn.softmax(self.bias, dim=2)
      outputs = squash(K.sum(c * inputs_hat, 1, keepdims=True))
      if i != self.num_routing - 1:
        self.bias += K.sum(inputs_hat * outputs, -1,
            keepdims=True)
    return K.reshape(outputs, [-1, self.num_capsule,
        self.dim_vector])
```

Алгоритм 6: Исходный код реализации динамической маршрутизации

```
import keras.backend as K
def squash(vectors, axis=-1):
    s_squared_norm = K.sum(K.square(vectors), axis,
        keepdims=True)
    scale = s_squared_norm / (1 + s_squared_norm) /
        K.sqrt(s_squared_norm)
    return scale * vectors
```

Алгоритм 7: Исходный код реализации формулы 1.7

```
import keras.backend as K
from keras import layers
class Length(layers.Layer):
    def call(self, inputs, **kwargs):
        return K.sqrt(K.sum(K.square(inputs), -1))
    def compute_output_shape(self, input_shape):
        return input_shape[:-1]
```

Алгоритм 8: Исходный код реализации получения выходных значений КНС