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 ИУ «Информатика, искусственный интеллект и системы управления»

КАФЕДРА ИУ-1 «Системы автоматического управления»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

НА ТЕМУ:

Система распознавания жестов на видеопоследовательности для реализации интерфейсов человеко-машинного взаимодействия

Студент	ИУ1-81Б	28/05/2022	Д.И. Юдаков
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Руководитель		28/05/2022	К.В. Парфентьев
		(Подпись, дата)	(И.О. Фамилия)
Нормконтролер		28/05/2022	Т.Ю. Цибизова
		(Подпись, дата)	(И.О. Фамилия)

2022 г.

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)

УТВЕРЖДАЮ

Заведующий кафедрой ИУ-1
(Индекс)

_____ К.А. Неусыпин
(И.О. Фамилия)

« 11 » февраля 20 22 г.

З А Д А Н И Е
на выполнение выпускной квалификационной работы бакалавра

Студент группы ИУ1-81Б

Юдаков Дмитрий Игоревич
(фамилия, имя, отчество)

Тема квалификационной работы Система распознавания жестов на видеопоследовательности для реализации интерфейсов человеко-машинного взаимодействия

Источник тематики (НИР кафедры, заказ организаций и т.п.)

Тема квалификационной работы утверждена распоряжением по факультету
№ _____ от « ____ » _____ 20 _____ г.

Часть 1. _____

Часть 2.

Часть 3.

Оформление квалификационной работы:

Расчетно-пояснительная записка на _____ листах формата А4.

Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)

Презентация на слайдах

Дата выдачи задания « 11 » февраля 20 22 г.

В соответствии с учебным планом выпускную квалификационную работу выполнить в полном объеме в срок до « 28 » мая 20 22 г.

Руководитель ВКР 11/02/2022 (Подпись, дата) К.В. Парфентьев (И.О. Фамилия)

Студент 11/02/2022 (Подпись, дата) Д.И. Юдаков (И.О. Фамилия)

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 ИУ

КАФЕДРА ИУ-1

ГРУППА _____

УТВЕРЖДАЮ

Заведующий кафедрой ИУ-1
(Индекс)

_____ К.А. Неусыпин
(И.О. Фамилия)

« 11 » февраля 20 22 г.

КАЛЕНДАРНЫЙ ПЛАН
выполнения выпускной квалификационной работы

студента: Юдакова Дмитрия Игоревича
(фамилия, имя, отчество)

Тема квалификационной работы Система распознавания жестов на видеопоследовательности для реализации интерфейсов человеко-машинного взаимодействия

№ п/п	Наименование этапов выпускной квалификационной работы	Сроки выполнения этапов		Отметка о выполнении	
		план	факт	Должность	ФИО, подпись
1.	Задание на выполнение работы. Формулирование проблемы, цели и задач работы	11/02/2022 дата	11/02/2022 дата	Руководитель ВКР	
2.	1 часть	26/03/2022 дата	26/03/2022 дата	Руководитель ВКР	
3.	Утверждение окончательных формулировок решаемой проблемы, цели работы и перечня задач	26/03/2022 дата	26/03/2022 дата	Заведующий кафедрой	
4.	2 часть	27/04/2022 дата	27/04/2022 дата	Руководитель ВКР	
5.	3 часть	10/05/2022 дата	10/05/2022 дата	Руководитель ВКР	
6.	1-я редакция работы	15/05/2022 дата	15/05/2022 дата	Руководитель ВКР	
7.	Подготовка доклада и презентации	27/05/2022 дата	27/05/2022 дата	Руководитель ВКР	
8.	Заключение руководителя	28/05/2022 дата	28/05/2022 дата	Руководитель ВКР	
9.	Допуск работы к защите на ГЭК (нормоконтроль)	28/05/2022 дата	28/05/2022 дата	Нормоконтролер	
10.	Внешняя рецензия	28/05/2022 дата	28/05/2022 дата		
11.	Защита работы на ГЭК	дата	дата		

Студент: 11/02/2022 (подпись, дата)

Руководитель работы: 11/02/2022 (подпись, дата)

АННОТАЦИЯ

??????????????????????????????????????

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Вычислительная способность компьютера стала необходимой в жизни современного человека – без неё довольно трудно жить и работать в нынешних реалиях, а именно его умение быстро помочь в решении самых разнообразных задач, начиная от проведения инженерных расчётов до оптимизации управления предприятием, высоко оценивается человечеством. Камнем преткновения остаётся взаимодействие системы «человек-машина» -- оно должно быть максимально упрощено, чтобы люди и машины коммуницировали на уровне естественных средств общения.

Развитие компьютерных технологий получило прирост за последнюю половину века и стремительно увеличивается и по сей день. Они уже способны обрабатывать и анализировать информацию подобно человеку: распознавать текст, изображения, анализировать звуки и мелодии, произносить осмысленные предложения, распознавать голосовые команды и реагировать на прикосновения пальцев. Но одной из самых приоритетных и наиболее трудных задач в области информационных технологий и интеллектуальных систем является задача распознавания жестов.

Актуальность рассматриваемой тематики обусловлена возможностью применения предлагаемого подхода для управления объектами без тактильного контакта и голосовой идентификации команд, а также простотой с точки зрения конечного пользователя.

В данной работе идёт речь о детектировании кисти руки, извлечения её конфигурации с помощью ключевых точек и распознавание жеста для генерации в соответствии с ним конкретного сигнала выхода. В ходе работы исследуются различные методы и техники детектирования кисти человека и распознавания жеста, в частности методы без и с использованием средств машинного обучения. Важным требованием для реализации является минимизация времени задержки на обработку каждого кадра (скорость работы) и точность идентификации жеста.

Поставленной **целью** является разработка метода детектирования, извлечения конфигурации и классификации жеста с помощью языка высокого уровня Python.

Для достижения цели необходимо решить ряд задач:

1. Сделать обзор существующих решений детектирования кисти руки и распознавания жестов.
2. Сделать обзор теоретического материала по детектированию кисти руки человека и её последующего описания с помощью ключевых точек.
3. Найти и описать наиболее популярные методы обнаружения кисти человека и выделить из них самые эффективные.
4. Ознакомиться с методами извлечения конфигураций кисти человека.
5. Реализовать несколько алгоритмов решения задачи распознавания жестов и провести их сравнение.

1. ОБЗОР СУЩЕСТВУЮЩИХ РЕШЕНИЙ РАСПОЗНАВАНИЯ ЖЕСТОВ

Несмотря на актуальность задачи, уже существуют различные способы распознавания жестов. Эти способы можно разделить на несколько групп, объединённых по способу распознавания рук человека. Рассмотрим некоторые из реализаций данных групп.

1.1. Распознавание при помощи вычисления расстояния между точками

Примером реализации данного метода является «Система распознавания жестов в реальном времени с цветными перчатками» (Real-Time Hand-Tracking with a Color Glove), которую создали Роберт Ванг (Robert Y. Wang) и Йован Попович (Jovan Popovic) из лаборатории компьютерных наук и искусственного интеллекта Массачусетского технологического института [14].

Распознавание жестов возможно при надевании цветных перчаток, по которым программа распознаёт жесты рук при помощи веб-камеры по наличию цветных пикселей на изображении. После обнаружения этих пикселей происходит вычисление расстояния между ними. Вычислив это расстояние, программа выдаёт информацию о распознанном жесте. Вид цветных перчаток изображён на рисунке 1.

Рис. 1. Перчатки, при помощи которых программа распознаёт жесты.

Этот способ распознавания очень интересен в своей реализации, но он имеет огромный недостаток: необходимость надевания специальных перчаток для того, чтобы программа смогла выделить цветные пиксели. Без них программа неработоспособна.

1.2. Распознавание жестов при помощи сигналов тензодатчиков

Представителем этой группы являются перчатки GRASP – говорящие перчатки для глухонемых (Glove-base Recognition of Auslan using Simple Processing) [9].

Их принцип работы следующий: к компьютеру подключены две перчатки при помощи кабелей. Программа обрабатывает сигналы тензодатчиков и, в зависимости от информации, полученной с этих датчиков, выдаёт информацию о распознанном жесте. Внешний вид устройства приведён на рисунке 2.

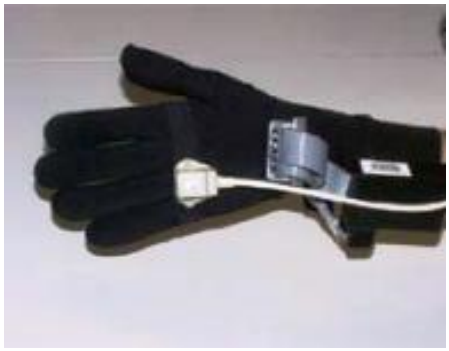


Рис. 2. Перчатка, при помощи которой программа распознаёт жесты.

Данный способ реализации так же имеет недостаток в необходимости надевания специальной перчатки, без которой программа не сможет распознать жест.

1.3. Автоматическое преобразование жестов русской ручной азбуки в текстовый вид

Программа [1], преобразующая жесты русской ручной азбуки в текстовый вид, работает непосредственно с веб-камерой, при помощи которой

определяется положение руки в кадре. После этого строится «скелет» человеческой руки, по которому программа распознаёт жест и переводит его в букву русской азбуки. Вид работы алгоритма приведён на рисунке 3.

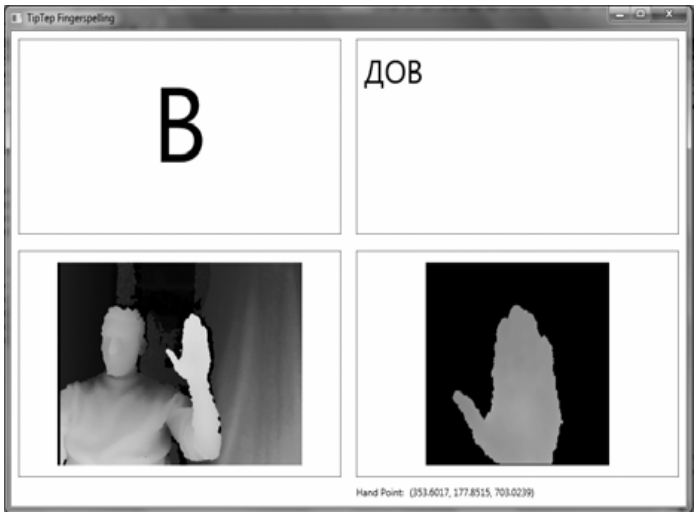


Рис. 3. Автоматическое преобразование жестов русской ручной азбуки в текстовый вид.

Эта программа является наиболее удобной в использовании, так как не требует наличия различных устройств, надетых на руку, но работает только с полутоновыми изображениями, то есть изображениями в оттенках серого. Такие изображения очень светочувствительны, что является большим недостатком этой программы.

1.4. Распознавание жестов при помощи нейронной сети

В последнее время нейронные сети получили мощный толчок в своём развитии. Одним из направлений их использования является обработка и извлечения информации из изображений.

Для распознавания жестов искусственная нейронная сеть могут использоваться для её обучения на заранее подготовленных и размеченных изображениях жеста, в таком случае используются свёрточные нейронные сети. Прежде чем начать распознавать жесты на изображении, необходимо локализовать его область, то есть детектировать кисть человека. Для детектирования кисти можно использовать подход Single Shot MultiBox детектора [17].

2. МЕТОДЫ И АЛГОРИТМЫ ДЕТЕКТИРОВАНИЯ И ОПИСАНИЯ КОНФИГУРАЦИИ КИСТИ БЕЗ ПРИМЕНЕНИЯ ТЕХНОЛОГИЙ ГЛУБОКОГО ОБУЧЕНИЯ

Человеко-машинное взаимодействие (Human-computer interaction - HCI) – это междисциплинарное научное направление, изучающее взаимодействие между людьми и машинами. Предметом HCI является изучения, планирование и разработка методов взаимодействия человека с машиной, где в роли машины может выступать персональный компьютер, компьютерная система больших масштабов, система управления процессами и т.д. [19]. Под взаимодействием понимается любая коммуникация между человеком и машиной. Одним из методов HCI, получившим широкое распространение в последние годы, является взаимодействие, основанное на жестах человека [20, 21].

Задачу распознавания жестов руки можно разделить на подзадачи:

1. Отделение кисти руки от остальной части изображения.
2. Построение контура кисти.
3. Нахождение ключевых точек на кисти.
4. Классификация жеста исходя из статического или динамического расположения точек.

Первая подзадача, а именно детектирование кисти человека в кадре является ключевой, поскольку от качества её решения зависит качество выполнения остальных подзадач. Рассмотрим первую подзадачу, а именно детектирование кисти человека в кадре.

2.1. Методы детектирования кисти человека без искусственных нейронных сетей

Существует множество решений задачи детектирования кисти человека. Наиболее популярными из них являются отделение фона изображения, распознавание цвета кожи в кадре и метод Оцу. Подробно рассмотрим каждое из них.

2.1.1. Отделение фона изображения

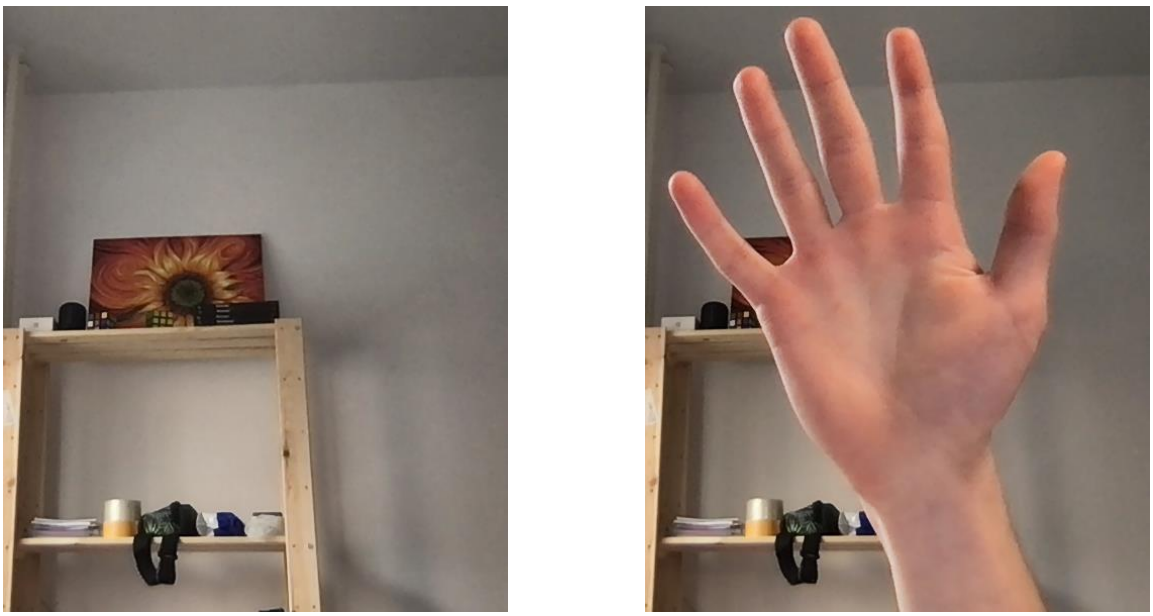
В данном решении принимается, что в кадре движется только рука, а остальные части тела, включая фон, остаются неподвижными. Таким образом, если вначале инициализировать фон как $bgr(x, y)$, а новое изображение с жестом рассматривать как $fgr(x, y)$, то изолированный жест можно принять как «разность» между этими изображениями:

$$gst_i = fgr_i(x, y) - bgr(x, y). \quad (1)$$

Полученный разностный жест переднего плана преобразуется в бинарное изображение, устанавливая соответствующий порог. Поскольку фон не является полностью статичным, например, если камера удерживается в руках оператора, то добавляется шумовая часть. Чтобы получить изображение руки без шумов, этот метод сочетается с распознаванием кожи человека. Чтобы удалить этот шум, применяется анализ связанных компонентов, чтобы заполнить пустоты, если применяется заливка какой-либо области, а также для получения чётких краёв применяется морфологическая обработка.

Недостаток данного решения состоит в том, что довольно сложно отделить фон, даже если он задан, поскольку не ясно какие из пикселей изменились, а какие остались прежними из-за тени, смещения фокуса, изменения экспозиции и т.д. Даже если зафиксировать все параметры камеры, то тень так или иначе испортит качество решения.

Приведём два примера отделения кисти от фона изображения. На рис. 4 изображены два изображения, одно из которых является инициализированным фоном, а второе – кисть на этом фоне.



а) б)

Рис. 4. Инициализированный фон (а) и изображение кисти на нём (б).

Для отделения кисти первым способом воспользуемся встроенной в библиотеку OpenCV функцией `absdiff()`.

Второй способ будет заключаться в следующем. Пусть даны два пикселя

$$p_1 = (r_1, g_1, b_1), \qquad p_2 = (r_2, g_2, b_2),$$

тогда различие между ними будет определяться как

$$D = \sqrt{(r_2 - r_1)^2 + (g_2 - g_1)^2 + (b_2 - b_1)^2}.$$

Затем создаётся битовая маска M, в которой значения определяются как

$$M_{(i,j)} = \begin{cases} 1, & D > Threshold \\ 0, & D \leq Threshold \end{cases}$$

Результаты работы первого и второго метода представлены на рис. 5 .

а) б)

Рис. 5. Результат первого метода (а) и второго (б).

Можно видеть, что оба метода не совсем точно отделяют нужный объект от фона, поэтому рассмотрим метод распознавания кожи в HSV и YCbCr цветовых моделях.

2.1.2. Метод распознавания кожи в HSV и YCbCr цветовых моделях

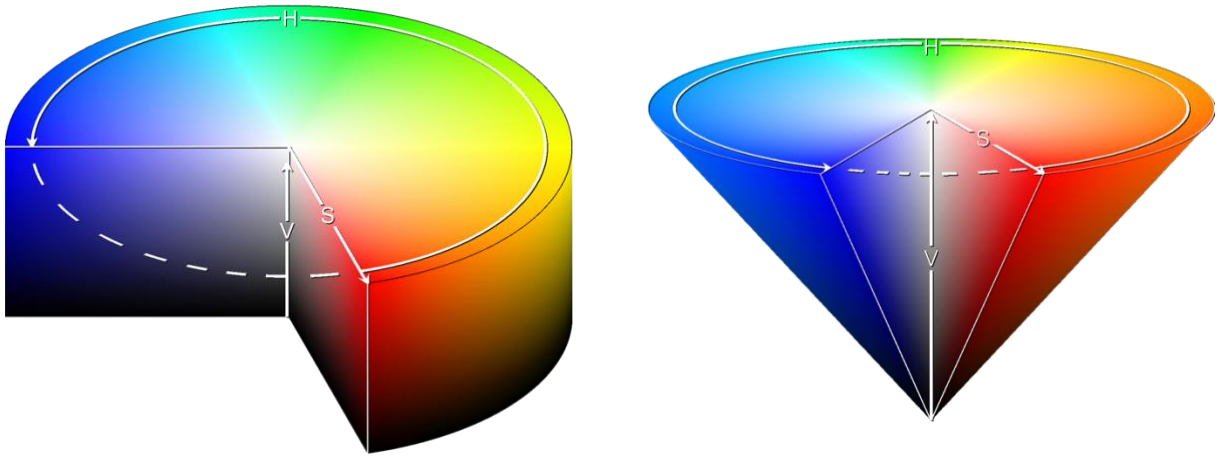
Для того, чтобы детектировать цвет кожи на изображениях очень часто применяются различные цветовых модели, а именно HSV и YCbCr.

HSV (*Hue*, *Saturation*, *Value*) или **HSB** (*Hue*, *Saturation*, *Brightness*) – цветовая модель, в которой координатами цвета являются:

1. **H**ue – цветовой тон (например, красный, зелёный или сине-голубой). Варьируется в пределах 0-360°, однако иногда приводится к диапазону 0-100 или 0-1.
2. **S**aturation – насыщенность. Варьируется в пределах 0-100 или 0-1. Чем больше этот параметр, тем «чище» цвет, поэтому иногда называют *чистотой* цвета.
3. **V**alue или **B**rightness – яркость. Так же задаётся в пределах 0-100 или 0-1.

Простейший способ отобразить HSV в трёхмерное пространство – воспользоваться цилиндрической системой координат (рис. 6а).

Здесь координата H определяется полярным углом, S – радиус-вектором, а V – Z-координатой. То есть, оттенок изменяется при движении вдоль окружности цилиндра, насыщенность – вдоль радиуса, а яркость – вдоль высоты. Несмотря на «математическую» точность, у такой модели есть существенный *недостаток*: на практике количество различимых глазом уровней насыщенности и оттенков уменьшается при приближении яркости (V) к нулю. Также на малых S и V появляются существенные ошибки округления при переводе RGB в HSV и наоборот. Поэтому чаще применяется коническая модель (рис. 6б).



а) б)

Рис. 6. HSV в цилиндрической (а) и конической (б) системе координат.

Как и в цилиндре, в конической модели оттенок изменяется по окружности конуса. Насыщенность цвета возрастает с отдалением от оси конуса, а яркость – с приближением к его основанию. Иногда вместо конуса используется шестиугольная правильная пирамида.

Для перевода RGB в HSV будем считать, что $H \in [0,360]$, $S, V, R, G, B \in [0,1]$.

Пусть $MAX = \max(R, G, B)$, а $MIN = \min(R, G, B)$. Тогда

$$H = \begin{cases} 60 \cdot \dfrac{G-B}{MAX-MIN} + 0, & MAX = R, \quad G \geq B \\ 60 \cdot \dfrac{G-B}{MAX-MIN} + 360, & MAX = R, \quad G < B \\ 60 \cdot \dfrac{B-R}{MAX-MIN} + 120, & MAX = G \\ 60 \cdot \dfrac{R-G}{MAX-MIN} + 240, & MAX = B \end{cases} \quad (2)$$

$$S = \begin{cases} 0, & MAX = 0 \\ 1 - \dfrac{MIN}{MAX} \end{cases} \quad (3)$$

$$V = MAX \quad (4)$$

Можно заметить, что уравнение (2) не определено, когда $MAX = MIN$, поэтому существуют другие уравнения:

$$H = \arccos \frac{0.5 \cdot \big((R-G) + R - B\big)}{\sqrt{(R-G)^2 + (R-B) + (G-B)}}$$

$$S = 1 - 3\frac{\min(R,G,B)}{R+G+B} \quad (5)$$

$$V = \frac{1}{3}(R+G+B)$$

Для перевода HSV в RGB будем считать, что $H \in [0,360], S \in [0,100]$ и $V \in [0,100]$. Если

$$H_i = \left\lfloor \frac{H}{60} \right\rfloor \bmod 6,$$

$$V_{min} = \frac{(100-S)V}{100},$$

$$a = (V - V_{min})\frac{H \bmod 6}{60},$$

$$V_{inc} = V_{min} + a,$$

$$V_{dec} = V - a,$$

тогда по таблице 1 выбираем нужные значения. Полученные значения красного, зелёного и синего каналов RGB исчисляются в процентах. Чтобы привести их в соответствие распространённому представлению COLORREF необходимо умножить каждое из них на $\frac{255}{100}$. При целочисленном кодировании для каждого

цвета в HSV есть соответствующий цвет в RGB. Однако обратное утверждение не является верным: некоторые цвета в RGB нельзя выразить в HSV так, чтобы значение каждого компонента было целым. Фактически, при таком кодировании доступна только $\frac{1}{256}$ часть цветового пространства RGB.

Таблица 1.

Таблица перевода значений в RGB из HSV.

H_i	R	G	B
0	V	V_{inc}	V_{min}
1	V_{dec}	V	V_{min}
2	V_{min}	V	V_{inc}
3	V_{min}	V_{dec}	V
4	V_{inc}	V_{min}	V
5	V	V_{min}	V_{dec}

Следующим рассмотрим цветовую модель YCbCr. Известно, что органы зрения человека менее чувствительны к цвету предметов, чем к их яркости. В цветовом пространстве RGB все три цвета считаются одинаково важными, и они обычно сохраняются с одинаковым разрешением. Однако можно отобразить цветное изображение более эффективно, отделив светимости от цветовой информации и представив её с большим разрешением, чем цвет. Цветовое пространство YC_BC_R и его вариации является популярным методом эффективного представления цветных изображений.

$YC_BC_R -$ семейство цветовых пространств, которые используются для передачи цветных изображений в компонентном видео и цифровой фотографии. Является частью рекомендации МСЭ-R BT.601 при разработке стандарта видео Всемирной цифровой организации и фактически является масштабированной и смещённой копией YUV.

Y $-$ компонента яркости, $C_B -$ синий компонент цветности, $C_R -$ красный компонент цветности.

Преобразование RGB в YC_BC_R можно описать следующей формулой:

$$\begin{pmatrix} Y \\ C_B \\ C_R \end{pmatrix} = \begin{pmatrix} 16 \\ 128 \\ 128 \end{pmatrix} + \begin{pmatrix} 65.481 & 128.553 & 24.996 \\ -37.797 & -74.203 & 112 \\ 112 & -93.786 & -18.214 \end{pmatrix} \cdot \begin{pmatrix} R \\ G \\ B \end{pmatrix} \quad (6)$$

Для того чтобы отделить кожу от остальной части изображения используют определённые диапазоны составляющих цветовых моделей, которые находится эмпирически или итеративно. В первом случае путём проб и ошибок подбираются значения составляющих. Можно заметить, что при данном подходе кожа будет иметь разные цветовые диапазоны при разном освещении. Покажем это. Зададим диапазон для изображения в HSV цветовой модели как

$$0 \leq H \leq 200,$$

$$15 \leq S \leq 255,$$

$$80 \leq V \leq 255,$$

и, получаем результат, представленный на рис. 7.

а) б)

Рис. 7. Изображения с удалённым фоном в HSV.

Можно видеть, что на первом изображении фон отделился практически идеально, но на втором видны фрагменты фона.

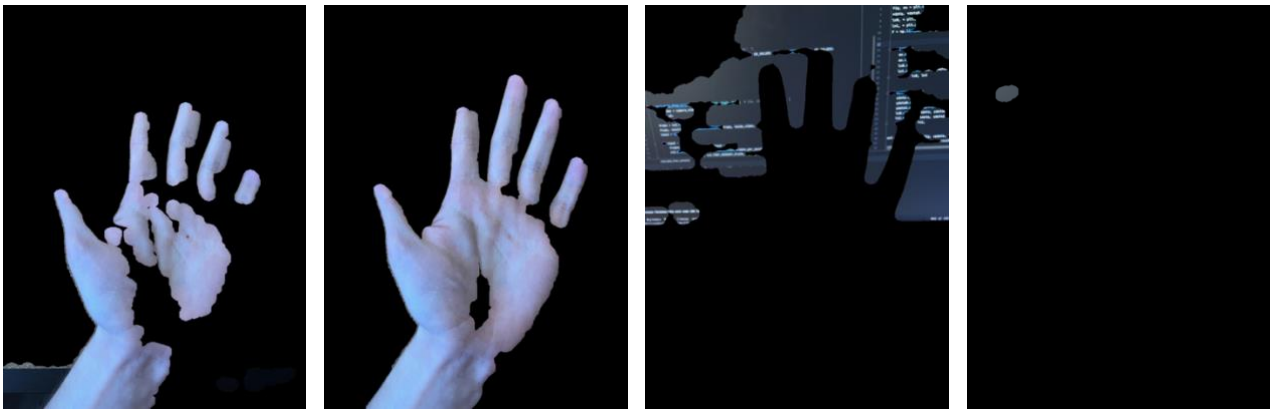
Для цветовой модели YCbCr в статье [22] было предложено два варианта диапазона компонент (7) и (8):

$$80 < Y \leq 255,$$

$$85 < C_b < 135, \quad (7)$$

$$135 < C_r < 180$$
$$Y \in \forall,$$
$$77 \leq C_b \leq 127, \quad (8)$$
$$133 \leq C_r \leq 173$$

Сравнение двух вариантов представлено на рисунке 8.



а) б)

Рис. 8. Изображения с удалённым фоном в YCbCr.

Легко видеть, что оба диапазона работают не идеально, но первый показал себя намного лучше.

Таким образом, можно сделать вывод, что для детектирования цвета кожи лучше использовать изображение в цветовой модели HSV.

Рассмотрим следующий метод отделения фона изображения, а именно метод Оцу.

2.1.3. Метод Оцу

В 1979 году Нобуюки Оцу опубликовал статью [23] метода порогового разделения, основываясь на гистограмме серых цветов изображения. Метод Оцу – это алгоритм вычисления порога бинаризации для полутонового изображения, используемый в области компьютерного распознавания образов и обработки изображений для получения чёрно-белых изображений. Алгоритм позволяет разделить пиксели двух классов («полезные» и «фоновые»), рассчитывая такой порог, чтобы внутриклассовая дисперсия была минимальной.

Пусть пиксели изображения представлены в L уровнях серого $[1,2,\dots,L]$. Количество пикселей уровня i обозначим как n_i, а количество всех пикселей как

$$N = n_1 + n_2 + \cdots + n_L.$$

Для того чтобы упростить алгоритм гистограмма монохромного изображений нормализована и рассматривается как распределение вероятностей:

$$p_i = \frac{n_i}{N}, \qquad p_i \geq 0, \qquad \sum_{i=1}^{L} p_i = 1.$$

Теперь предположим, что пиксели изображения разделены на два класса C_0 и C_1 (фон и объект) по пороговому значению k. C_0 обозначает пиксели с уровнями $[1,\dots,k]$, а $C_1 -$ пиксели с уровнями $[k+1,\dots,L]$. Следовательно, вероятности класса и *средний уровень*, соответственно, задаются как

$$\omega_0 = \sum_{i=1}^{k} p_i = \omega(k), \qquad \omega_1 = \sum_{i=k+1}^{L} p_i = 1 - \omega(k)$$

и

$$\mu_0 = \sum_{i=1}^{k} \frac{ip_i}{\omega_0} = \frac{\mu(k)}{\omega(k)}, \qquad \mu_1 = \sum_{i=k+1}^{L} \frac{ip_i}{\omega_1} = \frac{\mu_T - \mu(k)}{1 - \omega(k)},$$

где

$$\omega(k) = \sum_{i=1}^{k} p_i\,, \qquad \mu(k) = \sum_{i=1}^{k} ip_i\,, \qquad \mu_T = \mu(L) = \sum_{i=1}^{L} ip_i.$$

$\omega(k)$ и $\mu(k)$ являются кумулятивными моментами нулевого и первого порядка соответственно, а μ_T является общим средним уровнем исходного изображения. Легко можно доказать, что следующие уравнения справедливы для любого k:

$$\omega_0\mu_0 + \omega_1\mu_1 = \mu_T, \qquad \omega_0 + \omega_1 = 1.$$

Дисперсии классов определяются как

$$\sigma_0^2 = \sum_{i}^{k} \frac{(i-\mu_0)^2 p_i}{\omega_0}, \qquad \sigma_1^2 = \sum_{i=k+1}^{L} \frac{(i-\mu_1)^2 p_i}{\omega_1}.$$

Оцу показал, что минимизация дисперсии *внутри* класса равносильна максимизации дисперсии *между* классами:

$$\sigma_W^2 = \omega_0\sigma_0^2 + \omega_1\sigma_1^2,$$

$$\sigma_B^2 = \omega_0(\mu_0 - \mu_T)^2 + \omega_1(\mu_1 - \mu_T)^2 = \omega_0\omega_1(\mu_1 - \mu_0)^2,$$

$$\sigma_T^2 = \sum_{i=1}^{L}(i - \mu_T)^2 p_i$$

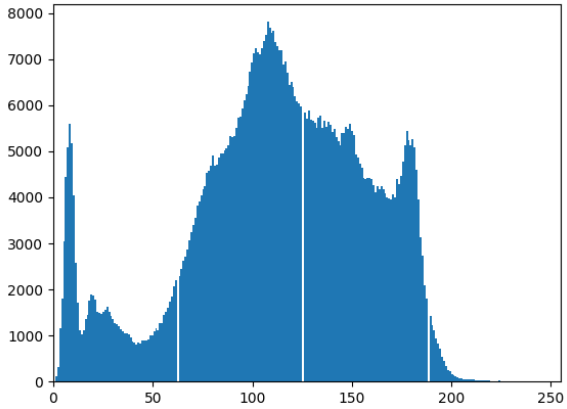
$\sigma_W, \sigma_B, \sigma_T -$ внутриклассовая дисперсия, дисперсия между классами и общая дисперсия всех уровней соответственно. Таким образом, алгоритм можно записать как:

Пусть дано монохромное изображение $G(i,j), i = \overline{1, Heıght}, j = \overline{1, Wıdth}$. Счётчик повторений $k = 0$.

1. Вычислить гистограмму $p(l)$ изображения и частоту $N(l)$ для каждого уровня интенсивности изображения G.
2. Вычислить начальные значения для $\omega_0(0), \omega_1(0)$ и $\mu_1(0), \mu_2(0)$.
3. Для каждого значения $t = \overline{1, \max(G)} -$ полутона – горизонтальная ось гистограммы:
   - Обновляем ω_0, ω_1 и μ_0, μ_1.
   - Вычисляем $\sigma_B^2(t) = \omega_0(t)\omega_1(t)[\mu_0(t) - \mu_1(t)]^2$.
   - Если $\sigma_B^2(t)$ больше, чем имеющееся, то запоминаем σ_B^2 и значение порога t.
4. Искомый порог соответствует максимуму $\sigma_B^2(t)$.

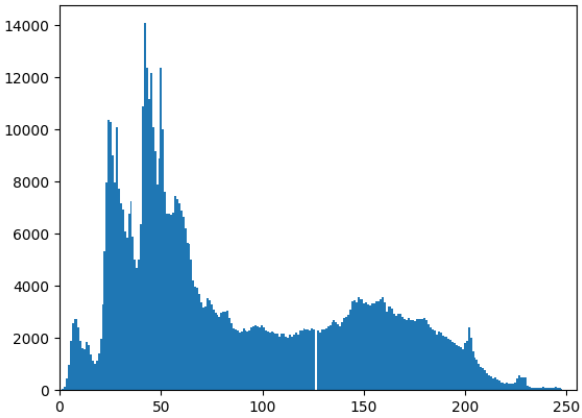
На рисунке 9 представлены примеры изображений с их гистограммами. По гистограммам можно выдвинуть гипотезу, что алгоритм Оцу на примерах сработает не лучшим образом, поскольку её нельзя чётко разделить какой-то одной единственной пороговой величиной. Доказательство этой гипотезы изображено на рис. 10.

а)



б)

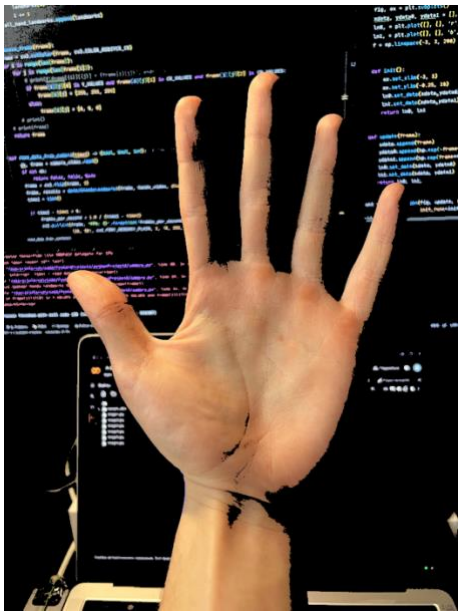
в)



г)

Рис. 9. Изображения 1 и 2 и их гистограммы.

а)



б)

Рис. 10. Результат работы программы с алгоритмом Оцу.

2.1.4. Смесь гауссианов

Стандартным подходом к построению модели фона, использующимся для многих прикладных задач, является смесь гауссовых распределений (Mixture of Gaussians, MOG) [24, 25]. Чаще всего для каждого пикселя текущего кадра с номером n строится функция плотности вероятности $P_n = P\big(\nu_n(p)\big)$, и в MOG используется именно этот подход. Предполагается, что для каждого пикселя текущего изображения она может быть представлена смесью нормальных распределений, где $G -$ их число в смеси. Обозначим за ω_i^n вес распределения Гаусса с номером i, $E_i^n -$ его математическое ожидание и $\sum_i^n$ – среднеквадратичное отклонение. Тогда для P_n получим:

$$P_n = \sum_{i=1}^{G} \omega_i^n \cdot N(\nu_n(p)|E_i^n, \sum_i^n).$$

Здесь в общем случае $N(\nu_n(p)|E_i^n, \sum_i^n) -$ многомерное нормальное распределение, имеющее вид:

$$\mathrm{N}(\nu_n(p)|E_i^n, \sum_i^n) = \frac{1}{(2\pi)^{H/2}\left|\sum_i^n\right|^{1/2}} \exp\left[-\frac{1}{2}(\nu_n(p) - E_i^n)^T \cdot \left(\sum_i^n\right)^{-1} \cdot (\nu_n(p) - E_i^n)\right] \quad (9)$$

где:

H – число компонентов цвета,

$\sum$ – матрица ковариации.

Для ускорения вычислений, чтобы не вычислять обратную матрицу в формуле (9), в случае цветного изображения предполагается, что для компонентов цвета соблюдается их независимость, а также что они имеют одинаковое стандартное отклонение. Тогда матрица ковариации примет вид:

$$\sum_{i}^{n} = (\sigma_i^n) \cdot E, \tag{10}$$

где:

E – единичная матрица размерности H.

После того, как для всех пикселей получена смесь, для каждой из них производится сортировка распределений по убыванию величины $\frac{\omega_i^n}{\sigma_i^n}$. Это делается для того, чтобы фоновые пиксели отвечали распределению с маленькой дисперсией и большим весом. Тогда число J первых гауссиан, соответствующих распределению цвета фона для пикселя p, будет определяться из условия:

$$J = \arg\min_{\text{a}} \left(\sum_{i}^{a} \omega_i^n > A \right), \tag{11}$$

где:

A – некоторый порог.

Решающее правило для текущего кадра выглядит следующим образом: для каждого пикселя определяют, какому из распределений смеси принадлежит его значение $\nu_{n+1}(p)$, используя расстояние Махаланобиса:

$$\sqrt{\left(\nu_{n+1}(p) - E_i^n\right)^T \cdot \left(\sum_{i}^{n}\right)^{-1} \cdot \left(\nu_{n_1}(p) - E_i^n\right)} < 2.5 \cdot \sigma_i^n.$$

После этого возможно два случая:

1. Распределение нашлось. Пиксель будет отнесён к фону, если эта гауссиана является одной из J первых. В противном случае он будет отнесён к движущемуся объекту.
2. Если они одной гауссианы не нашлось, то пиксель также относят к движущемуся объекту.

Теперь рассмотрим, как для пикселя p со значением $\nu_{n+1}(p)$ происходит обновление параметров его модели. Как и при принятии решения, относится ли пиксель к фону или нет, здесь также можно быть два случая. Не существует

единого понимания того, каким образом следует вычислять новые параметры, поэтому рассмотрим здесь один из наиболее популярных:

1. Распределение нашлось. Происходит обновление его параметров и веса. Если нашлось более одного распределения, данных процесс выполняется для каждого из них:

$$\omega_i^{n+1} = (1-\alpha)\cdot\omega_i^n + \alpha,$$

$$E_i^{n+1} = (1-g)\cdot E_i^n + g\cdot \nu_{n+1}(p),$$

$$\left(\sigma_i^{n+1}\right)^2 = (1-g)\cdot(\sigma_i^n)^2 + g\cdot\left(\nu_{n+1}(p) - E_i^{n+1}\right)$$

$$\cdot\left(\nu_{n+1}(p) - E_i^{n+1}\right)^T,$$

$$g = \alpha\cdot N\left(\nu_n(p)|E_i^n, \sum_i^n\right).$$

   В этих формулах $\alpha -$ заданная константа.

   Для всех остальных распределений в смеси пересчитываются только веса:

$$\omega_i^{n+1} = (1-\alpha)\cdot\omega_i^n.$$

2. Распределение в смеси не нашлось. Тогда самую последнюю в уже отсортированной смеси гауссиану заменяют новой: $E_G^{n+1} = \nu_{n+1}(p)$, дисперсия – максимально возможная, а вес – минимально допустимый.

Для инициализации гауссиан для каждого пикселя чаще всего применяют либо EM-алгоритм (Expectation-maximization algorithm), либо k-means, что достаточно затратно в вычислительном плане. Число входящих в смесь распределений G обычно принимают равным от 3 до 5. Также существует подход, позволяющий автоматически подбирать необходимое количество гауссиан [25].

В библиотеке OpenCV данный метод реализуется с помощью функции `createBackgroundSubtractorMOG2()`. Данная функция обучается на изображениях каждого кадра и лучше всего работает в режиме последовательности, то есть на видео. Результат работы метода MOG представлен на рисунке 11.

Рис. 11. Результат отделения фона от изображения с помощью метода MoG.

Можно сделать вывод, что этот метод наилучшим образом работает при какой-либо последовательности кадров и статическом фоне, поскольку «запоминает» фоновые пиксели и вычитает их при последующих итерациях алгоритма.

2.2. Методы построения контура

Отслеживание границ – один из основных методов обработки оцифрованных двоичных изображений, производящий последовательность координат или цепных кодов от границ между связным компонентом.

2.2.1. Выделение границ с помощью оператора Кэнни

Оператор Кэнни (детектор границ Кэнни или алгоритм Кэнни) – оператор обнаружения границ изображения. Кэнни изучил математическую проблему получения фильтра, оптимального по критериям выделения, локализации и минимизации нескольких откликов одного края. Он показал, что искомый фильтр является суммой четырёх экспонент. Он также показал, что этот фильтр может быть хорошо приближен первой производной гауссианы. Кэнни ввёл понятие *подавление немаксимумов* (Non-Maximum Suppression), которое

означает, что пикселями границ объявляются пиксели, в которых достигается локальный максимум градиента в направлении вектора градиента.

Целью Кэнни было разработать оптимальный алгоритм обнаружения границ, удовлетворяющий трём критериям:

- Хорошее обнаружение (повышение отношения сигнал/шум);
- Хорошее локализация (правильное определения положения границы);
- Единственный отклик на одну границу.

Затем из этих критериев строится целевая функция стоимости ошибок, минимизацией которой находится «оптимальный» линейный оператор для свёртки с изображением.

Первым этапом алгоритма является *сглаживание*, то есть размытие изображения для удаления шума. Оператор Кэнни использует фильтр, который может быть хорошо приближен к первой производной гауссианы. Для $\sigma = 1.4$:

$$B = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} \cdot A.$$

Следующим этапом является *поиск градиентов*. Границы отмечаются там, где градиент изображения приобретает максимальное значение и могут иметь различное направление, поэтому алгоритм Кэнни использует четыре фильтра для обнаружения горизонтальных, вертикальных и диагональных рёбер в размытом изображении.

$$G = \sqrt{G_x^2 + G_y^2}, \qquad \Theta = \arctan\left(\frac{G_y}{G_x}\right).$$

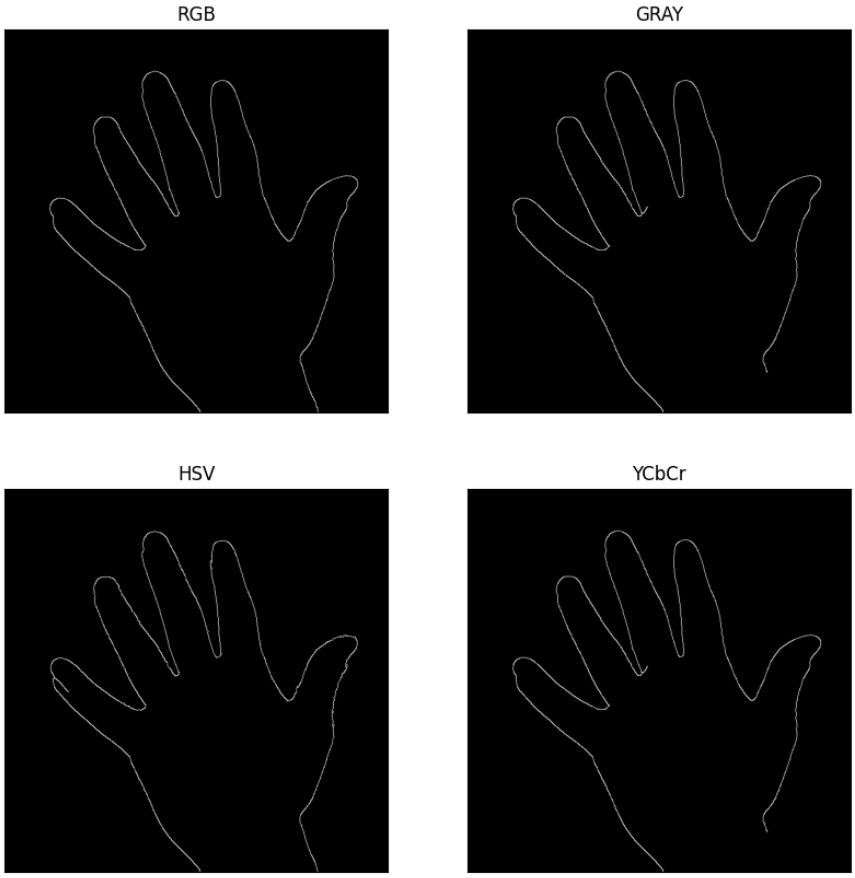
Угол направления вектора градиента при этом округляется и может принимать такие значения: $0°, 45°, 90°, 135°$.

Затем происходит *подавление немаксимумов*, когда только локальные максимумы отмечаются как границы, *двойная пороговая фильтрация* – потенциальные границы определяются порогами и *трассировка области*

неоднозначности, когда итоговые границы определяются путём подавления всех краёв, не связанных с определёнными (сильными) границами.

Перед применением детектора обычно преобразует изображение в оттенки серого, чтобы уменьшить вычислительные затраты. Примеры работы оператора Кэнни в цветовых моделях RGB, GRAY, HSV и YCbCr показаны на рис. 12.

а)



б)

Рис. 12. Примеры работы оператора Кэнни.

Можно видеть, что алгоритм Кэнни довольно неплохо позволяет определить границы объекта. Но для задачи извлечения конфигурации кисти необходимо извлекать особые точки изображения, а этот метод позволяет лишь выделить границы на изображении, не определяя сам контур. В связи с этим следует рассмотреть следующий метод – топологический структурный анализ цифрового бинарного изображения с помощью отслеживания границ.

2.2.2. Топологический структурный анализ изображения с помощью отслеживания границ

Этот метод был разработан Сатоши Сузуки и Кейчи Эйбом в 1985 году [26]. Алгоритм предполагает нахождение контуров с учётом вложенности, то есть способен определить, когда в контур одного объекта вложен другой. Реализация данного алгоритма лежит в основе функции `findContours()` библиотеки OpenCV.

Для отрисовки контуров, полученных с помощью данной функции, можно воспользоваться функцией `drawContours()`.

Поскольку функция `findContours()` находит все существующие контуры на изображении, то среди них нужно выделить один единственный, предположительно определяющий кисть руки. Допустим, что кисть *занимает наибольшее пространство* на изображении и, соответственно, имеет контур наибольшей площади. Тогда среди всех контуров следует выбрать тот, площадь которого имеет наибольшее значение. Площадь контура можно найти с помощью *формулы площади Гаусса*.

Формула площади Гаусса (формула землемера или формула шнурования или алгоритм шнурования) – формула определения площади простого многоугольника, вершины которого заданы декартовыми координатами на плоскости. В формуле векторным произведением координат и сложением определяется площадь области, охватывающей многоугольник, а затем из неё вычитается площадь окружающего многоугольника, что даёт площадь внутреннего многоугольника.

Формула может быть представлена следующим выражением:

$$S = \frac{1}{2}\left|\sum_{i=1}^{n-1} x_i y_{i+1} + x_n y_1 - \sum_{i=1}^{n-1} x_{i+1} y_i - x_1 y_n\right| = \\ = \frac{1}{2}|x_1 y_2 + x_2 y_3 + \cdots + x_{n-1} y_n + x_n y_1 - x_2 y_1 - x_3 y_2 - \cdots - x_n y_{n-1} \\ - x_1 y_n|, \tag{12}$$

где:

S – площадь многоугольника,

n – количество сторон многоугольника,

(x_i, y_i), $i = \overline{1,n}$ – координаты вершин многоугольника.

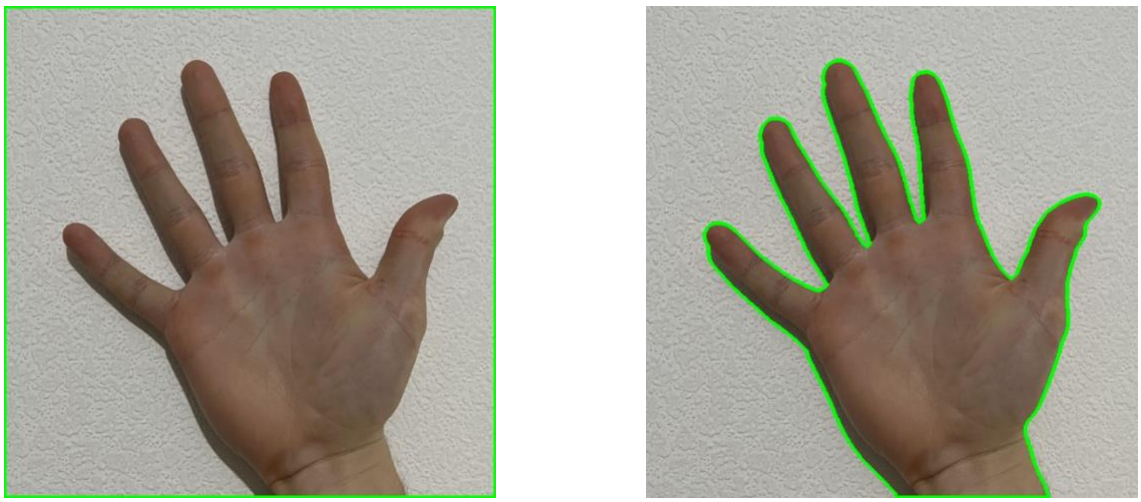
Другие представления этой же формулы:

$$S = \frac{1}{2}\left|\sum_{i=1}^{n} x_i(y_{i+1} - y_{i-1})\right| = \frac{1}{2}\left|\sum_{i=1}^{n} y_i(x_{i+1} - x_{i-1})\right| = \\ = \frac{1}{2}\left|\sum_{i=1}^{n} x_i y_{i+1} - x_{i+1} y_i\right| = \frac{1}{2}\left|\sum_{i=1}^{n} \det\begin{pmatrix} x_i & y_i \\ x_{i+1} & y_{i+1} \end{pmatrix}\right|, \quad (13)$$

где:

$$x_{n+1} = x_1, \qquad x_0 = x_n,$$
$$y_{n+1} = y_1, \qquad y_0 = y_n.$$

Рассмотрим пример использования данных функций. Сравним результат поиска контуров с предварительной обработкой изображения, заключающейся в отделении объекта от фона, и без обработки (только с переводом изображения в чёрно-белый формат) и получим результаты, представленные на рис. 13.



а) б)

Рис. 13. Пример поиска контура на изображении без предварительной обработки (а) и с предварительной обработкой с помощью метода Оцу (б).

Можно видеть, что пороговая бинаризация изображения методом Оцу позволила с высокой точностью определить истинное расположение контура ладони. Это означает, что для дальнейшего исследования контуров необходима предобработка изображения.

2.3. Методы поиска ключевых точек на кисти с помощью выпуклой оболочки

Особую сложность представляет задача определения положения ключевых точек на кисти человека. Эту нетривиальную задачу можно решить с помощью определения дефектов выпуклости на выпуклой оболочке точек кисти руки.

В предыдущем разделе был рассмотрен метод построения контура кисти с помощью топологического структурного анализа цифрового бинарного изображения с помощью отслеживания границ. Этот метод хорош тем, что позволяет определить точные координаты точек контура. С помощью этих точек существует возможность построить *выпуклую оболочку* или их *наименьшее множество*.

Выпуклой оболочкой множества X называется наименьшее выпуклое множество, содержащее X. «Наименьшее множество» здесь означает наименьший элемент по отношению к вложению множеств, то есть такое выпуклое множество, содержащее данную фигуру, что оно содержится в любом другом выпуклом множестве, содержащем данную фигуру.

Рассмотрим основные алгоритмы построения выпуклой оболочки.

2.3.1. Алгоритм Грэхема

Алгоритм Грэхема [27] – алгоритм построения выпуклой оболочки в двумерном пространстве. В этом алгоритме задача о выпуклой оболочке решается с помощью стека, сформированного из точек-кандидатов. Все точки входного множества заносятся в стек, а затем точки, не являющиеся вершинами выпуклой оболочки, со временем удаляются из него. По завершении работы алгоритма в стеке остаются только вершины оболочки в порядке их обхода против часовой стрелки.

Временная сложность алгоритма $= O(n \log n)$.

Описание алгоритма:

Пусть точки $p = [p_0, \dots, p_{n-1}] -$ изначальный массив точек.

1. Найти самую «нижнюю» точку в массиве (ту, в которой наименьшая среди всех координата y). Если таких точек несколько, то среди них следует выбрать точку с наименьшей координатой x. Найденная точка P_0 является первой точкой выпуклой оболочки;
2. Перебрать остальные $n - 1$ точек и отсортировать их по полярному углу относительно P_0 в направлении против часовой стрелки. Если полярный угол нескольких точек одинаков, то выбрать ближайшую к P_0;
3. После сортировки, проверить, есть ли точки с одинаковым полярным углом. Если да, то удалить все эти точки, кроме ближайшей к P_0. Положить размер нового массива равным m;

4. Если $m < 3$, то алгоритм прерывается. Выпуклую оболочку определить невозможно;
5. Создать пустой стек S и положить в него первые три точки нового массива p_0, p_1, p_2;
6. Для каждой из оставшихся $m - 3$ точек:
   – Удалять точки из стека пока ориентация трёх следующих точек не против часовой стрелки (они не совершают левый поворот): точка наверху стека, точка следующая после неё и текущая точка p_i;
   – Добавляем точку p_i в S;
7. Стек S содержит точки выпуклой оболочки.

Реализация алгоритма Грэхема на языке Python находится в приложении А. Протестировав алгоритм на случайном наборе точек, получаем результат, представленный на рисунке 14.

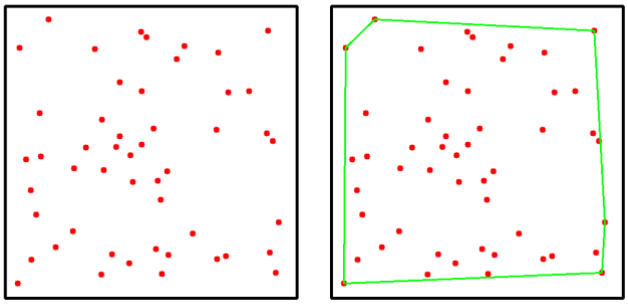


Рис. 14. Сгенерированные точки (слева) и их выпуклая оболочка, найденная с помощью алгоритма Грэхема (справа).

На рисунке видно, что алгоритм верно определил выпуклое множество сгенерированных точек.

2.3.2. Алгоритм Джарвиса

Алгоритм Джарвиса [28] (или алгоритм обхода Джарвиса, или алгоритм заворачивания подарка) определяет последовательность элементов множества, образующих выпуклую оболочку для этого множества. Метод можно представить как обтягивание верёвкой множества вбитых в доску гвоздей.

Алгоритм работает за время $O(nh)$, где $n -$ общее числа точек на плоскости, а $h -$ число точек в выпуклой оболочке. В худшем случае – $O(n^2)$, когда все точки попадают в выпуклую оболочку.

Описание алгоритма:

Пусть дано множество точек $P = \{p_1, p_2, \dots, p_n\}$. В качестве начальной берётся самая левая нижняя точка p_1 (её можно найти за $O(n)$ обычным проходом по всем точкам), эта точка точно является одной из вершин выпуклой оболочки. Следующая точка (p_2) берётся та, что имеет наименьший положительный полярный угол относительно точки p_1 как начала координат. После этого для каждой точки p_i $(2 < i \leq |P|)$ против часовой стрелки производится поиск такой точки p_{i+1}, в которой будет образовываться наибольший угол между прямыми $p_{i-1}p_i$ и p_ip_{i+1}. Она и будет следующей вершиной выпуклой оболочки. При этом нет необходимости находить сам угол – можно вычислять только его косинус через скалярное произведение между лучами $p_{i-1}p_i$ и $p_ip'_{i+1}$, где $p_i -$ последний найденный минимум, $p_{i-1} -$ предыдущий минимум, а $p'_{i+1} -$ претендент на следующий минимум. Новым минимумом будет та точка, в которой косинус будет принимать наименьшее значение. Поиск вершин выпуклой оболочки продолжается до тех пор, пока $p_{i+1} \neq p_1$. В тот момент, когда следующая точка в выпуклой оболочке совпала с первой, алгоритм останавливается – выпуклая оболочка построена (рис. 15).

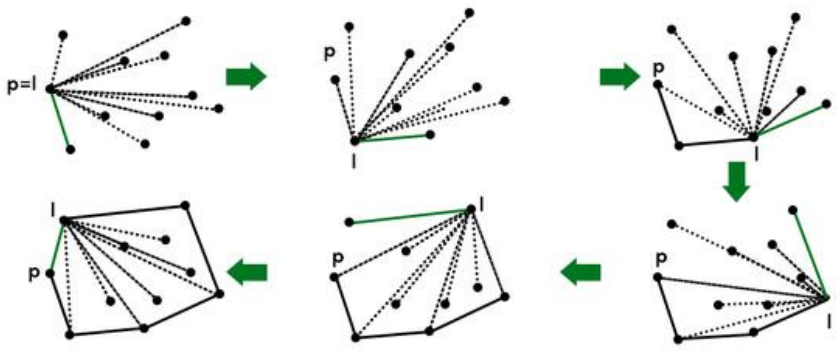


Рис. 15. Пример работы алгоритма Джарвиса.

Реализация алгоритма Джарвиса на языке Python находится в приложении Б. Так же протестируем алгоритм Джарвиса на случайном наборе точек и получим результат, представленный на рисунке 16.

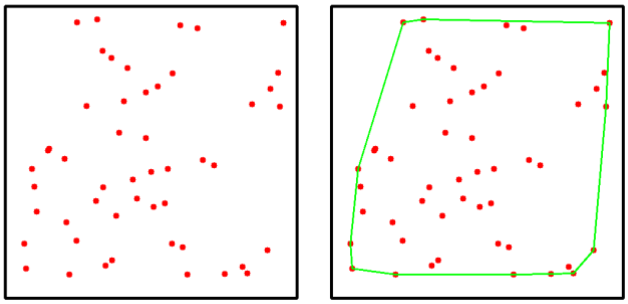


Рис. 16. Сгенерированные точки (слева) и их выпуклая оболочка, найденная с помощью алгоритма Джарвиса (справа).

Можно видеть, что алгоритм отработал верно, но у него есть существенный недостаток – он намного медленнее по сравнению с алгоритмом Грэхема.

2.3.3. Алгоритм Киркпатрика

Алгоритм Киркпатрика [29] заключается в построении выпуклой оболочки методом «разделяй и властвуй».

Описание алгоритма:

Дано множество S, состоящее из N точек.

1. Если $N \leq N_0$ (N_0 – некоторое небольшое целое число), то построить выпуклую оболочку одним из известных методов и остановиться, иначе перейти к шагу 2.
2. Разбить исходное множество S произвольным образом на два примерно равных по мощности подмножества S_1 и S_2 (пусть S_1 содержит $N/2$ точек, а S_2 содержит $(N - N/2)$ точек).
3. Рекурсивно найти выпуклые оболочки каждого из подмножеств S_1 и S_2.

4. Построить выпуклую оболочку исходного множества как выпуклую оболочку объединения двух выпуклых многоугольников $CH(S_1)$ и $CH(S_2)$.

Поскольку $CH(S) = CH(S_1 \cup S_2) = CH(CH(S_1) \cup CH(S_2))$, сложность этого алгоритма является решением рекурсивного соотношения $T(N) \leq 2T(N/2) + f(N)$, где $f(N) -$ время построения выпуклой оболочки объединения двух выпуклых многоугольников, каждый из которых имеет около $N/2$ вершин. Таким образом, временная сложность этого алгоритма равна $O(N \log N)$.

Реализация алгоритма Киркпатрика на языке Python находится в приложении В. Результат работы показан на рисунке 17.

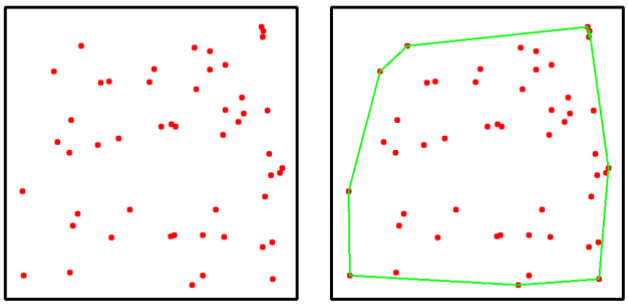


Рис. 17. Сгенерированные точки (слева) и их выпуклая оболочка, найденная с помощью алгоритма Киркпатрика (справа).

Так же видно, что алгоритм отработал верно. Проведём сравнение этих трёх алгоритмов.

2.3.4. Сравнительный анализ алгоритмов

Для того чтобы выбрать наиболее быстрый алгоритм, проведём их сравнение по времени работы на 10000 тестовых случайно сгенерированных множеств 1000 точек.

Результаты работы сведены в таблицу 2.

Таблица 2.

Сравнение алгоритмов построения выпуклой оболочки.

Алгоритм	Максимальное время, мс	Минимальное время, мс	Среднее время, мс
Грэхема	183	7	8,7976
Джарвиса	239	6	12,7415
Киркпатрика	180	11	25,3169

Можно видеть, что среди всех у алгоритма Киркпатрика самое маленькое максимальное время, но при этом самое большое минимальное и среднее. А самым эффективным алгоритмом в среднем в данном случае оказался алгоритм Грэхема.

2.3.5. Использование алгоритма Грэхема для поиска ключевых точек

Результат построения контура кисти руки и выпуклой оболочки этого контура изображён на рисунке 18.

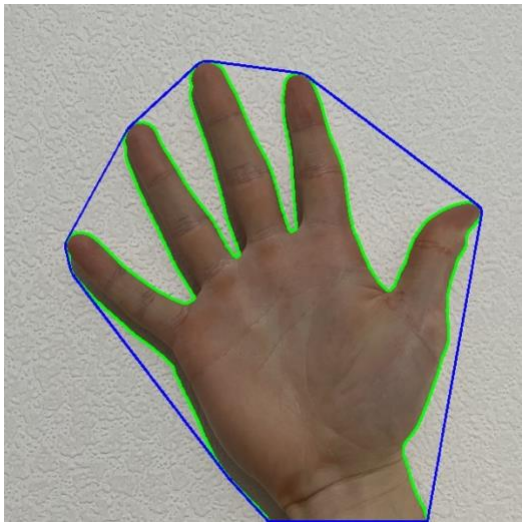


Рис. 18. Выделение контура и выпуклой его оболочки на кисти.

Для определения необходимых ключевых точек на кисти необходимо найти *дефекты выпуклости* контура и выпуклой оболочки.

Дефект выпуклости в данном случае – это максимальное расстояние между выпуклой оболочкой и контуром (рис. 19).

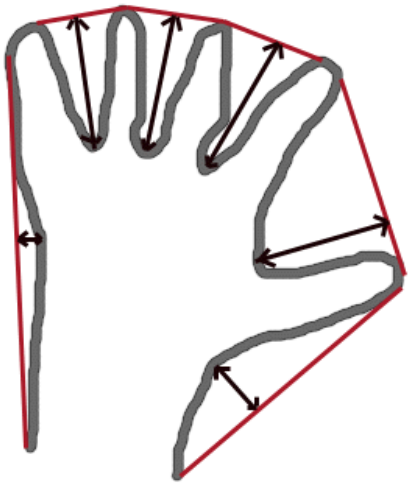


Рис. 19. Пример дефектов выпуклости.

Результат расчёта дефектов выпуклости представлен на рисунке 20.

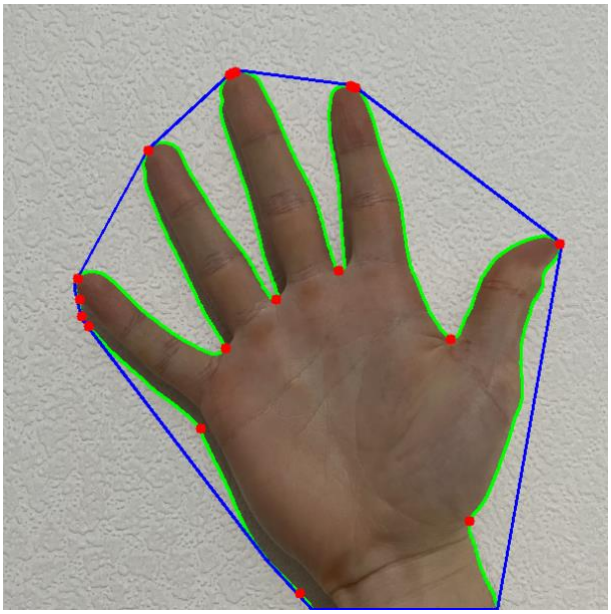


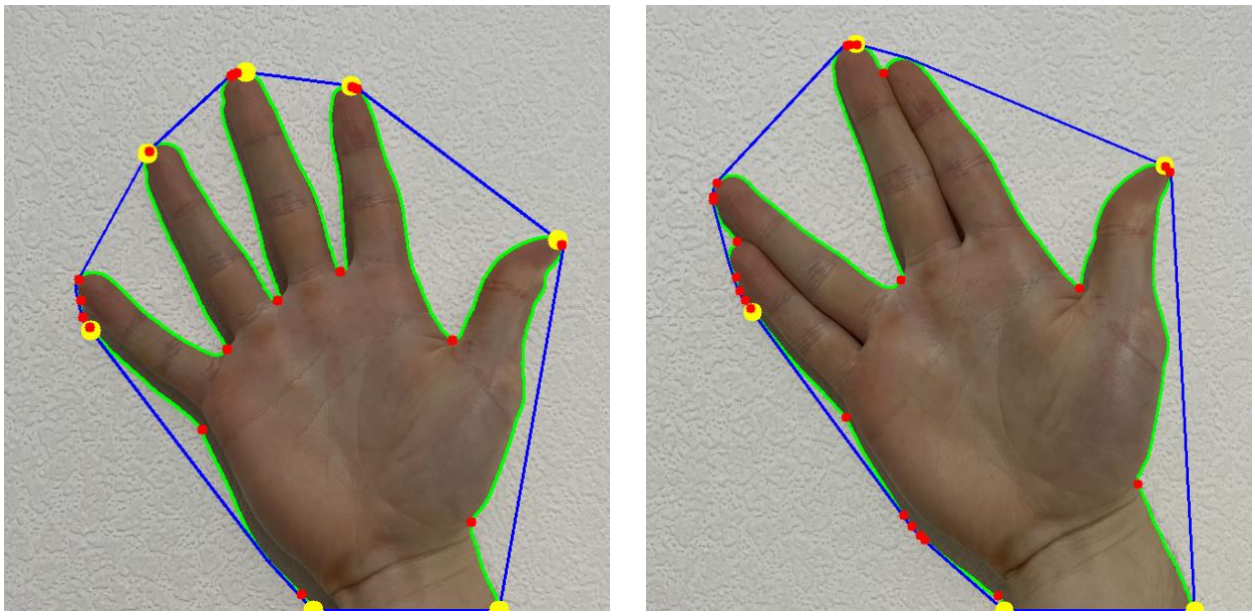
Рис. 20. Пример вычисления точек дефектов выпуклости (красным).

Можно видеть, что некоторые точки очень близко находятся друг к другу. Это связано с тем, что выпуклая оболочка строится не совсем ровно и касается

контура несколько раз почти в одном и том же месте. Чтобы избавиться от этого был разработан следующий алгоритм.

Берётся первая попавшаяся точка дефекта выпуклости, затем перебираются оставшиеся, пока они попадают в круг радиусом α с центром в первой точке. Эти точки не попадают в итоговый результат. Затем берётся следующая точка и т.д. Если после прохождения всех точек их количество больше 7 (2 точки снизу руки + максимум 5 точек на пальцах), то α увеличивается и алгоритм начинается заново.

Реализация алгоритма очистки находится в приложении Г. Результаты работы программы поиска точек дефектов выпуклости с очисткой «лишних» точек представлены на рисунке 21.



а) б)

Рис. 21. Результат очистки «лишних» точек дефектов выпуклости (оставшиеся точки отмечены на рисунках жёлтым цветом).

В контексте обработки изображения и компьютерного зрения каждая фигура состоит из пикселей и центром масс является взвешенное среднее всех пикселей, составляющих фигуру.

Центр масс контура можно найти с помощью *момента*. Момент изображения – это конкретное средневзвешенное значение интенсивности пикселей изображения. Как и во всех остальных науках, моменты характеризуют

распределение материи относительно точки или оси. Формула для момента изображения выглядит следующим образом:

$$M_{ij} = \sum_x \sum_y x^i y^j I(x,y), \tag{14}$$

где:

$I(x,y)$ – интенсивность пикселя в точке (x,y);

x, y относятся к строке и столбцу изображения.

Проблема формулы (14) состоит в том, что моменты чувствительны к позициям x и y. Если есть необходимость в определении моментов, независимых к месту расположения контура, то нужно использовать формулу *центральных моментов*:

$$M_{pq} = \sum_x \sum_y (x - \bar{x})^p (y - \bar{y})^q I(x,y), \tag{15}$$

где:

$\bar{x}$ и $\bar{y}$ – средние значения x и y соответственно.

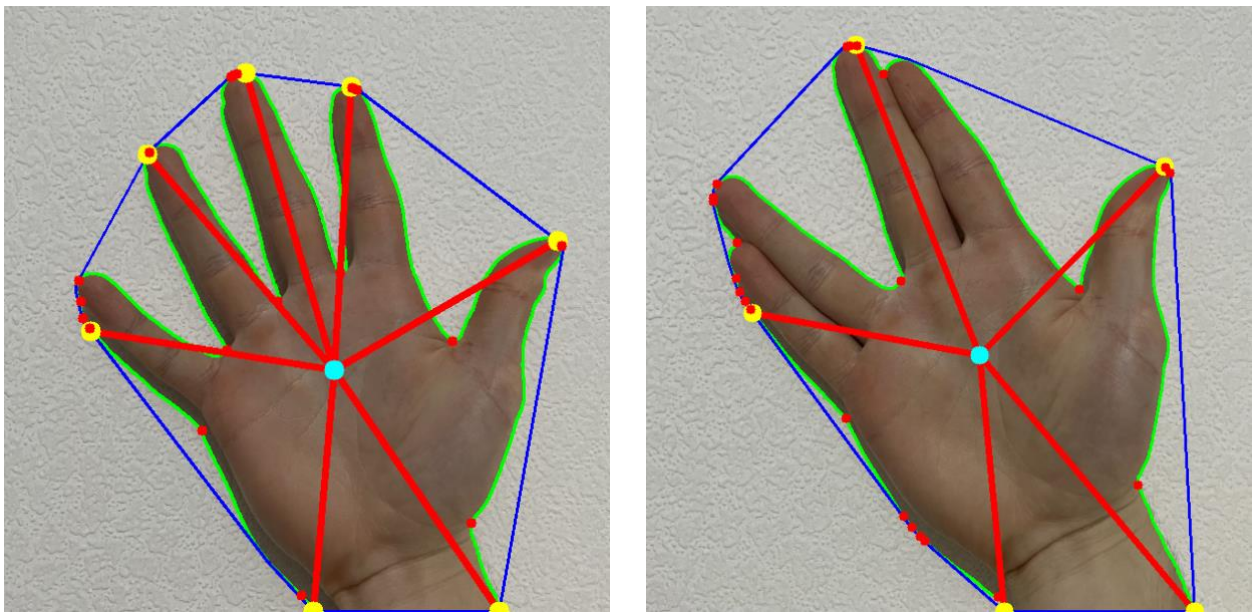
Таким образом координата центра масс изображения будет определяться как:

$$C_x = \frac{M_{10}}{M_{00}}, \quad C_y = \frac{M_{01}}{M_{00}}, \tag{16}$$

где:

(C_x, C_y) – координата центра масс изображения.

Определив координаты центра масс с помощью формул (15) и (16) и построив лучи от точек дефектов выпуклости к нему, получаем результат, представленный на рисунке 22.



а) б)

Рис. 22. Результат поиска ключевых точек на кисти.

Объединив всё вышеизложенное, можно составить общую методику детектирования, описания и извлечения конфигурации руки человека без использования глубокого обучения.

2.4. Методика и оценка её эффективности

Основными критериями выбора методики являются *гибкость* использования, *скорость* и *точность* работы. В связи с этим в качестве итогового алгоритма детектирования кисти и извлечения её конфигурации, не применяя методы глубокого обучения, был взят следующий:

1. Входное изображения разбивается на два множества «фон» и «полезная часть» с помощью метода MoG. Ранее было указано, что данный метод обладает свойством *обучения*, что позволяет его использовать в системах реального времени без жёсткого задания вида фона изображения. В свою очередь данный факт приводит к *гибкому* использованию данного алгоритма.
2. «Полезная часть» изображения переводится в монохромное изображение, искусственно размывается и подвергается обработке

методом Оцу, удаляя неточности, возникшие в результате первого шага. На данном этапе принимается, что кисть полностью отделена от фона.

3. На изображении кисти производится поиск контура с помощью топологического структурного анализа изображения с помощью отслеживания границ. Если найденных контуров несколько, то выбирается контур с наибольшей площадью, найденной по формуле площади Гаусса. На данном этапе контур кисти считается найденным.
4. Выполняется поиск точки центра масс контура с помощью формулы (16). Данная точка является центром ладони кисти.
5. Строится выпуклая оболочка точек контура кисти с помощью алгоритма Грэхема.
6. На основе выпуклой оболочки, найденной на шаге 4, и контура, найденного на шаге 3, находятся точки дефектов выпуклости, которые являются точками пальцев кисти.
7. Точка центра ладони и точки, найденные на шаге 6, составляют конфигурацию кисти, а именно её расположение в кадре и количество выгнутых пальцев.

Проведём тестирование данной методики. Входные данные – видеофрагмент длительностью 20 секунд с частотой кадров 30 кадров в секунду, то есть всего $20 \cdot 30 = 600$ изображений. Выходные данные – последовательность изображений в виде кадров из видеофрагмента с изображенными результатами работы.

При просмотре результатов было выявлено, что при резких перемещениях кисти руки в кадре алгоритм ненадолго «сбивается» вследствие переобучения метода MoG и из-за этого на некоторых кадрах ладонь распознаётся не точно, и, соответственно, остальные шаги выполняются неверно. Численные результаты представлены в таблице 3.

Таблица 3.

Результаты работы алгоритма.

Всего изображений	600
Кисть верно распознана	473
Конфигурация кисти верно определена	358

Таким образом, точность детектирования составила $473/600 = 0.79$, а точность извлечения конфигурации – $358/600 = 0.597$.

Разбор причин неверного определения конфигурации кисти показал, что при жесте «V» с двумя пальцами, безымянный палец и мизинец слегка выходят за границу ладони, вследствие чего данные пальцы были указаны как вытянутые и итоговое количество пальцев равно 4, что является неверным. Можно сделать вывод, что данная методика отлично работает при несложных движениях кисти, таких как сжатие в кулак, вытягивание пальцев по одному, либо по несколько, если пальцы расположены поодаль друг от друга.

3. НЕЙРОСЕТЕВЫЕ ТЕХНОЛОГИИ В ДЕТЕКТИРОВАНИИ И ОПИСАНИИ КИСТИ ЧЕЛОВЕКА

Использование одного из класса методов искусственного интеллекта – машинного обучения – для решения самых разнообразных задач приобретает всё более широкий спектр, особенно это характерно для тех задач, найти алгоритмическое решение которых либо не представляется возможным, либо оно неразумно и неэффективно. Машинное обучение характеризуется не прямым решением задачи, а обучением за счёт применения решений множества сходных задач. Для построения таких методов используются средства математической статистики, численных методов, математического анализа, методов оптимизации, теории вероятности, теории графов и различные техники работы с данными в цифровой форме.

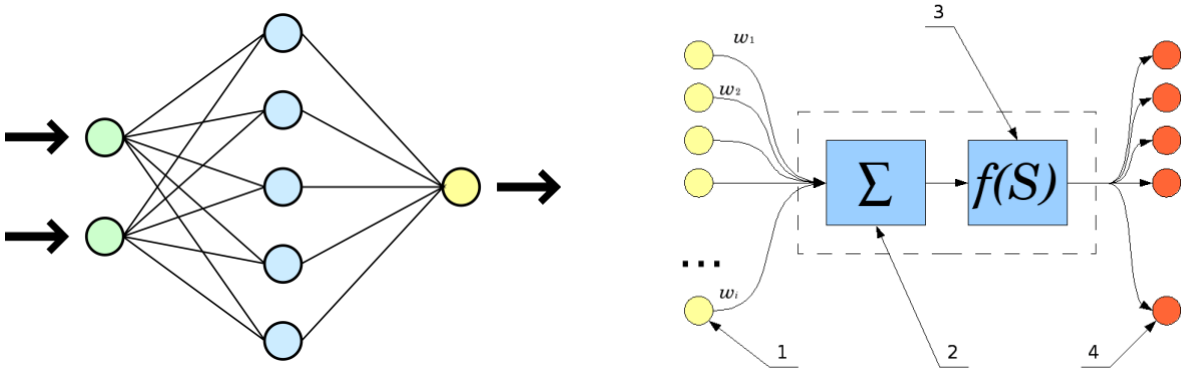
Подмножеством машинного обучения является глубокое обучение, являющееся совокупностью методов машинного обучения, основанных на обучении представлениям, а не специализированных алгоритмах под конкретные задачи, то есть на основе искусственных нейронных сетей (ИНС). Процесс обучения называется *глубоким*, так как структура ИНС состоит из нескольких входных, выходных и скрытых слоёв. Каждый слой содержит части, преобразующие входные данные в сведения, которые следующие слой может использовать для определённой задачи прогнозирования. Рассмотрим нейронные сети и их виды подробнее.

3.1. Нейронные сети и их виды

Нейронная сеть (или искусственная нейронная сеть, ИНС) – это математическая модель, а также её программная или аппаратная реализация, построенная по принципу организации и функционирования *биологических нейронных сетей* – сетей нервных клеток живого организма. Если же говорить проще, то нейронная сеть – это последовательность *нейронов*, соединённых между собой синапсами. На рис. 23а изображена схема простой нейросети,

состоящей из трёх слоёв: зелёным обозначены входные нейроны, голубым – скрытые нейроны (скрытый слой) и жёлтым – выходной нейрон.

Математически, нейрон представляют как некоторую нелинейную функцию от единственного аргумента – линейной комбинации всех входных сигналов. Данную функцию называют *функцией активации* или *передаточной функцией*. На рис. 23б изображена схема нейрона, цифрой 1 обозначены нейроны, выходные сигналы которых поступают на вход данного, цифрой 2 – сумматор входных сигналов, цифрой 3 – вычислитель передаточной функции и цифрой 4 – нейроны, на входы которых подаётся выходной сигнал данного.



а) б)

Рис. 23. Схема простой нейронной сети (а), искусственного нейрона (б).

Описать нейрон уравнением можно так:

$$y = f(u), \tag{17}$$

где:

$u = \sum_{i=1}^{n} w_i x_i + w_0 x_0$,

x_i и w_i – соответственно сигналы на входах и веса входов

u – индуцированное локальное поле,

$f(u)$ – передаточная функция.

Возможные значения сигналов на входах нейрона обычно считают заданными в отрезке $[0, 1]$. Дополнительный вход x_0 и соответствующий ему вес w_0 используются для инициализации нейрона. Под инициализацией понимается смещение активационной функции нейрона по горизонтальной оси, то есть

формирование порога чувствительности нейрона. Иногда к выходу нейрона специально добавляют некую случайную величину, называемую сдвигом.

Передаточная функция нейрона определяет зависимость сигнала на выходе нейрона от взвешенной суммы сигналов на его входах. В большинстве случаев она является монотонно возрастающей, а область значений лежит в $[-1, 1]$ или $[0, 1]$. Нейрон полностью характеризуется своей передаточной функцией. Рассмотрим основные типы передаточных функций.

Линейная передаточная функция. Задаётся формулой:

$$f(x) = \begin{cases} 0, x \leq 0 \\ 1, x \geq 1 \\ x \end{cases}$$

Иногда возможен сдвиг функции по обеим осям. Одним из преимуществ является дифференцируемость на всей числовой оси.

Пороговая передаточная функция. Иногда называют *функцией Хевисайда*. Представляет собой резкий скачок на некоторый уровень T. Задаётся формулой:

$$f(x) = \begin{cases} 1, x \geq T \\ 0 \end{cases}$$

Здесь $T = -w_0 x_0$ – сдвиг функции активации относительно горизонтальной оси. Недостаток – не дифференцируема на всей оси абсцисс.

Сигмоидальная передаточная функция. Является самой часто используемой функцией. Существует несколько видов её математической записи. *Логистическая функция*:

$$\sigma(x) = \frac{1}{1 + e^{-tx}},$$

где t – параметр функции, определяющий её крутизну. Важным достоинством этой функции является простота её производной:

$$\frac{d\sigma(x)}{dx} = t\sigma(x)\big(1 - \sigma(x)\big).$$

Особенностью нейронов с такой передаточной характеристикой является их усиление сильных сигналов существенно меньше, чем слабых. Это предотвращает насыщение от больших сигналов.

Другой математической записью сигмоидальной функции является *гиперболический тангенс*:

$$th(Ax) = \frac{e^{Ax} - e^{-Ax}}{e^{Ax} + e^{-Ax}}.$$

Она отличается от логистической функции тем, что её область значений лежит в интервале $(-1, 1)$.

Радиально-базисная функция передачи. Она принимает в качестве аргумента расстояние между входным вектором и некоторым наперёд заданным центром активационной функции. Значение этой функции тем выше, чем ближе входной вектор к центру. В качестве такой функции можно взять *функцию Гаусса*:

$$y = e^{-\frac{(S-R)^2}{2\sigma^2}},$$

где:

$S = \|X - C\|$ – расстояние между центром C и вектором входных сигналов X,

σ – определяет скорость спада функции при удалении вектора от центра и называется *шириной окна*,

R – определяет сдвиг активационной функции по оси абсцисс.

В качестве расстояния между векторами могут быть использованы различные метрики, обычно используется евклидово расстояние.

Также в качестве передаточных функций могут использоваться: *экспонента*, *тригонометрический синус*, *модуль, квадратичная*.

Каждая нейронная сеть включает в себя первый входной слой нейронов. Этот слой не выполняет каких-либо преобразований и вычислений, его задача состоит в том, чтобы принимать и распределять входные сигналы по остальным нейронам. Таким образом, этот слой единственный, являющийся общим для всех типов нейросетей, а критерием для деления является уже дальнейшая структура:

1. *Однослойная структура нейронной сети*. Представляет собой структуру взаимодействия нейронов, в которой сигналы со входного

слоя сразу направляются на выходной слой, который не только преобразует сигнал, но и сразу же выдаёт ответ. Как уже было сказано, первый входной слой только принимает и распределяет сигналы, а нужные вычисления происходят уже во втором слое. Входные нейроны являются объединёнными с основным слоем с помощью синапсов с разными весами, обеспечивающими качество связей.

2. *Многослойная нейронная сеть*. Данная сеть помимо выходного и входного слоёв имеет ещё несколько скрытых промежуточных слоёв. Число этих слоёв зависит от степени сложности нейронной сети. Соответствующие решения обладают большими возможностями, если сравнивать с однослойными, ведь в процессе обработки данных каждый промежуточный слой – это промежуточный этап, на котором осуществляется обработка и распределение информации.

Кроме количества слоёв, нейронные сети можно классифицировать по направлению распределения информации по синапсам между нейронами:

1. *Нейросети прямого распространения (однонаправленные)*. В этой структуре сигнал перемещается строго по направлению от входного слоя к выходному. Движение сигнала в обратном направлении не осуществляется и в принципе невозможно. Сегодня разработки этого плана распространены широко и успешно решают задачи распознавания образов, прогнозирования и кластеризации.
2. *Рекуррентные нейронные сети (с обратными связями)*. Для данных типов сетей характерно движение сигнала и в прямом, и в обратном направлении. В итоге результат выхода способен возвращаться на вход. Выход нейрона определяется весовыми характеристиками и входными сигналами и дополняется предыдущими выходами, снова вернувшимися на вход. Этим нейросетям присуща функция кратковременной памяти, на основании чего сигналы восстанавливаются и дополняются во время их обработки.
3. *Радиально-базисные функции*.

4. *Самоорганизующиеся карты*.

Основными критериями выбора типов среди огромного многообразия искусственных нейронных сетей являются способность эффективно работать с изображениями и с последовательностями. Сверточная нейронная сеть удовлетворяет первому критерию, а рекуррентная – второму. Рассмотрим их поподробнее.

3.1.1. Свёрточная нейронная сеть

Свёрточная нейронная сеть (CNN – Convolutional Neural Network) – специальная архитектура нейронных сетей, предложенная Яном Лекуном в 1988 году и нацеленная на эффективное распознавание образов. Как следует из названия, вместо обычных слоёв в ней используется операция *свёртки*, суть которой состоит в том, что каждый фрагмент изображения умножается на матрицу (ядро) свёртки по элементно, а результат суммируется и записывается в аналогичную позицию выходного изображения.

Работа свёрточной нейронной сети обычно интерпретируется как переход от конкретных особенностей изображения к более абстрактным деталям, и далее к ещё более абстрактным деталям вплоть до выделения понятий высокого уровня. При этом сеть самонастраивается и вырабатывает сама необходимую иерархию абстрактных признаков (последовательности карт признаков), фильтруя маловажные детали и выделяя существенное.

Подобная интерпретация носит скорее метафорический или иллюстративный характер. Фактически «признаки», вырабатываемые сложной сетью, малопонятны и трудны для интерпретации настолько, что на практике суть этих признаков даже не пытаются понять, тем более «подправлять», а вместо этого для улучшения результатов распознавания меняют структуру и архитектуру сети. Так, игнорирование системой каких-то существенных явлений может говорить о том, что либо не хватает данных для обучения, либо структура сети обладает недостатками, и система не может выработать эффективных признаков для данных явлений.

Оператор свёртки с ядром $K(i,j),\ i=\overline{0,N-1},\ j=\overline{0,M-1}$, получая на вход изображение $I(p,q)$, на выходе формирует матрицу $L(p,q)$:

$$L(p,q)=\sum_{n=0}^{N-1}\sum_{m=0}^{M-1}K(n,m)\cdot I\left(p+n-\frac{N}{2},q+m-\frac{M}{2}\right).$$

Для многоканального изображения $I(p,q,c)$ (или просто трёхмерной матрицы) используется трёхмерное ядро свёртки $K(i,j,c)$:

$$L(p,q)=\sum_{c}\sum_{n=0}^{N-1}\sum_{m=0}^{M-1}K(p,q,c)\cdot I(p+n-\frac{N}{2},q+m-\frac{M}{2},c).$$

Так же обычно в свёрточную нейронную сеть включаются оператор пулинга (MaxPool), для которого определяются два параметра: размер ядра ($M\times N$) и сдвиг ($D_x\times D_y$):

$$M(p,q,c)=\max\{I(i,j,c)\mid i=D_x\cdot p,\dots(D_x\cdot p+M);j=D_y\cdot q,\dots,\left(D_y\cdot q+N\right)\}.$$

Пример свёрточной сети изображён на рис. 24.

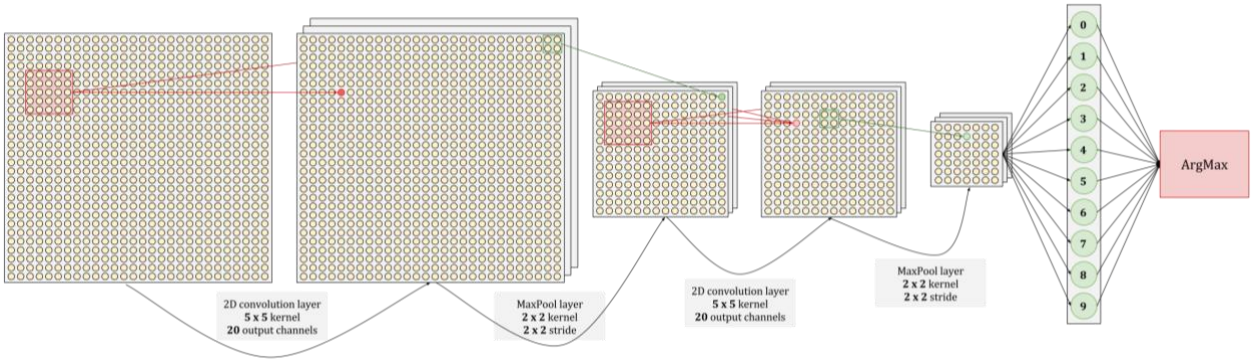


Рис. 24. Пример свёрточной сети.

Вначале применяются 20 разных ядер свёртки размерности 5×5, получая из матрицы $28\times28\times1$ трёхмерную матрицу $28\times28\times20$. Затем используется оператор пулинга 2×2, который уменьшает матрицу в два раза, сводя её к размеру $14\times14\times20$. Ещё 20 разных ядер свёртки той же размерности и слой пулинга «сжимают» матрицу до размера $7\times7\times20$, получая $7\cdot7\cdot20=980$ величин, которые подаются на выходной слой с 10 нейронами.

3.1.2. Рекуррентные нейронные сети

Рекуррентные нейронные сети (RNN – Recurrent Neural Network) – вид нейронных сетей, где связи между элементами образуют направленную последовательность. Благодаря этому появляется возможность обрабатывать серии событий во времени или последовательные пространственные цепочки. В отличие от многослойных перцептронов, рекуррентные сети могут использовать свою внутреннюю память для обработки последовательностей *произвольной длины*, поэтому сети RNN применимы в таких задачах, где нечто целостное разбито на части, например: распознавание рукописного текста, распознавание речи или распознавание жестов.

Любая рекуррентная нейронная сеть имеет форму цепочки повторяющихся модулей нейронной сети. В обычной RNN структура одного такого модуля очень проста, например, он может представлять собой один слой с функцией активации tanh (гиперболический тангенс) (рис. 25).

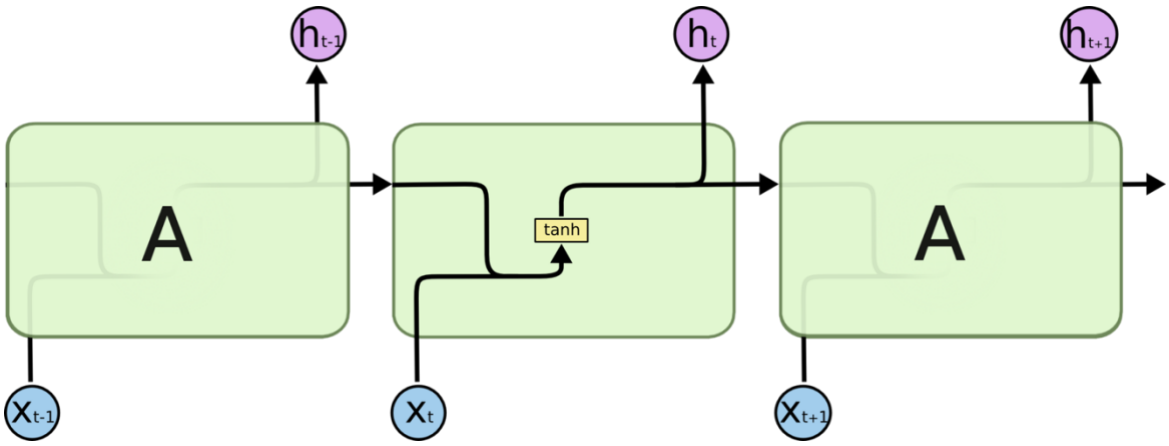


Рис. 25. Структура обычной RNN.

Одна из привлекательных идей RNN состоит в том, что они потенциально имеют возможность связывать предыдущую информацию с текущей задачей, так, например, знания о предыдущем кадре видео могут помочь в понимании текущего кадра. Однако предоставление такой возможности зависит от разрыва между актуальной информацией и точкой её применения. По мере роста этого

расстояния, RNN теряют способность связывать информацию. Однако модификация RNN LSTM не имеет такой проблемы.

Долгая краткосрочная память (Long Short-Term Memory – LSTM) – особая разновидность архитектуры рекуррентных нейронных сетей, способная к обучения долговременным зависимостям. Данный вид был разработан в 1997 году специально для решения проблемы долговременной зависимости, то есть они способны запоминать информацию на долгие периоды времени [31].

Структура LSTM по аналогии с RNN также напоминает цепочку, но её модули выглядят иначе (рис. 26): модули имеют четыре особым образом взаимодействующих между собой слоя вместо одного.

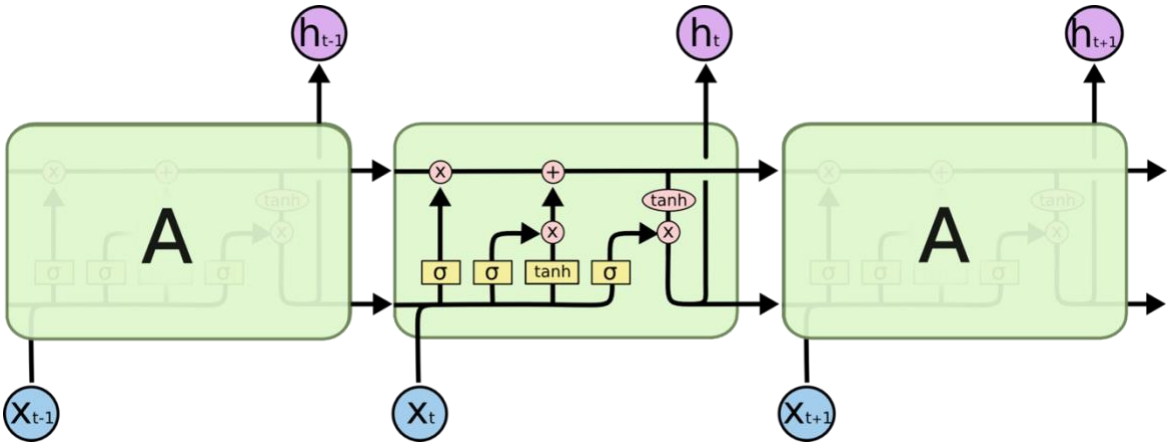


Рис. 26. Структура LSTM.

Ключевой компонент LSTM – это состояние ячейки (cell state) – горизонтальная линия, проходящая по верхней части схемы (рис. 27).

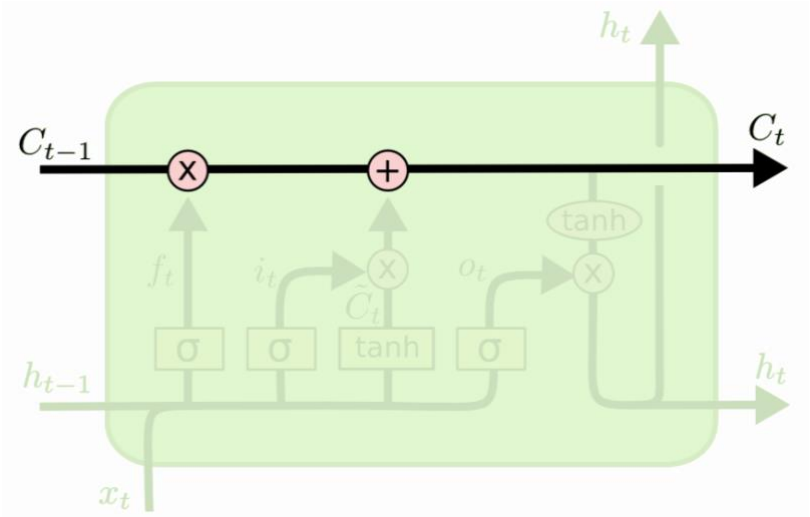
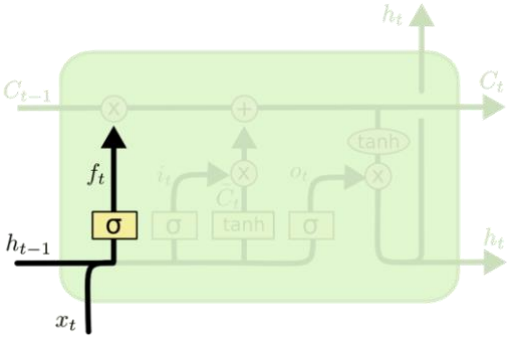


Рис. 27. Состояние ячейки в LSTM.

Состояние ячейки напоминает ленту на конвейере, проходя напрямую через всю цепочку, оно участвует лишь в нескольких линейных преобразованиях, то есть информация «течёт» по ней, не подвергаясь изменениям. Тем не менее, LSTM может удалять информацию из состояния ячейки, посредством регулирования структурами, называемыми фильтрами (gates). Фильтры позволяют пропускать информацию на основании некоторых условий и состоят из слоя сигмоидальной нейронной сети и операции поточечного умножения. Сигмоидальный слой имеет диапазон выходных значений, лежащий в отрезке [0,1], который обозначает, какую долю каждого блока информации следует пропустить дальше по сети. Ноль в данном случае означает «не пропускать ничего», единица – «пропустить всё». В LSTM три таких фильтра, позволяющих защищать и контролировать состояние ячейки.

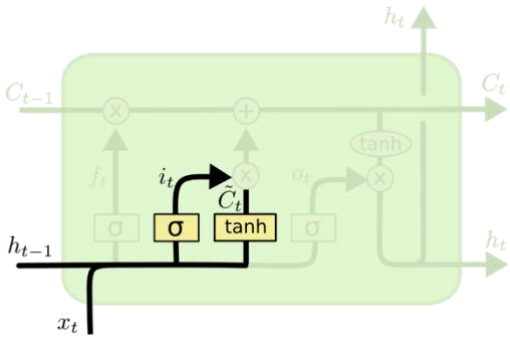
Первый шаг в LSTM – определить, какую информацию можно выбросить из состояния ячейки. Это решение принимает сигмоидальный слой, называемый «слоем фильтра забывания» (forget gate layer) (рис. 28а). В зависимости от h_{t-1} и x_t этот слой на основе выхода сигмоидального слоя принимает решение о сохранении каждого числа из состояния ячейки C_{t-1}. Следующий шаг – решить, какая новая информация будет храниться в состоянии ячейки. Этот этап состоит из двух частей. Сначала сигмоидальный слой под названием «слой входного

фильтра» (input layer gate) (рис. 28б) определяет, какие значения следует обновить. Затем tanh-слой строит вектор новых значений-кандидатов $\widetilde{C_t}$, которые можно добавить в состояние ячейки. И наконец старой состояние ячейки C_{t-1} заменяется на новое состояние C_t, умножая его на f_t и прибавляя $i_t \cdot \widetilde{C_t}$. Это новые значения-кандидаты, умноженные на t – на сколько необходимо обновить каждое из значений состояния. Последним этапом работы LSTM сети является отбор информации, получаемой на выходе. Выходные данные будут основаны на применении сигмоидального слоя и текущем состоянии ячейки, пройденном через tanh-слой.



$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

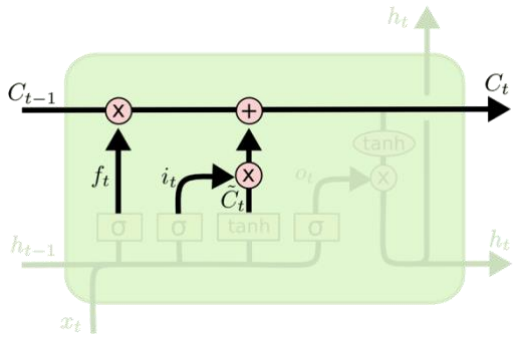
а)



$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

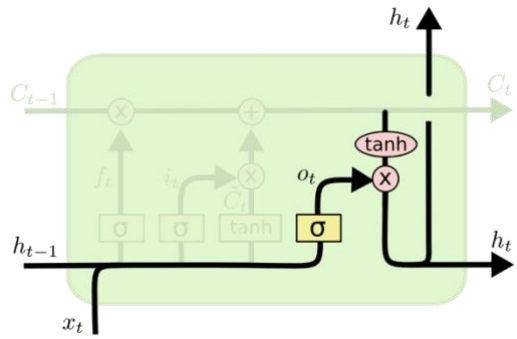
$$\widetilde{C_t} = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

б)



$$C_t = f_t \cdot C_{t-1} + i_t \cdot \widetilde{C_t}$$

в)



$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t \cdot \tanh(C_t)$$

г)

Рис. 28. Фильтры в LSTM: «слой фильтра забывания» (а), «слой входного фильтра» (б), замена старого состояния на новое (в), получение информации на выход (г).

Таким образом, хорошая приспособленность к обучению на задачах классификации и относительная невосприимчивость к длительности временных разрывов даёт LSTM преимущество по отношению к альтернативным рекуррентным нейронным сетям.

3.2. Детекторы, использующие ограничивающие прямоугольники

Когда нейронные сети в компьютерном зрении не применялись так широко, для детектирования объектов на изображении использовался достаточно простой подход. Тренировался бинарный классификатор, который мог определить есть ли на картинке объект определённого типа или нет. После того как классификатор был натренирован, по изображению «проходило» скользящее окно (sliding window) и для каждого прямоугольника, обрамлённого этим окном, применялся этот бинарный классификатор. Поскольку на одном изображении объекты могут быть и маленькими, и большими, то необходимо было либо запускать окно разных размеров, либо из исходного изображения строить пирамиду разных масштабов и запускать окно по каждому слою пирамиды. Основная задача была найти такие *особенности (features)*, которые бы позволили натренировать хороший бинарный классификатор.

С внедрением свёрточных нейронных сетей произошёл скачок в поиске хороших особенностей. Например, уже в 2013 появилась статья [15], про сеть Overfeat, где подход со скользящим окном обобщался на свёрточную нейронную сеть, применяемую в качестве классификатора.

Затем появилась линейка R-CNN, Fast R-CNN, Faster R-CNN в статьях [5], [4], [13], и расширения этого подхода. Здесь, вместо скользящего окна, используется предварительный детектор для поиска объектов, а затем сеть определяет класс объекта и уточняет его прямоугольник. Точность у данных детекторов довольно высока, но они всё ещё довольно медленные.

Наконец, третий подход появился в статье [12] – YOLO (You Only Look Once) и потом продолжился в [17] – SSD (Single Shot Detector). Для ускорения процесса детектирования в этих работах предлагается не искать объекты, пробегая по изображению скользящим окном или каким-то вариантом поискового алгоритма, а выбрать некоторое фиксированное количество прямоугольников и для каждого проверить наличие в нём объекта, и, если объект найден, то определить какого он класса и уточнить ограничивающую его рамку

(*bounding box*). Например, в [12] на изображение накладывается решётка, в каждой клетке которой проверяется наличие центра какого-нибудь объекта. Рассмотрим данный подход подробнее.

3.2.1. YOLO

Авторы [12] предложили вместо того, чтобы искать объекты с последующей их проверкой классификатором, разбить изображение равномерной сеткой $S \times S$ и для каждой ячейки определять есть ли в ней центр какого-то объекта, какого он класса и его точный прямоугольник. То есть для каждой ячейки определять:

1. Набор из B пятёрок вида: $[x, y, w, h, p]$, где $x, y -$ относительные координаты центра объекта внутри ячейки, $w, h -$ размеры объекта, $p -$ уверенность сети, что объект есть и его прямоугольник задан правильно;
2. C вероятностей (по количеству классов объектов), что объект с центром в данной ячейке принадлежит соответствующему классу.

Итак, получая на вход изображение, на выходе сеть должна выдавать тензор размера $S \times S \times (B \cdot 5 + C)$. В общем виде, схема сети представлена на рис. 29.

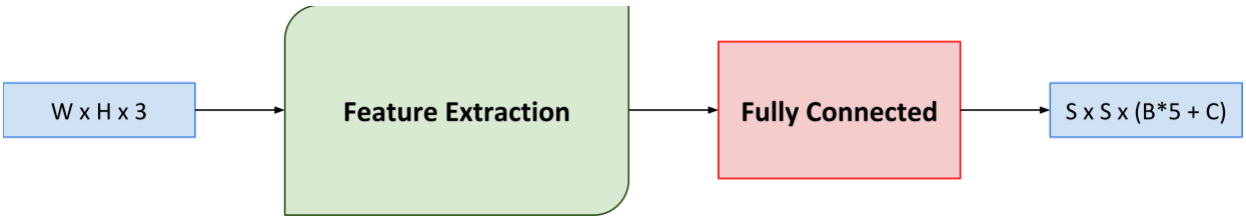


Рис. 29. Схема сети YOLO.

Авторы предложили:

1. Подавать на вход изображения размером 448×448;
2. Разбивать их при помощи решётки с $S = 7$;
3. Для каждой ячейки выдавать $B = 2$ предсказанных прямоугольника;

4. Так как в качестве набора данных для обучения берётся Pascal VOC, в котором размечено 20 классов, то $C = 20$.

Таким образом, на выходе сети будет тензор $7 \times 7 \times 30$. В качестве базовой сети для выделения особенностей авторы берут GoogLeNet с некоторыми упрощениями и в двух вариантах: 24 свёрточных слоя или 9 для особо быстрого детектирования.

3.2.3. SSD

YOLO показал неплохие результаты: скорость выросла, а качество упало не очень сильно. Поэтому авторы статьи [17] решили улучшить этот подход, повысив качество, при этом не снижая скорости.

Было предложено, так же, как и в YOLO, не искать объекты на фотографии, а задать фиксированный набор прямоугольников и проверять наличие объекта в каждом из них. Но, поскольку в YOLO претендентов было слишком мало, и к тому же особенности брались с последнего свёрточного слоя сети, а значит мелкие объекты могли в такой ситуации потеряться, то было решено брать особенности с разных свёрточных слоёв, что привело к увеличению набора претендентов. Для того, чтобы не заставлять сеть тренироваться, отдавая для каждого объекта ровно одно обнаружение, дополнительно после детектирования применяется NMS (Non-Maximum Suppression). На рисунке 30 изображено сравнение моделей SSD и YOLO. В SSD модель в конце базовой сети добавляются слоёв, предсказывающих смещения к *анкерам* (default box) различных масштабов и соотношений сторон и связанные с ними вероятности. SSD с размером входного изображения 300×300 значительно превосходит аналог YOLO 448×448 по точности и по скорости.

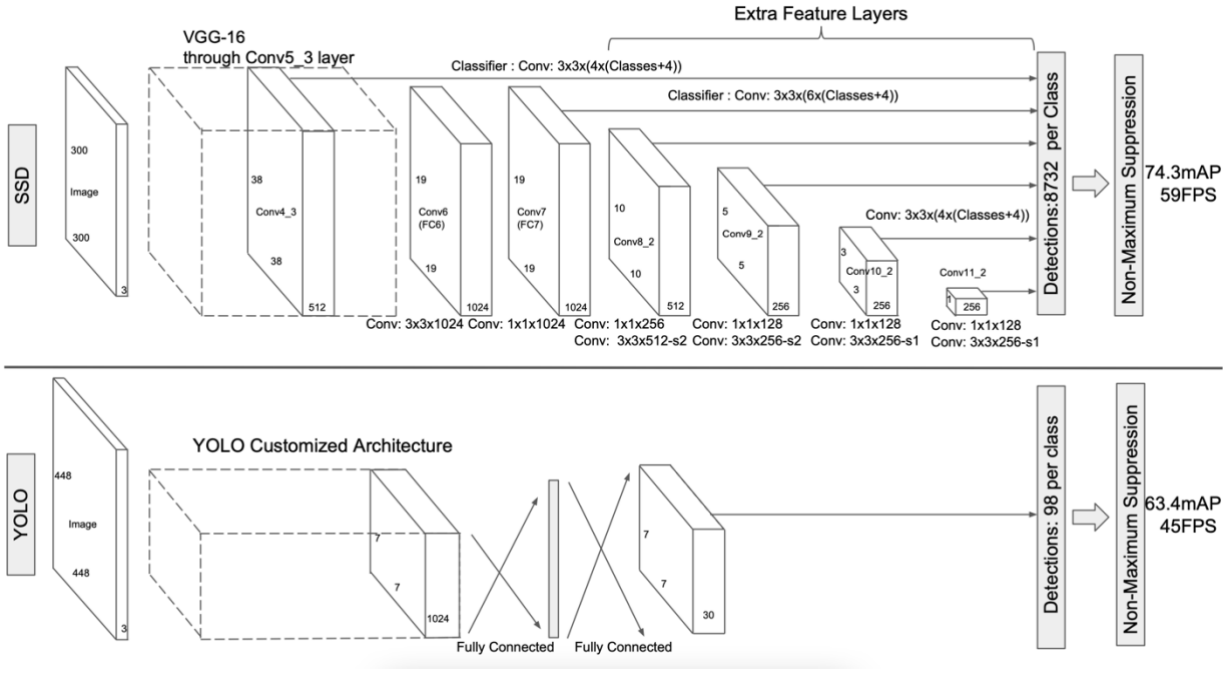
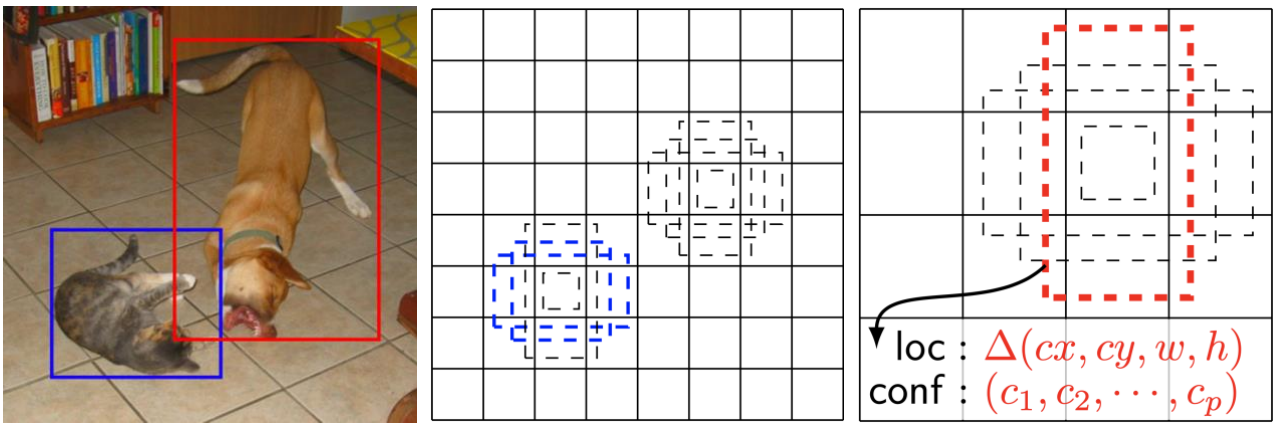


Рис. 30. Сравнение моделей детекторов SSD и YOLO.

Для работы SSD необходимо только входное изображение и рамки истинности для каждого объекта во время обучения (рис. 31а). С помощью свертки оценивается небольшой набор (например, 4) полей по умолчанию с разными соотношениями сторон в каждом месте на нескольких картах признаков с разными масштабами (например, 8×8 (рис. 31б) и 4×4 (рис. 31в)). Для каждой рамки по умолчанию прогнозируется как смещение формы, так и достоверность для всех категорий объектов $(c_1, c_2, \ldots, c_p)$. Во время обучения сначала сопоставляются эти рамки по умолчанию с рамками истинности. Например, совпали две рамки по умолчанию с котом и одно с собакой на рис. 31а, которые рассматриваются как положительные, а остальные – как отрицательные. Потеря модели представляет собой взвешенную сумму между потерей локализации (Smooth L1 [4]) и потерей достоверности (например, Softmax).



а) б) в)

Рис. 31. Пример работы SSD детектора.

Как уже было сказано, подход SSD основан на свёрточной сети с прямой связью, которая создаёт набор ограничивающих рамок (прямоугольников) фиксированного размера и оценивает наличие экземпляров класса объектов в этих рамках с последующим подавлением немаксимумов для получения результатов окончательных обнаружений. Модель SSD изображена на рис. 32.

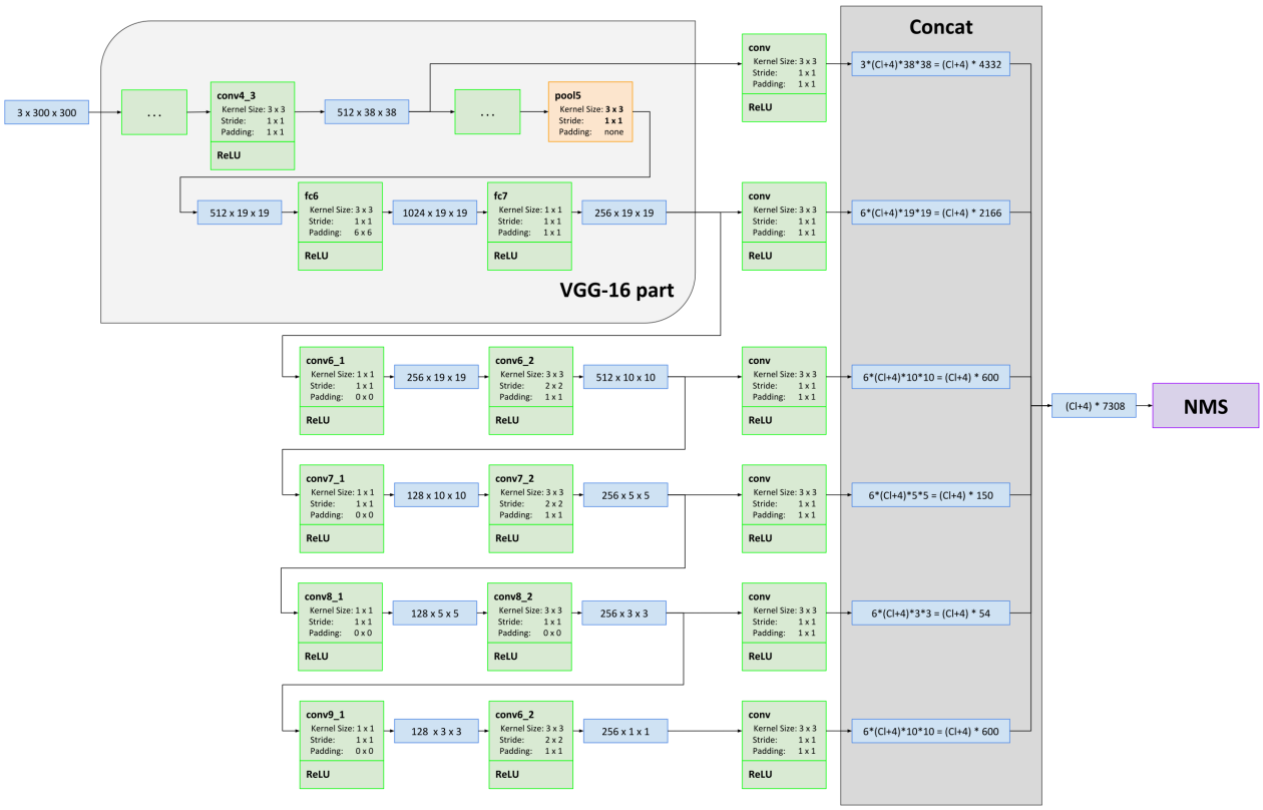


Рис. 32. Подробное описание модели SSD.

Первые слои нейросети основаны на стандартной архитектуре VGG-16, используемой для высококачественной классификации изображений (*базовая сеть)*. Затем в сеть добавляется вспомогательная структура для создания обнаружений со следующими ключевыми особенностями:

1. *Мультимасштабные карты ключевых точек для детектирования*.

   В конец усечённой базовой сети добавляются свёрточные слои ключевых точек. Эти слои постепенно уменьшаются в размере и позволяют прогнозировать обнаружение в нескольких масштабах. Свёрточная модель для прогнозирования обнаружений различается для каждого слоя объектов (Overfeat [15] и YOLO [12]), которые работают с картой признаков в одном масштабе;

2. *Свёрточные «предсказатели» для детектирования.*

   Каждый добавленный слой признаков (или, возможно, существующий слой признаков из базовой сети) может создавать фиксированный набор прогнозов обнаружения с использованием набора свёрточных фильтров. Для слоя признаков размера $m \times n$ с p каналами базовым элементом для прогнозирования параметров потенциального обнаружения является небольшое ядро 3×3, которое даёт либо оценку для категории, либо смещение формы относительно координат анкера. В каждом из $m \times n$ положений, где ядро применяется, оно выдает выходное значение. Выходные значения смещения прямоугольника измеряются относительно положения анкера относительно каждого местоположения карты признаков (архитектура YOLO использует промежуточный полносвязный слой вместо свёрточного фильтра для этого шага).

3. *Анкеры (*или *default boxes).*

   Набор анкеров связывается с каждой ячейкой карты признаков для нескольких карт признаков в начале нейросети. Анкеры разбивают карту признаков свёрткой так, что положение каждой ячейки относительно соответствующей ячейки фиксировано. В каждой ячейке

карты признаков прогнозируется смещение относительно форм анкеров в ячейке, а также оценки для каждого класса, которые указывают на присутствие экземпляра класса в каждой из этих рамок. В частности, для каждой рамки из k в заданном месте вычисляются оценки класса c и 4 смещения относительно исходной формы анкера. В результате получается $(c + 4)k$ фильтров, которые применяются вокруг каждого местоположения на карте признаков, которые дают $(c + 4)kmn$ выходов для карты признаков размера $m \times n$.

Ключевое отличие обучения SSD от обучения типичного детектора, который использует предложения регионов (то есть предлагается какой-либо регион изображения и классифицируется) заключается в эталонной информации, которая должна быть назначена конкретным выходам в фиксированном наборе выходов детектора. Некоторая вариация этого также требуется для обучения YOLO [12] и для этапа предложения региона Faster R-CNN [13] и Multibox [2]. Как только это назначение определено, функция потерь и обратное распространение применяются сквозным образом. Обучение также включает в себя выбор набора анкеров и масштабов по умолчанию для обнаружения, а также стратегии аугментации и получения сложных «неверных» тренировочных данных (hard negative mining).

Стратегия сопоставления. Во время обучения необходимо определять, какие анкеры соответствуют верному обнаружению и тренировать сеть соответствующим образом. Каждый верный прямоугольник, сопоставляются с каждым анкером, различающимся по местоположению, соотношению сторон и масштабу и с наилучшим коэффициентом Жаккара (как в MultiBox [2]). Затем, в отличие от MultiBox, в SSD сопоставляются анкеры с истинными прямоугольниками с коэффициентом Жаккара выше порогового значения $(0{,}5)$. Это упрощает задачу обучения, позволяя сети предсказывать высокие значения для нескольких перекрывающихся прямоугольников, а не требовать от неё выбора только одного с максимальным перекрытием.

Цель обучения. Цель обучения SSD получена из цели Multibox [2], но расширена до обработки нескольких категорий объектов. Пусть $x_{ij}^{p} = \{1, 0\} -$ индикатор для сопоставления i-того анкера с j-м истинным прямоугольником категории p. Из приведённой выше стратегии сопоставления следует, что можно иметь $\sum_i x_{ij}^{p} \geq 1$. Общая целевая функция потерь представляет собой взвешенную сумму потерь локализации (loc) и потерь достоверности (conf):

$$L(x, c, l, g) = \frac{1}{N}\Big(L_{conf}(x, c) + \alpha L_{loc}(x, l, g)\Big),$$

где:

N – число совпавших анкеров. Если $N = 0$, то потери $L = 0$,

α – весовой коэффициент, устанавливающийся равным 1 с помощью перекрёстной проверки (кросс-валидации или cross-validation).

– Потери локализации – это потери Smooth L1 [4] между предсказанным прямоугольником (l) и истинным прямоугольником (g). Подобно Faster R-CNN [13] происходит регрессия к смещениям для центра (cx, cy) ограничивающего блока по умолчанию (d), а также для его ширины (w) и высоты (h):

$$L_{loc}(x, l, g) = \sum_{i \in Pos}^{N} \sum_{m \in \{cx, cy, w, h\}} x_{ij}^{k} \mathrm{smooth}_{L1}\left(l_i^m - \hat{g}_j^m\right),$$

$$\hat{g}_j^{cx} = \frac{g_j^{cx} - d_i^{cx}}{d_i^w}, \qquad \hat{g}_j^{cy} = \frac{g_j^{cy} - d_i^{cy}}{d_i^h},$$

$$\hat{g}_j^{w} = \log\left(\frac{g_j^w}{d_i^w}\right), \qquad \hat{g}_j^{h} = \log\left(\frac{g_j^h}{d_i^h}\right).$$

– Потери достоверности – это потери Softmax по достоверности нескольких классов (c):

$$L_{conf}(x, c) = -\sum_{i \in Pos}^{N} x_{ij}^{p} \log\left(\hat{c}_i^p\right) - \sum_{i \in Neg} \log\left(\hat{c}_i^0\right),$$

где:

$$\hat{c}_i^p = \frac{\exp(c_i^p)}{\sum_p \exp(c_i^p)}.$$

Выбор масштабов и соотношений сторон для анкеров. Для обработки различных масштабов объектов некоторые методы [15, 7] предполагают обработку изображения разных размеров и последующее объединение результатов. Однако, используя карты признаков из нескольких разных слоёв в одной сети для прогнозирования, можно имитировать тот же эффект, а также использовать общие параметры для всех масштабов объектов. Предыдущие работы [11, 6] показали, что использование карт признаков из нижних слоёв может улучшить качество семантической сегментации, поскольку нижние слои захватывают более мелкие детали входных объектов. Точно так же в [10] показано, что добавление главного контекста, объединённого из карты признаков, может помочь сгладить результаты сегментации. Руководствуясь этими методами, в SSD для обнаружения используются как нижняя, так и верхняя карта признаков.

Известно, что карты признаков с разных уровней внутри сети имеют разные (эмпирические) размеры рецептивного поля [18]. К счастью, в структуре SSD анкеры не обязательно должны соответствовать фактическим рецептивным полям каждого слоя. Мозаика анкеров разрабатывается таким образом, чтобы определённые карты признаков научились реагировать на определённые масштабы объектов. Предположим, нужно использовать m карт признаков для предсказания. Масштаб блоков по умолчанию для каждой карты признаков рассчитывается как:

$$s_k = s_{min} + \frac{s_{max} - s_{min}}{m - 1}(k - 1), \qquad k \in [1, m],$$

где $s_{min} = 0.2, s_{max} = 0.9$, означающие, что самый нижний слой имеет масштаб 0.2, а самый верхний слой – 0.9 и все слои меду ними расположены через равные промежутки. Для анкеров устанавливаются различные пропорции и обозначаются как $a_r \in \left\{1, 2, 3, \frac{1}{2}, \frac{1}{3}\right\}$. Можно вычислить ширину ($w_k^a = s_k\sqrt{a_r}$) и

высоту $\left(h_k^a = \frac{s_k}{\sqrt{a_r}}\right)$ для каждого анкера. Для соотношения сторон 1 также добавляется анкер, масштаб которого $s_k' = \sqrt{s_k s_{k+1}}$, и в результате получается 6 анкеров для каждого местоположения карты признаков. Центр каждого анкера устанавливается в $(\frac{i+0.5}{|f_k|}, \frac{j+0.5}{|f_k|})$, где $|f_k|$ – это размер k-ого квадрата карты признаков, $i, j \in [0, |f_k|)$.

На практике можно также спроектировать распределение анкеров, чтобы наилучшим образом соответствовать конкретному набору данных. Вопрос о том, как спроектировать оптимальную мозаику, также остаётся открытым.

Комбинируя прогнозы для всех анкеров с разными масштабами и соотношениями сторон из всех местоположений на многих картах признаков, получается разнообразный набор прогнозов, охватывающий различные размеры и формы входных объектов. Например, на рис. 31 собака сопоставляется с блоком по умолчанию на карте признаков 4×4, но не с каким-либо блоком по умолчанию на карте признаков 8×8. Это связано с тем, что эти блоки имеют разные масштабы и не соответствуют блоку собаки, и поэтому во время тренировки считаются негативами.

Получение сложных «неверных» тренировочных данных (hard negative mining). После этапа сопоставления большинство анкеров являются неверными, особенно когда количество возможных анкеров велико. Это вносит значительный дисбаланс между положительными и отрицательными обучающими примерами. Вместо того, чтобы использовать все отрицательные примеры, они сортируются по их наибольшей потере достоверности для каждого анкера и выбираются так, чтобы соотношение между отрицательными и положительными примерами было не более чем 3: 1. Было обнаружено [17], что это приводит к более быстрой оптимизации и более стабильному обучению.

Аугментация данных. Чтобы сделать модель более устойчивой к различным размерам и формам входных объектов, каждое обучающее изображение случайным образом изменяется по одному из следующих правил:

– Используется исходное изображение без изменений;

– Из исходного изображения вырезается фрагмент таким образом, чтобы минимальный коэффициент Жаккара с объектами составлял $0.1, 0.3, 0.5, 0.7$ или 0.9;
– Из исходного изображения вырезается случайный фрагмент.

Размер каждого фрагмента выборки лежит в отрезке $[0.1, 1.0]$ от исходного размера изображения, а соотношение сторон находится в диапазоне от $\frac{1}{2}$ до 2. Перекрывающаяся часть истинного прямоугольника сохраняется, если его центр находится в фрагменте выборки. После вышеупомянутого шага размер каждого фрагмента изменяется до фиксированного и фрагмент горизонтально отражается с вероятностью 0.5, в дополнение к применению некоторых фотометрических искажений, подобных описанным в [8].

Как уже было отмечено, существует два класса методов детектирования объектов на изображениях: один основан на скользящих окнах, а другой – на классификации предложений регионов. До появления свёрточных нейронных сетей уровень развития этих двух подходов – модели DPM (Deformable Part Model [3]) и выборочного поиска [16] – имели сопоставимую производительность. Однако после резкого улучшения, внесённого R-CNN (Region-Based Convolutional Neural Network) [5], который сочетает в себе выборочный поиск предложений регионов и пост-классификацию на основе свёрточной сети, методы обнаружения объектов на основе предложений регионов стали преобладающими. Сравнение подходов представлено в таблице 4.

Таблица 4.

Сравнение подходов на тесте Pascal VOC2007.

Метод	mAP, %	FPS
Faster R-CNN (VGG16)	73.2	7
Fast YOLO	52.7	155
YOLO (VGG16)	66.4	21
SSD300	74.3	46
SSD512	76.8	19

SSD300 – единственный метод, способный работать в реальном времени и набравший более 70% mAP (mean Average Precision – средняя точность предсказания), являющийся компромиссом между скоростью и качеством работы.

Первоначальный подход R-CNN был улучшен различными способами. Первый набор подходов улучшает качество и скорость пост-классификации, поскольку требует классификации тысяч кадров изображений, что дорого и затратно по времени. SPPnet [7] значительно ускоряет исходный подход R-CNN за счёт ввода слоя объединения слоя объединения пространственных пирамид, который более устойчив к размеры и масштабу региона и позволяет слоям классификации переиспользовать признаки, вычисленные по картам признаков, сгенерированным для нескольких разрешений изображения. Fast R-CNN [4] расширяет возможности SPPnet, так что он может точно настраивать все уровни от начала до конца, минимизируя потери как для достоверности, так и для регрессии ограничивающей рамки, которая впервые была представлена в MultiBox [6] для изучения объектности.

Второй набор подходов улучшает качество генерации предложений, используя глубокие нейронные сети. В самых последних работах, таких как MultiBox [2], предложения региона выборочного поиска, которые основаны на низкоуровневых признаках изображения, заменены предложениями, сгенерированными непосредственно из отдельной глубокой нейронной сети. Это

дополнительно повышает точность обнаружения, но приводит к несколько сложной настройке, требующей обучения двух нейронных сетей с зависимостью между ними. Faster R-CNN [13] заменяет предложения выборочного поиска предложениями, полученными из сети региональных предложений (RPN), и вводит метод интеграции RPN с Fast R-CNN путем чередования точной настройки общих свёрточных слоёв и слоёв прогнозирования для этих двух сетей. Таким образом, предложения регионов используются для объединения признаков среднего уровня, а последний этап классификации обходится дешевле. SSD очень похож на сеть региональных предложений (RPN) в Faster R-CNN, поскольку также используется фиксированный набор блоков (по умолчанию) для прогнозирования, аналогично полям привязки в RPN. Но вместо того, чтобы использовать их для объединения признаков и оценки другого классификатора, оценка для каждой категории объектов получается одновременно в каждом блоке. Таким образом, этот подход позволяет избежать осложнений, связанных с объединением RPN с Fast R-CNN, и его легче обучать, быстрее и проще интегрировать в другие задачи.

Другой набор методов, непосредственно связанных с подходом SSD, полностью пропускает шаг предложения и напрямую прогнозирует ограничивающие рамки и достоверность для нескольких категорий. OverFeat [15], глубокая версия метода скользящего окна, предсказывает ограничивающую рамку непосредственно из каждого местоположения самой верхней карты признаков после того, как известны достоверности базовых категорий объектов. YOLO [12] использует всю самую верхнюю карту признаков для прогнозирования как достоверности для нескольких категорий, так и ограничивающих рамок (которые являются общими для этих категорий). Метод SSD попадает в эту категорию, поскольку в нём нет шага предложения, но используются блоки по умолчанию. Однако этот подход является более гибким, чем существующие методы, поскольку существует возможность использовать блоки по умолчанию с разными соотношениями сторон для каждого местоположения признака из нескольких карт признаков в разных масштабах.

Если будет использоваться только один блок по умолчанию для каждого местоположения из самой верхней карты признаков, то SSD будет иметь архитектуру, аналогичную OverFeat [15]; если используется вся самая верхняя карта признаков и добавляется полносвязный слой для прогнозов вместо свёрточных предсказателей и не будет явно учитываться несколько соотношений сторон, то приблизительно воспроизведётся архитектура YOLO [12, 2].

Таким образом, резюмируя всё выше сказанное, можно отметить следующие особенности Single Shot детектора:

- Архитектура SSD позволяет детектировать объекты в реальном времени;
- Качество работы близко к Faster R-CNN;
- Детектирование происходит на разных масштабах, что позволяет локализовывать объекты разных размеров;
- Используется большое количество рамок по умолчанию, покрывающих входное изображение на разных масштабах;
- На этапе работы архитектура SSD300 осуществляет детектирование 7308 объектов, большая часть из которых впоследствии фильтруется;

Ключевой же особенностью является использование многомасштабных ограничительных рамок на выходах свёрточных слоёв, прикреплённых к нескольким картам признаков в верхней части сети. Экспериментально подтверждено в [17], что при соответствующих стратегиях обучения, большее количество тщательно выбранных анкеров приводит к повышению производительности.

Для решения задачи детектирования кисти на изображении Single Shot детектор подходит наилучшим образом, поскольку он имеет хорошее соотношение между скоростью работы и точностью. Этот же детектор также используется в библиотеке MediaPipe Hands, использующейся для поиска ключевых точек на руках. Рассмотрим её поподробнее.

3.3. Применение SSD в задаче детектирования кисти

Одним из применений Single Shot детектора в задаче поиске кисти на изображении является подход, реализованный в фреймворке MediaPipe от Google – MediaPipe Hands [30]. Для определения начального положения рук была разработана модель SSD, оптимизированная для использования в реальном времени. Поскольку обнаружение рук чрезвычайно сложная задача из-за огромного количества их конфигураций, то вместо детектора рук обучался детектор ладоней, так как получать прямоугольники негнущихся объектов, таких как ладони и кулаки, значительно проще, нежели обнаружение рук вместе с пальцами. Кроме того, поскольку ладони являются более мелкими объектами, алгоритм подавления немаксимумов хорошо работает даже для случаев пересечения двух рук, таких как рукопожатия. Более того, ладони могут быть смоделированы с помощью квадратных анкеров, игнорируя другие соотношения сторон, уменьшая количество анкеров в 3-5 раз. Во-вторых, кодирующий-декодирующий экстрактор признаков используется для большей осведомленности о контексте сцены даже для небольших объектов. Наконец, во время обучения минимизируется функция кросс-энтропийных потерь с динамическим масштабированием (focal loss), чтобы поддерживать большое количество анкеров, возникающих из-за большой дисперсии масштаба. С помощью описанных выше методов достигается средняя точность обнаружения ладони 95,7%. Использование обычной функции кросс-энтропийных потерь и без декодера даёт точность всего 86,22%.

После обнаружения ладони на всём изображении последующая модель выполняет точную локализацию 21 ключевой точки (рис. 33а) внутри обнаруженных областей руки посредством регрессии (как для определения цифры, написанной рукой), то есть прямого прогнозирования координат. Модель обучается последовательному внутреннему представлению позиции рук и устойчива даже к частично видимым рукам и самопересечениям. Так же стоит

отметить, что кроме 21 ключевой точки модель так же определяет вероятность присутствия кисти на кадре и классификацию типа кисти – левой или правой.

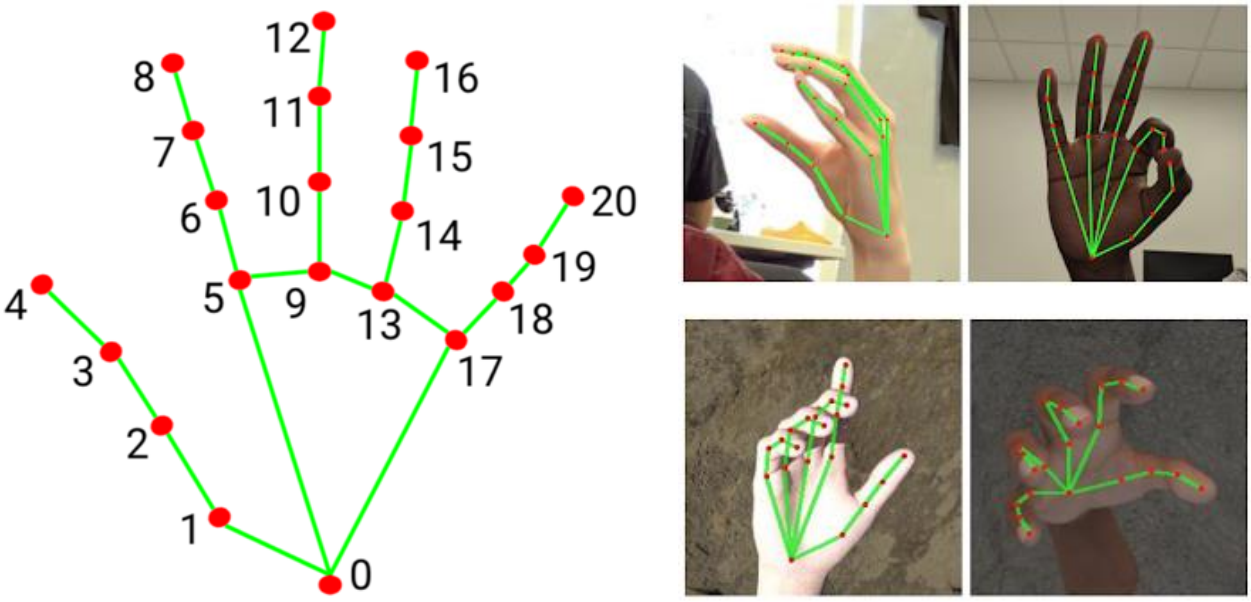


Рис. 33. Ключевые точки на руке, определяемые с помощью MediaPipe Hands (а). Данные для обучения (б). Сверху: реальные изображения рук, аннотированные вручную. Снизу: сгенерированные синтетические изображения рук с аннотациями.

Чтобы получить данные для обучения, вручную было промаркировано около 30 000 изображений реального мира с 21 координатой, как показано на рис. 33б. А также, чтобы лучше охватить возможные позиции рук и обеспечить дополнительное наблюдение за характером геометрии рук, была визуализирована высококачественная синтетическая модель руки на различных фонах, таким образом синтезируя набор из 100 000 изображений. Для детектора ладони использовались лишь 6 000 изображений данного набора, поскольку этого достаточно для его обучения локализации кисти.

Таким образом, библиотека состоит из детектора ладоней, определяющего область изображения, на которой изображена рука и модели трекинга руки, предсказывающая ключевые точки на области, полученной детектором.

Основываясь на данной технологии, полученные ключевые точки можно использовать в дальнейших приложениях, например, в распознавании жестов.

4. РАЗРАБОТКА ИНТЕРФЕЙСА ДЛЯ УПРАВЛЕНИЯ ЖЕСТАМИ

Прежде чем начать разрабатывать эргономичный интерфейс для управления путём распознавания, необходимо обобщить все ранее отмеченные выкладки и сформировать окончательный алгоритм работы человеко-машинного взаимодействия с помощью распознавания жестов.

4.1. Методики управления жестами

Схема общего алгоритма работы интерфейса изображена на рис. 34.

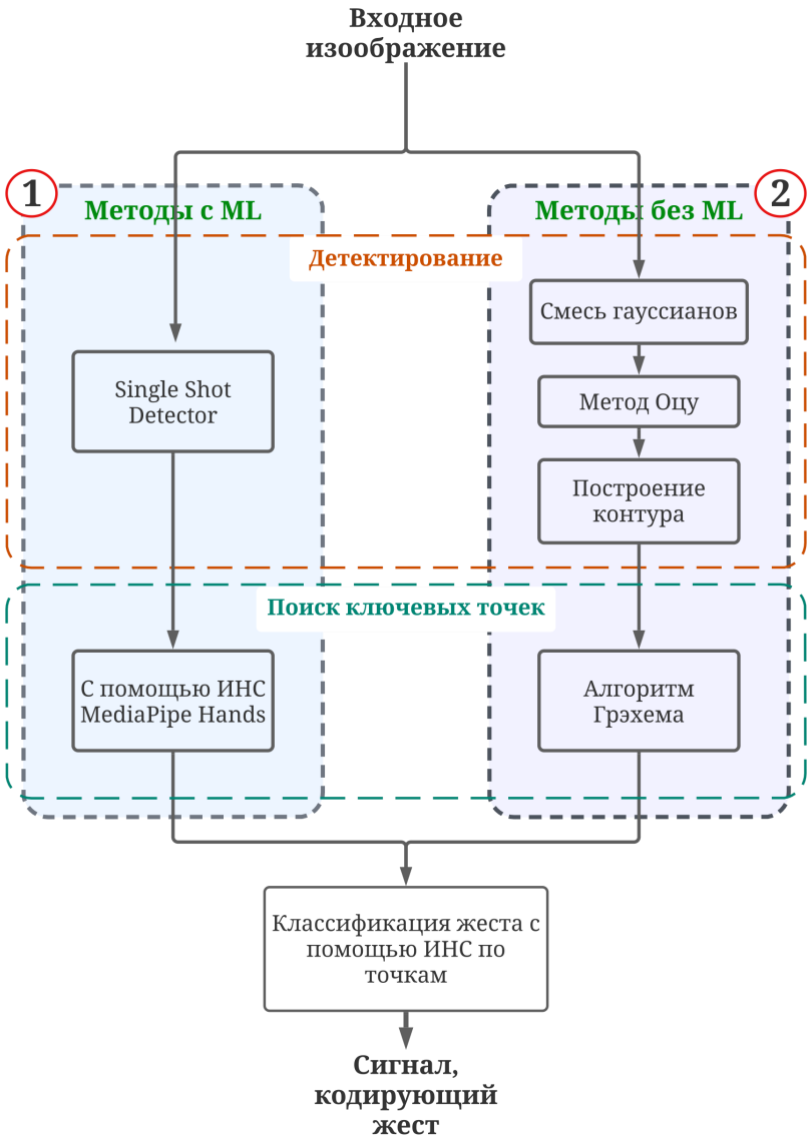


Рис. 34. Общая схема алгоритма работы интерфейса.

В основе алгоритма находятся два уже рассмотренных способа детектирования кисти и извлечения её конфигурации путём описания с помощью ключевых точек с применением методов машинного обучения и без. Поскольку последний способ позволяет получить только *максимум* 7 точек, которых может быть недостаточно для корректного распознавания статического жеста, а также при критических недостатках, связанных с точным определением точек лишь при не сложных движениях руки, то в качестве основного был выбран первый способ, всегда определяющий 21 точку и не имеющий проблем в их определении при разных движениях. Таким образом, ветвь 2 схемы на рисунке **Error! Reference source not found.** использоваться ***не*** будет.

4.2. Требования к реализации

Прежде чем выдвинуть требования к реализации интерфейса человеко-машинного взаимодействия необходимо вывести основные требования для эффективной работы систем «человек — техника», следующих из *инженерной психологии*:

1. Наличие высокопроизводительной и надёжной техники;
2. Конструкция техники и организация производственного процесса должны позволять человеку реализовывать все технические возможности;
3. Человек должен быть способным по своим качествам реализовывать все эти возможности, добиваться высокой производительности труда и обеспечивать выполнение производственных операций.

Требования к разрабатываемому человеко-машинному интерфейсу относятся ко второму пункту, то есть должно быть достигнуто согласование человека и техники как элементов единой системы. Под согласованием понимается максимальное приспособление техники к человеку (по параметрам конструкции и технологического процесса), максимальное приспособление человека к технике (по параметрам профессиональной пригодности и профессиональной подготовленности) и рациональное распределение функций

между человеком и автоматическими устройствами в системах «человек – техника».

Отсюда следует, что одним из требований к реализации интерфейса человеко-машинного взаимодействия на основе распознавания жестов является *эргономичность*, включающая в себя *гибкость* использования. Здесь под гибкостью понимается лёгкость и быстрота добавления новых жестов обычным пользователем, поскольку в зависимости от машины, сферы применения и количества действий количество жестов может варьироваться, а также изменяться их вид – могут добавляться новые или удаляться/переназначаться старые. Гибкость использования реализуется посредством реализации двух режимов работы: «*обучение*», включающее в себя добавление новых жестов, и «*применение*», идентифицирующее жест и посылающее связанный с этим жестов сигнал.

Таким образом в режиме «обучение» нужно учесть следующее:

- Выбор режима работы перед запуском программы;
- Переключение между записью статического и динамического жеста;
- Непрерывный мониторинг уже записанных жестов и количества записей по ним;
- Вывод текущего записанного жеста;

Спроектированный вид интерфейса в режиме «обучение» изображён на рис. 35. В верхней части выводится количество записей по каждому из жестов, предоставляя возможность выравнивать количество данных по ним. При нажатии на клавишу

- ‘q’ или Escape – программа завершает свою работу;
- цифру, то есть ‘0’-‘9’ – начинается запись или записывается жест по данной цифре;
- ‘c’ – переключаются режимы одиночной и непрерывной записи статического жеста. В первом режиме координаты ключевых точек записываются только при нажатии на соответствующую цифру.

Последний же позволяет сохранять координаты ключевых непрерывно, а не только при нажатии на цифру. Данный подход позволяет намного быстрее собирать большое количество данных, непрерывно модифицируя и перемещая руку с жестом;

– ‘s’ – включается/выключается запись динамического жеста, которая продолжается до нажатия какой-либо другой клавиши. При последовательной записи динамических жестов удобно пользоваться только цифровыми клавишами, поскольку они одновременно останавливают предыдущую запись и начинают новую.

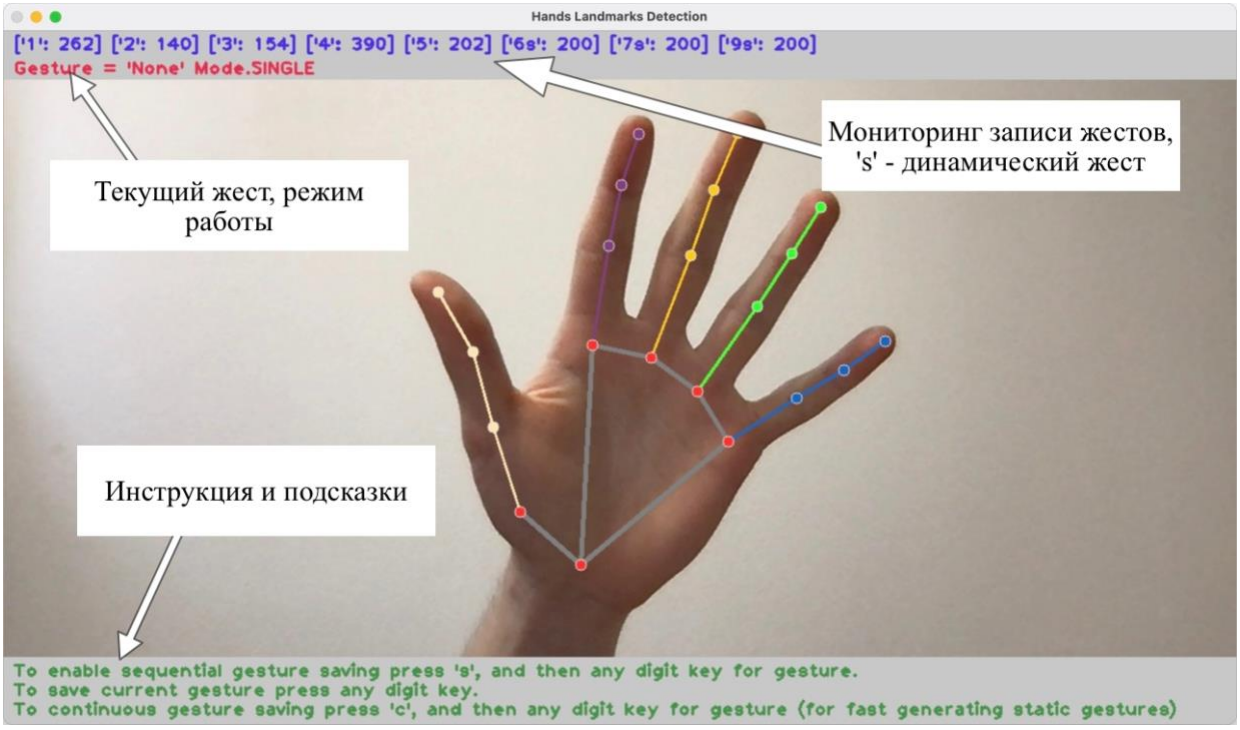


Рис. 35. Вид программы в режиме "обучение.

Таким образом, собрав необходимое количество данных, необходимо на их основе обучить модель нейронной сети распознавать конкретных жест исходя из ключевых точек.

4.3. Подготовка и обучение модели распознавания статических жестов

Подготовка данных для обучения модели заключается в преобразовании координат к необходимому нормализованному виду, как, например, на рис. 36. Исходные значения координат поступают с привязкой к расположению ключевой точки на изображении, то есть в пикселях. Поскольку жест не всегда может находиться в конкретном данном расположении, то необходимо отвязать их от изображения. Для этого все координаты пересчитываются относительно ключевой точки, находящейся на запястье, и нормируются.

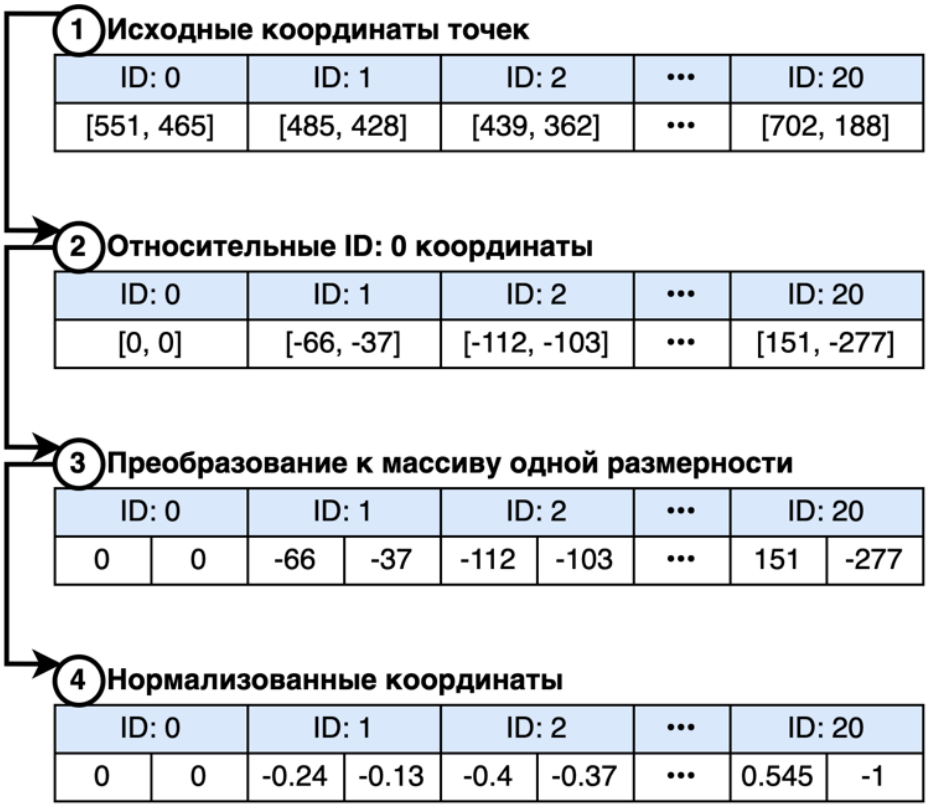


Рис. 36. Преобразование исходных координат.

Затем, набор массивов координат, привязанных к конкретным жестам, подаётся на вход искусственной нейронной сети с предварительно отобранной структурой, показанной на рис. **Error! Reference source not found.**37.

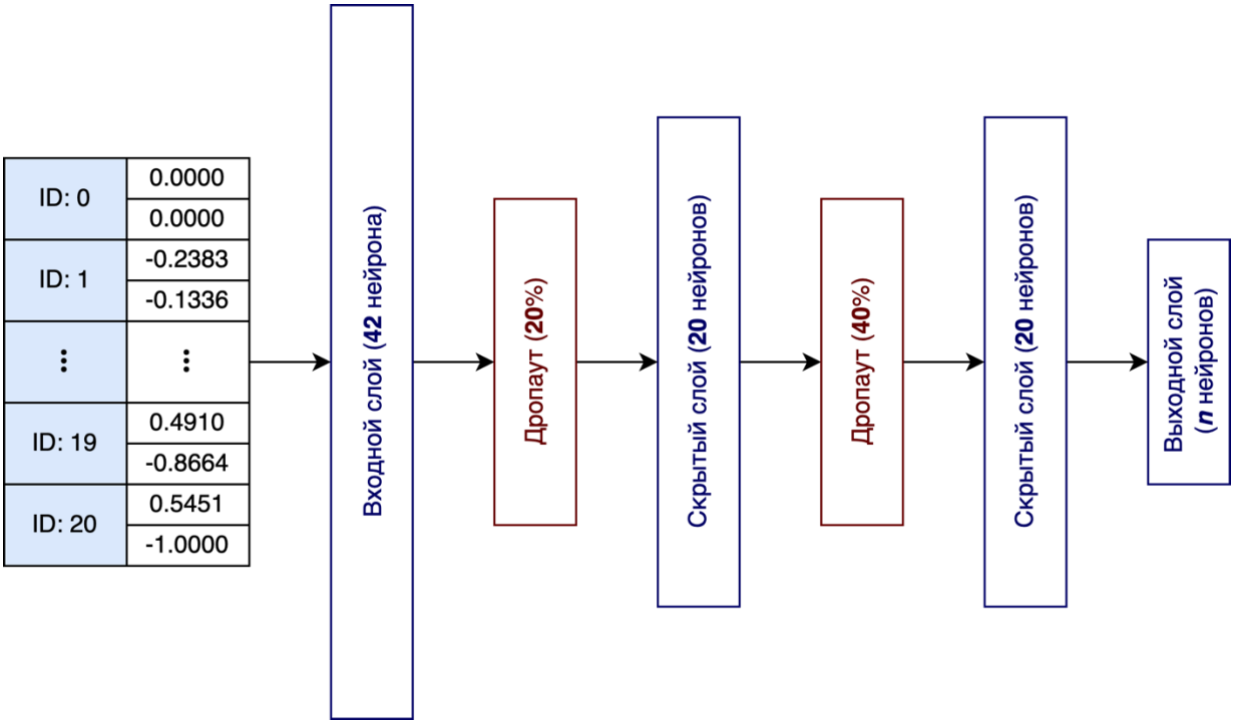


Рис. 37. Структура нейронной сети для распознавания статических жестов.

Более подробно структуру можно записать следующим образом:

- Входной слой (42×1)
- Дропаут 20%
- Скрытый слой (20×1) с функцией активации ReLU:

$$f(x) = \begin{cases} 0, x < 0 \\ x, x \geq 0 \end{cases}$$

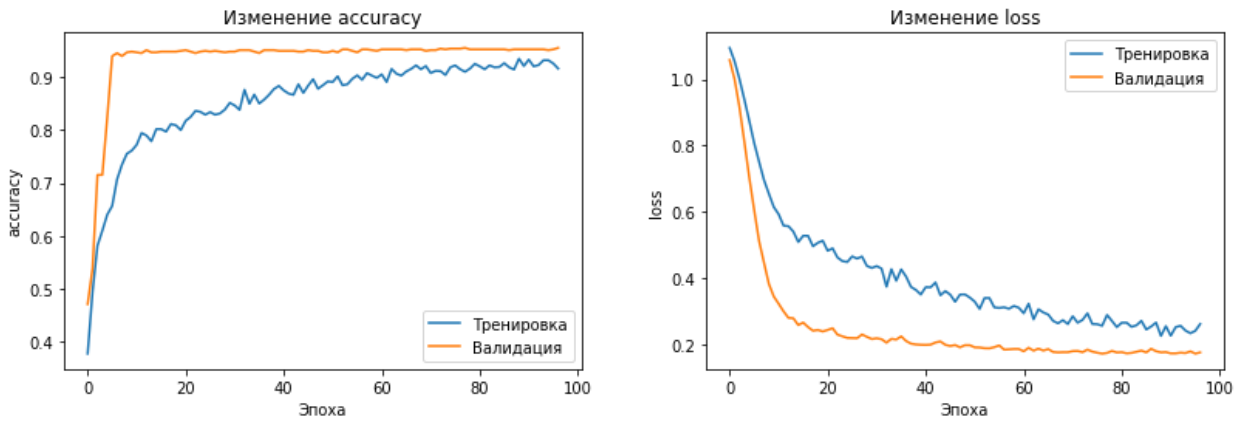
- Дропаут 40%
- Скрытый слой (10×1) с функцией активации ReLU
- Выходной слой $(n \times 1)$ с функцией активации Softmax, где $n -$ количество жестов.

Исключение или *дропаут* – метод регуляризации искусственных нейронных сетей, предназначен для уменьшения переобучения сети за счёт предотвращения сложных коадаптаций отдельных нейронов на тренировочных данных во время обучения, характеризует исключение определённого процента случайных нейронов на разных итерациях во время обучения нейронной сети. Такой приём значительно увеличивает скорость обучения, качество обучения на

тренировочных данных, а также повышает качество предсказаний модели на новых тестовых данных.

В качестве функции оптимизации была выбрана функция Adam. поскольку это один из самых эффективных алгоритмов оптимизации в обучении нейронных сетей. Он сочетает в себе идеи среднеквадратичного распространения корня (RMSProp) и оптимизатора импульса. Вместо того чтобы адаптировать скорость обучения параметров на основе среднего первого момента (среднего значения), как в RMSProp, Adam также использует среднее значение вторых моментов градиентов. В частности, алгоритм вычисляет экспоненциальное скользящее среднее градиента и квадратичный градиент.

В качестве примера были выбраны 3 жеста с вытянутым одним, двумя и тремя пальцами соответственно и заранее сгенерированный набор из 3138 точек для этих жестов. Обучение для него проходило на 97 эпохах (рис. 38). Доля правильных ответов (accuracy) $= 0{,}9656$, потери (loss) $= 0{,}1423$. Основные метрики по каждому классу представлены в таблице 5.



а) б)

Рис. 38. Изменение метрики accuracy (а) и loss (б) на тренировочных и валидирующих данных во время обучения.

Таблица 5.

Показатели метрик классификации статических жестов.

Класс	Точность (precision)	Полнота (recall)	F-мера (f1-score)
1	0,96	1,00	0,98
2	0,95	0,97	0,96
3	1,00	0,91	0,95

На матрице ошибок для задачи распознавания статических жестов (рис. 39) показано соотношение истинных значений (снизу) с ответами нейросети (справа).

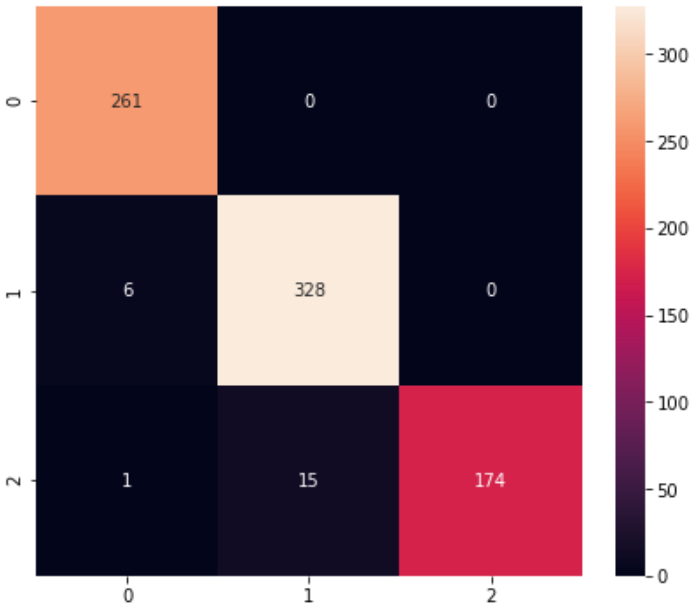


Рис. 39. Матрица ошибок для задачи распознавания статических жестов.

Интуитивно понятной, очевидной и почти неиспользуемой метрикой является accuracy – доля правильных ответов алгоритма:

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Для оценки качества работы алгоритма на каждом из классов по отдельности вводятся метрики precision (точность) и recall (полнота):

$$\text{precision} = \frac{TP}{TP + FP}, \qquad \text{recall} = \frac{TP}{TP + FN}.$$

Precision можно интерпретировать как долю объектов, названных классификатором положительными и при этом действительно являющимися

положительными, а recall показывает, какую долю объектов положительного класса из всех объектов положительного класса нашёл алгоритм. Существует несколько различных способов объединить precision и recall в агрегированный критерий качества. F-мера (в общем случае F_β) – среднее гармоническое precision и recall:

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}.$$

Переменная β в данном случае определяет вес точности в метрике, и при $\beta = 1$ это среднее гармоническое. F-мера достигает максимума при полноте и точности, равными единице, и близка к нулю, если один из аргументов близок к нулю.

Проанализировав результаты, можно сделать вывод, что при тестировании нейросеть показала неплохие результаты, а значит её можно применять в задаче классификации статических жестов по ключевым точкам.

4.4. Подготовка и обучение модели распознавания динамических жестов

По аналогии с подготовкой данных для распознавания статических жестов для обучения модели распознавания динамических жестов с каждым массивом координат точек необходимо проделывать все те же действия, описанные ранее (рис. 40).

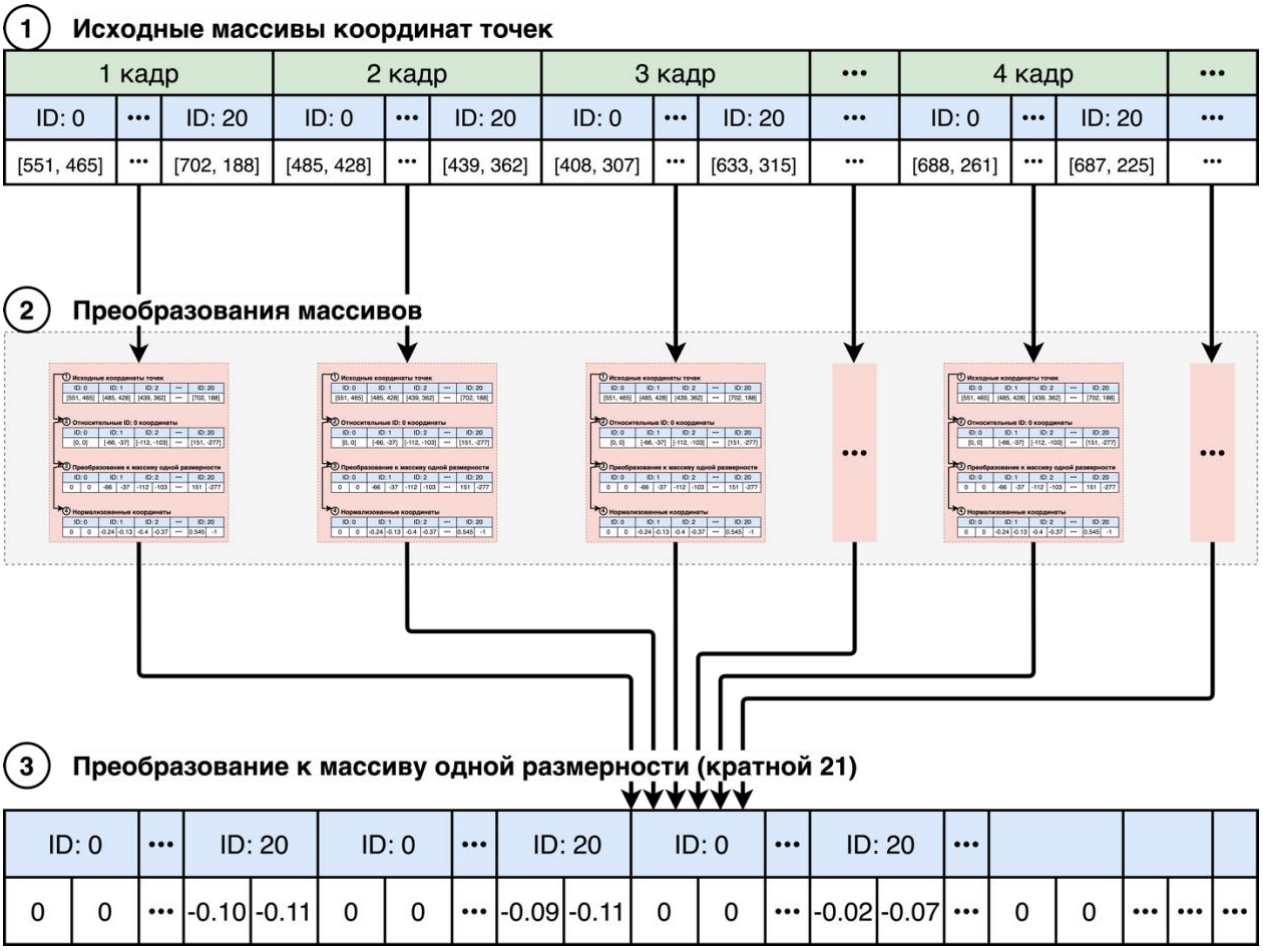


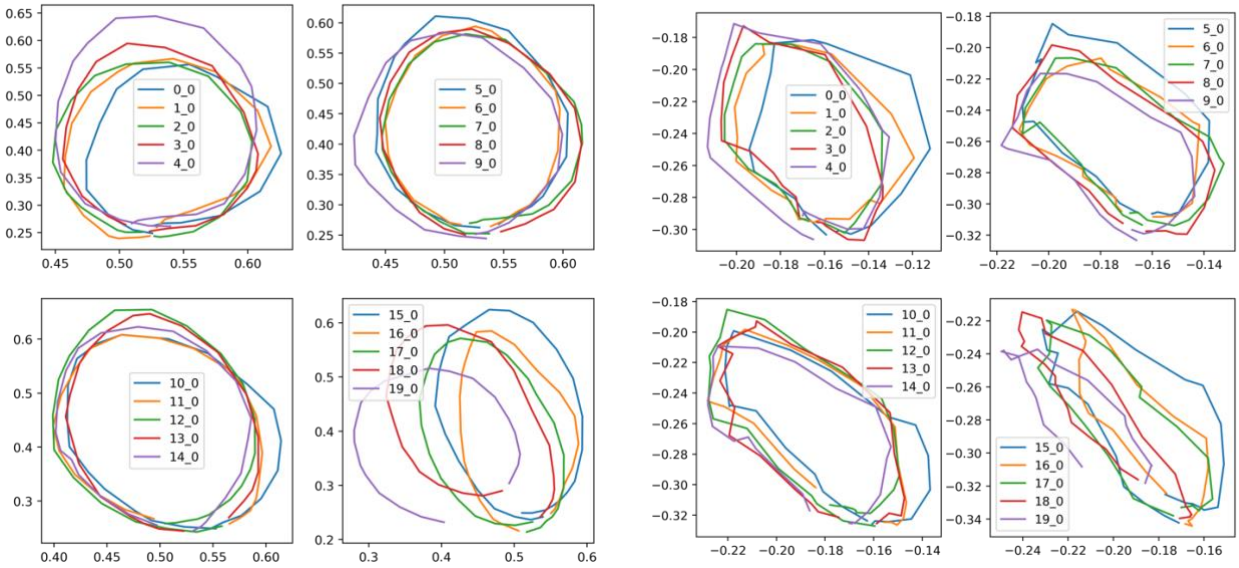
Рис. 40. Подготовка данных для обучения нейросети распознавания динамических жестов.

Поскольку динамический жест является последовательностью координат различной длины, то каждую запись жеста необходимо привести к одному общему размеру. В данном случае можно дополнить каждую из записей незначащими нулями справа до максимального размера среди всех записей.

Собранный набор данных для обучения включает в себя 3 класса динамических жестов: «вращение указательным пальцем по часовой стрелке», «вращение указательным пальцем против часовой стрелки» и «какой-то другой жест».

Необходимость последнего обусловлена тем, что при работе система непрерывно собирает распознанные координаты в буфер и так же непрерывно распознаёт динамический жест. Поскольку в текущий момент времени динамического жеста может не быть, но при этом может быть статический, то

необходимо отделять «полезные» динамические жесты от побочных. Всего было собрано 549 записей, распределение между классами соответственно 174, 187 и 188 записей. Графическое изображение последовательности координат указательного пальца (точка 8 на рис. 33а**Error! Reference source not found.**) в 40 записях изображено на рисунке 41а, а на рисунке 41б изображены те же координаты, приведённые к относительным запястью (точке 0). После сбора и нормализации данных происходит фильтрация сильно длинных последовательностей, чтобы остальные наборы координат дополнялись нулями несущественно.



а) *б)*

*Рис. 41. Графическое изображение примеров последовательности координат указательного пальца (а) относительно запястья (б) (*0 − *«вращение по часовой стрелке»,* 1 −*«против»).*

Формат легенды: <номер записи>_<класс>.

Затем, все эти наборы координат подаются на вход искусственной нейронной сети, проиллюстрированной на рис. 42. В структуре нейросети для классификации динамических жестов присутствует LSTM-слой с 50 выходами, слой дропаута 20%, скрытый слой с 100 нейронами и функцией активации ReLU и выходной слой с количеством нейронов, равным количеству динамических жестов (в текущем случае – 3), и функцией активации Softmax. При обучении будет так же использоваться функция оптимизации Adam.

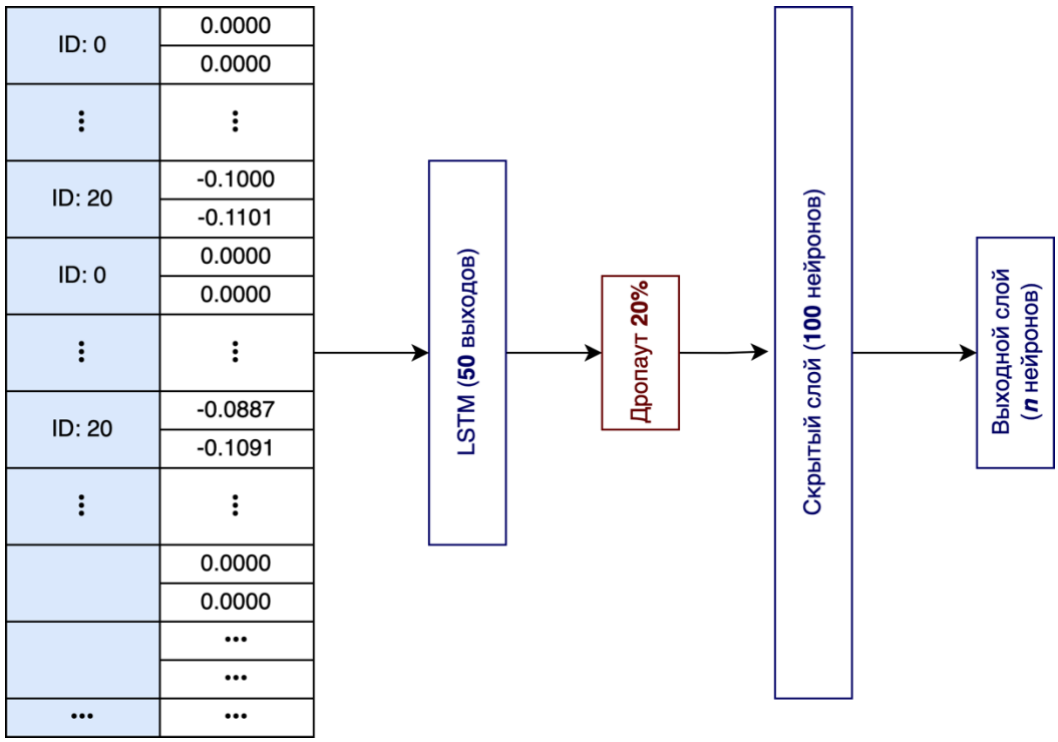
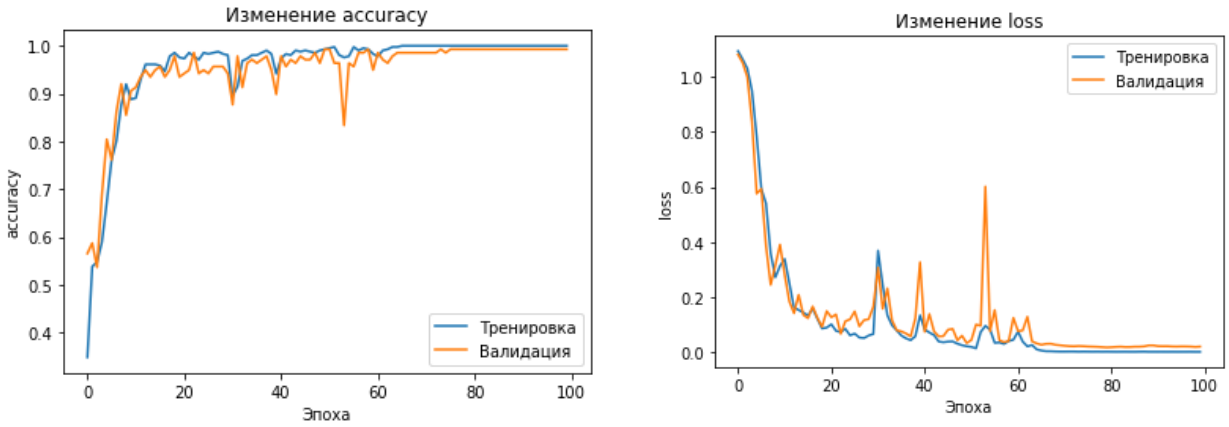


Рис. 42. Структура нейронной сети для распознавания динамических жестов.

Обучение сети проходило на 100 эпохах (рис. 43). Доля правильных ответов (accuracy) составила 0,9928 , а потери (loss) – 0,9928. Основные метрики по каждому классу представлены в таблице 6.



а) б)

Рис. 43. Инициализированный фон (а) и изображение кисти на нём (б).

Таблица 6.

Показатели метрик классификации динамических жестов.

Класс	Точность (precision)	Полнота (recall)	F-мера (f1-score)
1	1,00	1,00	1,00
2	0,98	1,00	0,99
3	1,00	0,98	0,99

Матрица ошибок для задачи распознавания динамических жестов изображена на рисунке 39.

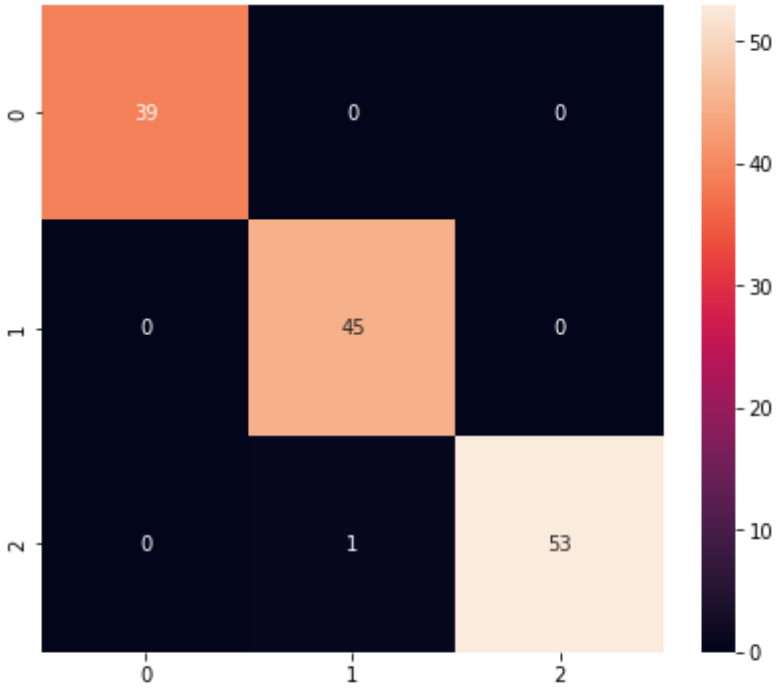


Рис. 44. Матрица ошибок для задачи распознавания динамических жестов.

Таким образом, можно сделать вывод, что нейросеть классификации динамических успешно обучена и её можно применять в задаче классификации динамических жестов.

4.5. Оценка работы интерфейса

В качестве практического применения интерфейса и тестирования распознавания жестов было выбрано управление курсором мыши, соотношения между жестами и действием которого распределено следующим образом:

– Жест 1 (показывается только указательный палец, остальные собраны в кулак) – перемещение курсора, где верхняя точка указательного пальца (точка 8 на рис. 3Зa) отвечает за его координаты на экране;
– Жест 2 (по аналогии с 1 показывается только указательный и средний палец) – нажатие левой кнопки мыши;
– Жест 3 (по аналогии с 1 и 2 показывается указательный, средний и безымянный пальцы) – нажатие правой кнопки мыши;

Интерфейс программы изображён на рис. 45.

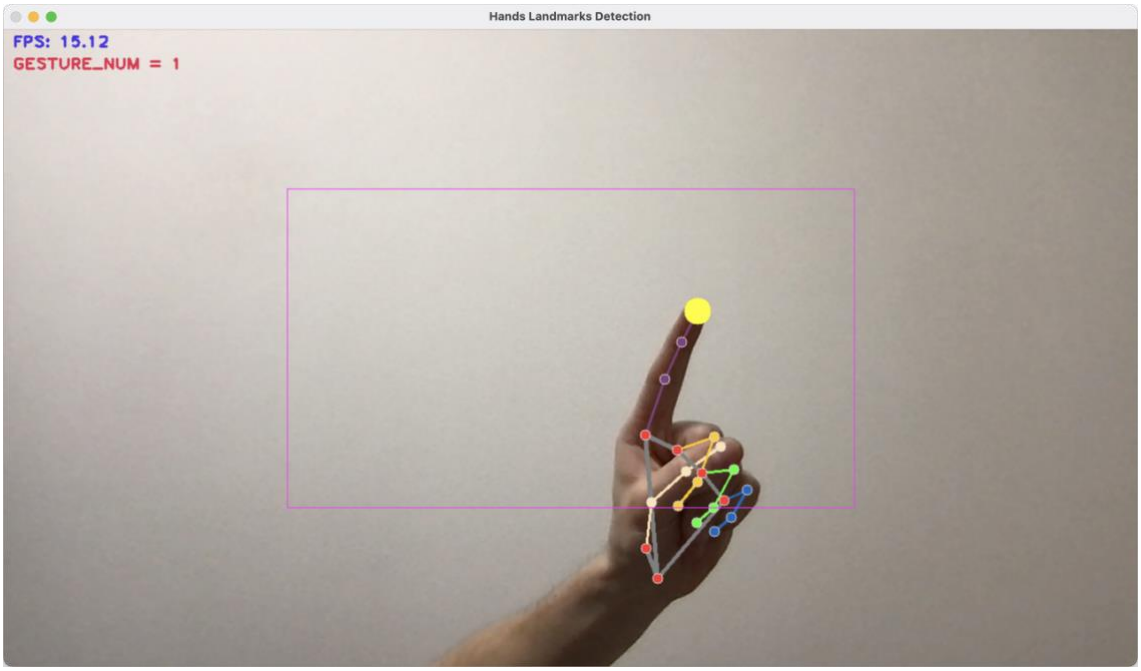


Рис. 45. Интерфейс программы при управлении курсором мыши.

В верхней части отображается количество кадров в секунду (FPS, Frames Per Second) и номер распознанного жеста. В центральной части отображается область проекции экрана, то есть левый верхний угол данной области соответствует левому верхнему углу экрана и так далее. Перемещая указательный палец в данной области, перемещается курсор по экрану. Причина,

по которой была выбрана именно область, а не всё изображение, состоит в том, что в крайних его частях плохо или вовсе не распознаётся жест, поскольку рука не полностью помещается в кадр.

При первом тестировании такой системы был найден недостаток такого подхода – из-за небольшой тряски руки курсор постоянно находится в движении и трудно корректно его позиционировать. В качестве решения был предложен метод *фильтра перемещения в малом*, заключающийся в игнорировании малых перемещений точки в пределах ± 5 пикселей, решивший данную проблему.

При последующих тестированиях алгоритм работал достаточно точно и быстро, в большинстве случаев верно определяя ключевые точки и классифицируя их, выдавая заданный жест. Неточности возникали при больших наклонах руки – это связано прежде всего с неудачными данными для обучения. При увеличении количества различных ситуаций и количества данных алгоритм работал еще точнее.

В качестве основного недостатка данной системы можно выделить неустойчивость при резких передвижениях руки в кадре. Это вызвано тем, что алгоритм предсказывает следующее положение руки в малой окрестности, а при резком перемещении срабатывает детектор, которому необходимо немного больше времени для работы распознавания.

ЗАКЛЮЧЕНИЕ

Детектирование рук и распознавание жестов являются одними из наиболее популярных прикладных задач компьютерного зрения, но до сих пор в их решении имеются проблемы с получением точного результата из-за огромного количество конфигураций кисти.

В результате данной работы были рассмотрены уже существующие решения на основе самых разнообразных алгоритмов и способов, но большинство из них работают с использованием каких-либо других инструментов, что не является удобным для конечного потребителя.

Для решения задачи детектирования кисти были рассмотрены различные подходы, а именно детектирование на основе цветов в разных цветовых моделях (YCbCr и HSV), разделения пикселей изображения на две группы – «фон» и «передняя» (полезная) часть и рекурсивные методы построения модели фона. В результате исследования было получено, что наиболее эффективным способом в статическом изображении является детектирование в цветовой модели HSV, а при наблюдении в динамике, то есть на видеофрагменте, то наилучшие результаты показал метод MoG.

Для построения контура кисти на изображении были исследованы два метода: выделение с помощью оператора Кэнни и топологический структурный анализ изображения с помощью отслеживания границ. Последний позволяет находить точные координаты контура, поэтому является наилучшим среди выбранных.

Для определения конфигурации кисти, было решено использовать её описание в "ключевых" точках. Их поиск был произведён с помощью определения дефектов выпуклости алгоритмами Грэхема, Джарвиса и Киркпатрика. Сравнение данных алгоритмов производилось по времени, поскольку именно оно является решающим в детектировании кисти в реальном времени, при одинаковой точности. Алгоритм Грэхема показал наилучшие

результаты с разницей более чем в 3 раза по сравнению с алгоритмом Киркпатрика и более чем в 25 раз по сравнению с алгоритмом Джарвиса.

На основе исследований была разработана методика детектирования и извлечения конфигурации кисти, показавшая неплохие результаты при несложных движениях кисти человека и однообразном фоне.

Затем были подробно изучены теоретические аспекты актуального способа детектирования объектов на основе искусственных нейронных сетей – Single Shot детектора. В результате сравнения с аналогами был сделан вывод, что данный детектор наилучшим образом подходит для решения требуемой задачи с использованием средств машинного обучения. Также была описана работа современной библиотеки MediaPipe Hands, использующей Single Shot детектор и регрессию для определения 21 ключевой точки на руке.

После этого на основе библиотеки была разработана методика обучения и распознавания статических и динамических жестов для управления курсором мыши. Для этого:

- был разработан программный комплекс для быстрого сбора данных;
- были собраны два набора данных: первый – состоящий из 3000 записей ключевых точек для трёх динамических статических жестов, и второй – 549 записей для трёх классов динамических жестов;
- была спроектирована и обучена искусственная нейронная сеть, классифицирующая *статические* жесты со средней точностью по классам равной 96%;
- была спроектирована и обучена искусственная нейронная сеть с рекуррентным слоем LSTM, классифицирующая *динамические* жесты со средней точностью по классам равной 99%;
- была разработана программа управления курсором мыши для тестирования методики, в результате использования которой

исправлены незначительные недостатки методики и выявлены особо крупные.

По итогу работы следует отметить, что методика, основанная на использовании средств машинного обучения, справилась с решением задачи распознавания намного лучше, чем методика без их использования, но для неё необходимо затратить дополнительное время и средства на обучение искусственной нейросети, а также оптимизацию архитектуры с ростом количества классов жестов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. В.Э. Нагапетян, В.М. Хачумов. Автоматическое преобразование жестов русской ручной азбуки в текстовый вид // Искусственный интеллект и принятие решений. 2013. с. 59-66.
2. Erhan, D., Szegedy, C., Toshev, A., Anguelov, D. Scalable object detection using deep neural networks. CVPR. 2014.
3. Felzenszwalb, P., McAllester, D., Ramanan, D. A discriminatively trained, multiscale, deformable part model. CVPR. 2008.
4. Girshick, R. Fast R-CNN. ICCV. 2015.
5. Girshick, R., Donahue, J., Darrell, T., Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. CVPR. 2014.
6. Hariharan, B., Arbelaez, P., Girshick, R., Malik, J. Hypercolumns for object segmentation and fine-grained localization. CVPR. 2015.
7. He, K., Zhang, X., Ren, S., Sun, J. Spatial pyramid pooling in deep convolutional networks for visual recognition. ECCV. 2014.
8. Howard, A.G. Some improvements on deep convolutional neural network based image classification. 2013.
9. Kadous, Mohammed. Auslan Sign Recognition Using Computers and Gloves. 1998.
10. Liu W., Rabinovich A., Berg A.C. ParseNet: Looking wider to see better. ICLR. 2016.
11. Long, J., Shelhamer, E., Darrell, T. Fully convolutional networks for semantic segmentation. CVPR. 2015.
12. Redmon, J., Divvala, S., Girshick, R., Farhadi, A. You only look once: Unified, real-time object detection. CVPR. 2016.
13. Ren, S., He, K., Girshick, R., Sun, J. Faster R-CNN: Towards real-time object detection with region proposal networks. NIPS. 2015.
14. Robert Y. Wang and Jovan Popovic. Real-Time Hand-Tracking with a Color Glove. ACM Transaction on Graphics. 2009.

15. Sermanet P., Eigen D., Zhang X., Mathieu M., Fergus R., LeCun Y. Overfeat: Integrated recognition, localization and detection using convolutional networks. ICLR, 2014.
16. Uijlings, J.R., van de Sande, K.E., Gevers, T., Smeulders, A.W. Selective search for object recognition. IJCV. 2013.
17. Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg. SSD: Single Shot MultiBox Detector. 2016.
18. Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., Torralba, A. Object detectors emerge in deep scene cnns. ICLR. 2015.
19. Dix A., Finlay J., Abowd G.D., Beale R. Human-Computer Interaction. - Third Edition, Pearson Education Limited: 2004. - p. 857
20. Jiangqin W., Wen G. A FAst Sign Word Recognition Method for Chinese Sign Language // In Proceedings of the Third International Conference on Advances in Multimodal Interfaces (ICMI '00). - Springer-Verlag, London, 2000. - p.599-606
21. Sanna A., Lamberti F., Paravati G., Henao R., Eduardo A., Manuri F. A Kinect-Based Natural Interface for Quadrotor Control // Intelligent Technologies for Interactive Entertainment, Volume 78. Springer Berlin Heidelberg, 2012. - p. 48-56
22. Basilio, Jorge & Torres, Gualberto & Sanchez-Perez, Gabriel & Toscano, Karina & Perez- Meana, Hector. Explicit image detection using YCbCr space color model as skin detection, 2011. - p. 123-128.
23. Otsu N. A threshold selection method from gray-level histograms (англ.) // IEEE Trans. Sys., Man., Cyber. : journal. — 1979. — Vol. 9. — p. 62—66.
24. Kaewtrakulpong P., Bowden R. An improved adaptive background mixture model for real- time tracking with shadow detection // Video-Based Surveillance Systems, pp. 135-144. Springer, 2002.

25. Zivkovic Z., Heijden F. Efficient adaptive density estimation per image pixel for the task of background subtraction // Pattern recognition letters, Vol. 27(7), pp. 773-780, 2006.
26. Satoshi S., Keiich A. 1985. Topological Structural Analysis of Digitized Binary Images by Border Following. Computer vision, graphics, and image processing, 30.
27. Graham R. L. An efficient algorithm for determining the convex hull of a finite planar set //Info. Pro. Lett. – 1972. – T. 1. – C. 132-133.
28. Jarvis R.A. On the identification of the convex hull of a finite set of points in the plane // Information Processing Letters, Volume 2, Issue 1, 1973, p. 18-21.
29. Kirkpatrick, David G.; Seidel, Raimund (1986). "The ultimate planar convex hull algorithm?". SIAM Journal on Computing. 15 (1): 287–299.
30. Fan Z., Valentin B., Andrey V., Andrei T., George S., Chuo-Ling C., Matthias G. MediaPipe Hands: On-device Real-time Hand Tracking. CVPR, 2020.
31. Sepp H., Jurgen S. Long short-term memory. Neural computation 9(8):1735-1780, 1997.

ПРИЛОЖЕНИЯ

Приложение А. Код реализации алгоритма Грэхема.

```
from functools import cmp_to_key

class Point:
    def __init__(self, x=None, y=None):
        self.x = x
        self.y = y

# Расстояние между двумя точками
def dist_sq(p1, p2):
    return ((p1.x - p2.x) * (p1.x - p2.x) +
            (p1.y - p2.y) * (p1.y - p2.y))

# Определение ориентации трёх точек
# (перекрестное произведение векторов pq и qr)
def orientation(p, q, r):
    val = ((q.y - p.y) * (r.x - q.x) -
           (q.x - p.x) * (r.y - q.y))
    if val == 0:
        return 0  # коллинеарны
    elif val > 0:
        return 1  # по часовой
    else:
        return 2  # против часовой

def compare(p1, p2):
    global p0
    o = orientation(p0, p1, p2)
    if o == 0:
        return -1 if dist_sq(p0, p2) >= dist_sq(p0, p1) else 1
    else:
        return -1 if o == 2 else 1
```

```
def convex_hull(input_points: list) -> list:
    points = [Point(point[0], point[1]) for point in input_points]
    n = len(input_points)
    # Находим минимальную точку
    min_y = points[0].y
    min_i = 0
    for i in range(1, n):
        y = points[i].y
        if ((y < min_y) or
             (min_y == y and points[i].x < points[min_i].x)):
            min_y = points[i].y
            min_i = i
    points[0], points[min_i] = points[min_i], points[0]
    # Сортируем массив
    global p0
    p0 = points[0]
    points = sorted(points, key=cmp_to_key(compare))
    m = 1 # Начальный размер нового массива
    # Удаляем лишние точки
    for i in range(1, n):
        while ((I < n - 1) and
           (orientation(p0, points[i], points[i + 1]) == 0)):
            i += 1
        points[m] = points[i]
        m += 1
    if m < 3:
        return None
    S = [points[0], points[1], points[2]]
    for i in range(3, m):
        while ((len(S) > 1) and
             (orientation(S[-2], S[-1], points[i]) != 2)):
            S.pop()
        S.append(points[i])
    return [(p.x, p.y) for p in S]
```

Приложение Б. Код реализации алгоритма Джарвиса.

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

# поиск самой левой нижней точки
def left_index(points):
    minn = 0
    for i in range(1, len(points)):
        if points[i].x < points[minn].x or \
                (points[i].x == points[minn].x and
                 points[i].y > points[minn].y):
            minn = i
    return minn

# аналогично orientation в алгоритме Грэхема
def orientation(p, q, r):
    val = (q.y - p.y) * (r.x - q.x) - \
          (q.x - p.x) * (r.y - q.y)
    if val == 0:
        return 0
    elif val > 0:
        return 1
    return -1

def convex_hull(init_points):
    n = len(init_points)
    if n < 3:
        return
    points = [Point(p[0], p[1]) for p in init_points]
    l = left_index(points)
    hull = []
    p = l
    q = 0
```

```
    while True:
        hull.append(p)
        q = (p + 1) % n
        for i in range(n):
            if (orientation(points[p],
                            points[i], points[q]) == 2):
                q = i
        p = q
        if p == l:
            break
    return [(points[i].x, points[i].y) for i in hull]
```

Приложение В. Код реализации алгоритма Киркпатрика.

```
from collections import namedtuple
from random import randint

Point = namedtuple('Point', 'x y')

def flipped(points):
    return [Point(-point.x, -point.y) for point in points]

def quickselect(ls, index, lo=0, hi=None, depth=0):
    if hi is None:
        hi = len(ls) - 1
    if lo == hi:
        return ls[lo]
    if len(ls) == 0:
        return 0
    pivot = randint(lo, hi)
    ls = list(ls)
    ls[lo], ls[pivot] = ls[pivot], ls[lo]
    cur = lo
    for run in range(lo+1, hi+1):
        if ls[run] < ls[lo]:
            cur += 1
            ls[cur], ls[run] = ls[run], ls[cur]
    ls[cur], ls[lo] = ls[lo], ls[cur]
    if index < cur:
        return quickselect(ls, index, lo, cur-1, depth+1)
    elif index > cur:
        return quickselect(ls, index, cur+1, hi, depth+1)
    else:
        return ls[cur]

def bridge(points, vertical_line):
    candidates = set()
    if len(points) == 2:
```

```
        return tuple(sorted(points))
    pairs = []
    modify_s = set(points)
    while len(modify_s) >= 2:
        pairs += [tuple(sorted([modify_s.pop(), modify_s.pop()]))]
    if len(modify_s) == 1:
        candidates.add(modify_s.pop())
    slopes = []
    for pi, pj in pairs[:]:
        if pi.x == pj.x:
            pairs.remove((pi, pj))
            candidates.add(pi if pi.y > pj.y else pj)
        else:
            slopes += [(pi.y-pj.y)/(pi.x-pj.x)]
    median_index = len(slopes)//2 - (1 if len(slopes) % 2 == 0 else
0)
    median_slope = quickselect(slopes, median_index)
    small = {pairs[i] for i, slope in enumerate(slopes) if slope <
median_slope}
    equal = {pairs[i] for i, slope in enumerate(slopes) if slope ==
median_slope}
    large = {pairs[i] for i, slope in enumerate(slopes) if slope >
median_slope}
    max_slope = max(point.y-median_slope*point.x for point in
points)
    max_set = [point for point in points if \
        point.y - median_slope * point.x == max_slope]
    left = min(max_set)
    right = max(max_set)
    if left.x <= vertical_line < right.x:
        return left, right
    if right.x <= vertical_line:
        candidates |= {point for _, point in large | equal}
        candidates |= {point for pair in small for point in pair}
    if left.x > vertical_line:
        candidates |= {point for point, _ in small | equal}
```

```
        candidates |= {point for pair in large for point in pair}
    return bridge(candidates, vertical_line)

def connect(lower, upper, points):
    if lower == upper:
        return [lower]
    max_left = quickselect(points, len(points)//2-1)
    min_right = quickselect(points, len(points)//2)
    left, right = bridge(points, (max_left.x + min_right.x)/2)
    points_left = {left} | {point for point in points if point.x <
left.x}
    points_right = {right} | {point for point in points if point.x
> right.x}
    return connect(lower, left, points_left) + connect(right, upper,
points_right)

def upper_hull(points):
    lower = min(points)
    lower = max({point for point in points if point.x == lower.x})
    upper = max(points)
    points = {lower, upper} | {p for p in points if lower.x < p.x <
upper.x}
    return connect(lower, upper, points)

def convex_hull(init_points):
    points = [Point(p[0], p[1]) for p in init_points]
    upper = upper_hull(points)
    lower = flipped(upper_hull(flipped(points)))
    if upper[-1] == lower[0]:
        del upper[-1]
    if upper[0] == lower[-1]:
        del lower[-1]
    return upper + lower
```

Приложение Г. Код функции очистки лишних точек дефектов выпуклости.

```
import math
import numpy as np

def squeeze(array):
    array = array[0] if len(array) == 1 else array
    return [p[0] for p in array] if len(array[0]) == 1 else array

# "очистка" выпуклой оболочки до не больше 7 точек
def clear_convex_hull(hull_index, contour):
    distance_point = lambda p1, p2: \
      math.sqrt((p1[0] - p2[0]) ** 2 + (p1[1] - p2[1]) ** 2)
    hull_index, contour = squeeze(hull_index), squeeze(contour)
    ALPHA = 10
    while True:
        boundary = len(hull_index) - 1
        clean_hull, i = [], 0
        while i < boundary:
            clean_hull.append(hull_index[i])
            while i < boundary and \
                    distance_point(contour[hull_index[i]],
                    contour[hull_index[i + 1]]) < ALPHA:
                i += 1
            i += 1
        if len(clean_hull) > 7:
            ALPHA += 1
        else:
            break
    return np.array(clean_hull[:-1]) if \
        distance_point(contour[clean_hull[0]],
        contour[clean_hull[-1]]) < ALPHA else np.array(clean_hull)
```