

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Воронежский государственный технический университет»
(ФГБОУ ВО «ВГТУ», ВГТУ)

Факультет Радиотехники и электроники
Кафедра Радиоэлектронных устройств и систем
Направление подготовки (специальность) 11.05.01 «Радиоэлектронные системы и комплексы»
(код и наименование направления подготовки)
Профиль/программа/направленность Радиоэлектронные системы передачи информации

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

дипломная работа

(вид выпускной квалификационной работы – бакалаврская работа, дипломная работа (проект), магистерская диссертация)

Семенов Владислав Юрьевич

(фамилия, имя, отчество обучающегося)

Тема: Разработка программного обеспечения распознавания психоэмоционального
возбуждения оператора беспилотного летательного аппарата

Состав выпускной квалификационной работы:

Расчетно-пояснительная записка на 117 страницах
Графическая часть на — листах

Расчетно-пояснительная записка к выпускной квалификационной работе:

Заведующий кафедрой	(подпись)	Д.В. Журавлёв)
		(инициалы, фамилия)	
Руководитель	(подпись)	А.И. Сукачев)
		(инициалы, фамилия)	
Консультанты:			
по экономической части	(наименование раздела, подпись)	И.А. Бейнар)
		(инициалы, фамилия)	
по нормам и требованиям ЕСКД	(наименование раздела, подпись)	Е.А. Ищенко)
		(инициалы, фамилия)	
по	(наименование раздела, подпись))
		(инициалы, фамилия)	
по	(наименование раздела, подпись))
		(инициалы, фамилия)	
по	(наименование раздела, подпись))
		(инициалы, фамилия)	
Обучающийся	(подпись)	В.Ю. Семенов)
		(инициалы, фамилия)	

« 18 » января 2024 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «ВГТУ», ВГТУ)

Факультет Радиотехники и электроники
Кафедра Радиоэлектронных устройств и систем
Направление подготовки/специальность 11.05.01 «Радиоэлектронные системы и комплексы»
(код, наименование)

Профиль/программа/направленность Радиоэлектронные системы передачи информации

Утверждаю 29.06.2023
(дата)

Зав.кафедрой РЭУС
Журавлёв Д.В. 
(подпись)

ЗАДАНИЕ
ПО ПОДГОТОВКЕ ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

дипломная работа
дипломная работа (проект), бакалаврская работа, магистерская диссертация
Обучающемуся Семенову Владиславу Юрьевичу
фамилия имя, отчество

1. Тема: Разработка программного обеспечения распознавания психоэмоционального возбуждения оператора беспилотного летательного аппарата
Воронежский государственный технический университет
по заданию какой организации выполняется работа

Приказ № 3-5049 от 22.12.2023 г
утверждена приказом по университету

2. Срок сдачи обучающимся выпускной квалификационной работы:
3. Исходные данные: разрешение экрана 1600 x 900 пикселей, разрешение веб-камеры 1280 x 720 пикселей, максимальная среднеквадратическая ошибка определения координат 5 см.
4. Краткое содержание выпускной квалификационной работы:
Было разработано программное обеспечение, позволяющее определить координаты взгляда пользователя на экране устройства (персонального компьютера) по изображению пользователя, полученного при помощи веб-камеры.

перечень подлежащих разработке в выпускной квалификационной работе вопросов

5. Перечень графического материала (с точным указанием обязательных чертежей по разделам)
61 рисунок, 6 таблиц

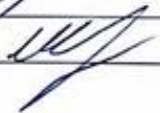
6. Консультанты (с указанием относящихся к ним разделов)
И. А. Бейнар (по экономической части) 
Е. А. Ищенко (по нормам и требованиям ЕСКД) 

График выполнения выпускной квалификационной работы

№ п/п	Наименование разделов выпускной квалификационной работы	Срок выполнения этапов работы
	Введение	04.07.2023
1	Обзор литературы	17.07.2023
2	Разработка системного решения	01.09.2023
3	Сбор и предварительная обработка набора обучающих данных	12.09.2023
4	Разработка модуля распознавания лиц	10.10.2023
5	Разработка модуля определения траектории взгляда, обучение нейронной сети	10.11.2023
6	Разработка модуля анализа движений глаз и пользовательского интерфейса	20.11.2023
7	Технико-экономическое обоснование разработки программного обеспечения распознавания психоэмоционального состояния оператора беспилотного летательного аппарата	30.11.2023
	Заключение	01.12.2023

Руководитель



Дата выдачи задания

29.06.2023

Задание принял к исполнению

29.06.2023

Подпись обучающегося



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Воронежский государственный технический университет»

АННОТАЦИЯ
выпускной квалификационной работы

Обучающегося Семенова Владислава Юрьевича

Тема ВКР: Разработка программного обеспечения распознавания психоэмоционального возбуждения оператора беспилотного летательного аппарата

Характеристика выпускной квалификационной работы

Актуальность исследования

Мониторинг психоэмоционального состояния военнослужащих представляет собой одну из перво-степенных задач с точки зрения обеспечения эффективности Вооруженных Сил. Особенно остро вопрос мониторинга психоэмоционального состояния стоит в подразделениях, осуществляющих эксплуатацию сложной и дорогостоящей техники, в частности, в подразделениях беспилотных летательных аппаратов (БПЛА). Широкое внедрение автоматизированных систем распознавания психоэмоционального возбуждения в данных подразделениях позволит существенно сократить число ошибок, допускаемых операторами БПЛА вследствие неадекватного психоэмоционального состояния, и тем самым сберечь значительное количество материальных и человеческих ресурсов.

Задачи, решаемые в ВКР

В результате выполнения ВКР будет разработано программное обеспечение распознавания психоэмоционального возбуждения человека по его глазодвигательной активности.

Структура ВКР

117 страниц, 61 рисунок, 6 таблиц, 19 источников.

Краткая характеристика полученных результатов (по разделам ВКР)

В разделе 1 проводится выбор и описание объекта исследований – систем мониторинга психоэмоционального состояния человека.

Раздел 2 посвящен анализу существующих методов мониторинга психоэмоционального состояния. В разделе 3 поэтапно описана разработка программного обеспечения распознавания психоэмоционального возбуждения.

В разделе 4 проводится технико-экономическое обоснование разработки программного обеспечения.

Автор ВКР


(подпись)

Семенов Владислав Юрьевич

Ministry of Science and Higher Education of the Russian Federation
Federal State Budgetary Educational Institution higher education
Voronezh State Technical University

ANNOTATION
final qualifying work

Of the training Semenov Vladislav Yurievich

Final qualifying work topic: Development of software for recognizing psycho-emotional arousal of an unmanned aerial vehicle operator

Characteristics of the final qualifying work

The relevance of research

Monitoring the psycho-emotional state of military personnel is one of the primary tasks from the point of view of ensuring the effectiveness of the Armed Forces. The issue of monitoring the psycho-emotional state is especially acute in departments that operate complex and expensive equipment, in particular in the departments of unmanned aerial vehicles (UAVs). The widespread introduction of automated systems for recognizing psycho-emotional arousal in these units will significantly reduce the number of errors made by UAV operators due to an inadequate psycho-emotional state, and thereby save a significant amount of material and human resources.

Tasks solved in the final qualifying work

As a result of the final qualifying work, software will be developed for recognizing a person's psycho-emotional arousal based on his oculomotor activity.

Final qualifying work structure

117 pages, 61 figures, 6 tables, 19 references.

Brief description of the results obtained *(by sections of the final qualifying work)*

Section 1 selects and describes the object of research (systems for monitoring the psycho-emotional state of a person).

Section 2 is devoted to the analysis of existing methods for monitoring psycho-emotional state.

Section 3 describes the development of software for recognizing psycho-emotional arousal step by step.

Section 4 carries out a feasibility study of software development.

Автор ВКР



СЕМЕНОВ ВЛАДИСЛАВ ЮРЬЕВИЧ

СОДЕРЖАНИЕ

Введение	7
1 Выбор и описание объекта исследований	9
2 Обзор существующих решений	11
2.1 Пупиллометрия	11
2.2 Анализ движений глаз	15
2.2.1 Нейронная сеть iTracker	15
2.2.2 Модель оценки усталости Google Research	18
2.3 Метод виброизображений	22
2.4 Анализ движений тела. Мультимодальные системы	26
2.4.1 Алгоритм CABERA	26
2.4.2 Мультимодальная система на основе сетей каскадного внимания	28
3 Описание разработанного программного обеспечения	31
3.1 Разработка системного решения	31
3.2 Поиск и предобработка набора обучающих данных	34
3.2.1 Набор данных OperatorEYEV	34
3.2.2 Набор данных MPIIGaze	41
3.2.3 Сбор и анализ характеристик собственного набора данных	44
3.3 Разработка модуля распознавания лиц	48
3.4 Разработка модуля определения траектории взгляда	56
3.5 Разработка тестовой программы и модуля анализа движения глаз	69
4 Технико-экономическое обоснование разработки программного обеспечения	75
Заключение	79
Список литературы	81
Приложение А Листинг файлов программы для сбора данных	83
Приложение Б Листинг исходного кода для обучения нейронных сетей	89
Приложение В Листинг файлов тестовой программы	103

ВВЕДЕНИЕ

Надлежащее выполнение военнослужащим своих обязанностей зависит не только от имеющихся в его распоряжении материальных средств, но и от психоэмоционального состояния. К настоящему моменту сформирован достаточно широкий перечень психоэмоциональных состояний, оказывающих негативное влияние на несение службы солдатами и офицерами. Данные состояния можно разделить на две группы:

- проявляющиеся в активной форме: предбоевая «лихорадка», состояния стресса, страха, паники и т.д.;
- проявляющиеся в пассивной форме: состояния тревоги, беспокойства, синдрома навязчивости, фрустрации, ригидности и т.д.

Именно в своевременном выявлении и устранении подобных нежелательных состояний и заключается одна из главных задач специалистов по психологической работе в Вооруженных Силах Российской Федерации (ВС РФ).

Необходимо отметить, что автоматизация мониторинга психоэмоционального состояния позволит:

- значительно сократить время, затрачиваемое на процесс установления конкретного психоэмоционального состояния человека;
- повысить точность определения различных показателей, характеризующих психоэмоциональное состояние сотрудника;
- увеличить охват личного состава различных подразделений.

В последние годы область мониторинга психоэмоционального состояния человека получила бурное развитие, превратившись из раздела психиатрии в междисциплинарное направление на стыке медицинских и технических наук. Существует довольно много методов и разработок, позволяющих автоматизировать процесс мониторинга. Однако их соответствие специфике силовых структур требует подробного рассмотрения.

Особенно остро вопрос мониторинга психоэмоционального состояния военнослужащих стоит в подразделениях, осуществляющих видеонаблюдение либо

эксплуатацию сложной или дорогостоящей техники, в частности, беспилотных летательных аппаратов (БПЛА), за счет видеосвязи. Мировые события продемонстрировали, что БПЛА позволяют эффективно решать множество разнородных задач, становясь незаменимым инструментом в боевой работе. В то же время некоторые типы БПЛА имеют высокую ценность как с точки зрения затраченных производственных ресурсов, так и с точки зрения практического применения, и нарушение правил их эксплуатации может повлечь за собой не только ощутимые финансовые убытки, но и серьезное ухудшение тактической или оперативной обстановки. В связи с этим к операторам БПЛА предъявляются особые требования по способности управлять вверенными им аппаратами.

Целью данной выпускной квалификационной работы является разработка программного обеспечения распознавания психоэмоционального возбуждения оператора БПЛА. Достижение поставленной цели возможно при решении следующих задач:

- обзор существующих решений в области автоматизации мониторинга психоэмоционального состояния;
- разработка собственного системного решения;
- разработка вспомогательного (для сбора и предобработки набора обучающих данных) и основного (для распознавания психоэмоционального возбуждения посредством специального интерфейса) программного обеспечения.

1 Выбор и описание объекта исследований

Как было отмечено ранее, автоматизация мониторинга психоэмоционального состояния является относительно популярным направлением научной мысли, которое включает в себя большой набор теоретических методов и конкретных технических разработок.

Следует заметить, что изменение психоэмоционального состояния человека представляет собой комплексный процесс, затрагивающий множество компонентов на различных уровнях жизнедеятельности организма:

- на физиологическом уровне изменение психоэмоционального состояния приводит к изменению артериального давления, частоты пульса и т.д.;
- на моторном уровне изменение психоэмоционального состояния проявляется в изменении мимики, громкости голоса, темпа речи, ритма дыхания и т.д.;
- на эмоциональном уровне изменение психоэмоционального состояния выражается в виде тех или иных переживаний;
- на когнитивном уровне изменение психоэмоционального состояния оказывает влияние на логичность мышления, способность прогнозирования и т.д.;
- на поведенческом уровне изменение психоэмоционального состояния определяет правильность совершаемых действий, их точность, соответствие текущей обстановке и т.д.;
- на коммуникативном уровне изменение психоэмоционального состояния находит проявление в характере общения человека с другими людьми.

Очевидно, большое количество вышеуказанных параметров влечет за собой типологическое многообразие методик мониторинга психоэмоционального состояния и функционирующих на их основе устройств и программного обеспечения.

В рамках данной работы было принято решение сосредоточиться на рассмотрении программных и аппаратных решений, использующих визуальный мониторинг психоэмоционального состояния. Безусловное преимущество разработок этого класса заключается в отсутствии специализированных датчиков: для распознавания нежелательного психоэмоционального состояния достаточно провести анализ

изображений человека с камеры, которая может быть легко интегрирована в различные устройства. Вместе с тем известно, что обработка изображений является нетривиальной задачей, требующей применения алгоритмов глубокого обучения и компьютерного зрения.

Таким образом, объектом исследований являются разработки в области мониторинга психоэмоционального состояния, принцип работы которых заключается в распознавании нежелательных психоэмоциональных состояний посредством анализа получаемых с камеры изображений.

2 Обзор существующих решений

2.1 Пупиллометрия

Пупиллометрией называется методика измерения величины зрачка при освещении различной интенсивности. Данная процедура активно применяется в медицине, в сфере обеспечения безопасности движения [1] и др. Анализ нарушений нормальной реакции зрачков на изменения освещения позволяет надежно выявить отклонения в физиологическом или психическом состоянии человека. Общеизвестно, что при наркотическом опьянении диаметр зрачка будет существенно выше нормы. В случае наркотического и алкогольного опьянения, а также при наличии состояний усталости, монотонии, напряженности и психического пресыщения зрачок значительно медленнее реагирует на свет. Нарушения нормальной реакции зрачков также наблюдаются при некоторых психических расстройствах (например, шизофрении).

Проведение пупиллометрии ручным способом приводит к низкой точности измерений, для увеличения представляется целесообразным использование алгоритмов компьютерного зрения, в частности – сверточных нейронных сетей.

Представляют интерес результаты исследования по автоматизации пупиллометрии, которое проводилось сотрудниками Сегедского университета [2].

Была проведена серия экспериментов на двух группах крыс: контрольной группе Wistar и группе крыс с шизофреноподобным поведением WISKET (общее количество особей – 54). Предварительно крысам были введен седативный препарат. После 10-минутного периода адаптации к темноте осуществлялась 15-секундная запись световой реакции зрачка (PLR) крыс. Животных помещали рядом с камерой, и в левый глаз каждой особи направлялся пучок видимого света высокой интенсивности (примерно 300 кд/м^2 в течение 600 мс). ИК-камера записывала реакции зрачков со скоростью 24 кадра в секунду при инфракрасном освещении. Во время измерений PLR крысы совершили несколько незначительных движений, что сказалось на качестве записываемых видеороликов. Кроме того, у крыс-альбиносов отсутствуют пигменты на теле, в том числе на глазах, что снижает контраст между радужной оболочкой и зрачком. В результате обработки данных производилось построение

пупилограмм (зависимостей относительного диаметра зрачка от времени, в данном случае – от количества кадров). В дальнейшем из пупилограмм извлекались важные с точки зрения классификации признаки (всего 40 признаков), на основе которых происходило обучение решающего дерева. Существенными для решающего дерева оказались следующие признаки: минимальный относительный диаметр; начальный относительный диаметр и средняя скорость повторного расширения, рассчитанная как отношение изменения относительного диаметра зрачка к продолжительности кадра. Полученное по итогам экспериментов решающее дерево представлено на рисунке 1.



Рисунок 1 – Решающее дерево для классификации групп крыс
(0 соответствует контрольной группе, 1 – тестовой группе WISKET)

Решающее дерево показало невысокую точность, равную 71%. В дальнейшем авторы исследования усовершенствовали процесс получения данных о диаметре зрачка, сделав два важных нововведения.

Во-первых, были внесены изменения в процесс освещения зрачка. Вокруг объектива камеры было установлено кольцо ИК-светодиодов, что позволило расположить камеру и освещающие ИК-светодиоды практически на одной оптической оси. Таким образом появилась возможность обнаруживать свет, отраженный от сетчатки, что приводит к т.н. «эффекту яркого зрачка», и камеру можно разместить дальше от

животного, чтобы движения животного не влияли на качество записи. Также была разработана новая встроенная система для формирования различных последовательности импульсов видимого света, которая управлялась с помощью графического пользовательского интерфейса на ПК. Эта система реализует точную синхронизацию времени и регулирует время активации световых раздражителей. В системе использовался RGB-светодиод, который позволял создавать свет различной интенсивности и цвета. Благодаря этим улучшениям на изображениях улучшилось отношение сигнал-шум (зрочок стал лучше различим), а записи стали более надежными и точными. Улучшение качества изображения можно наблюдать на рисунке 2.

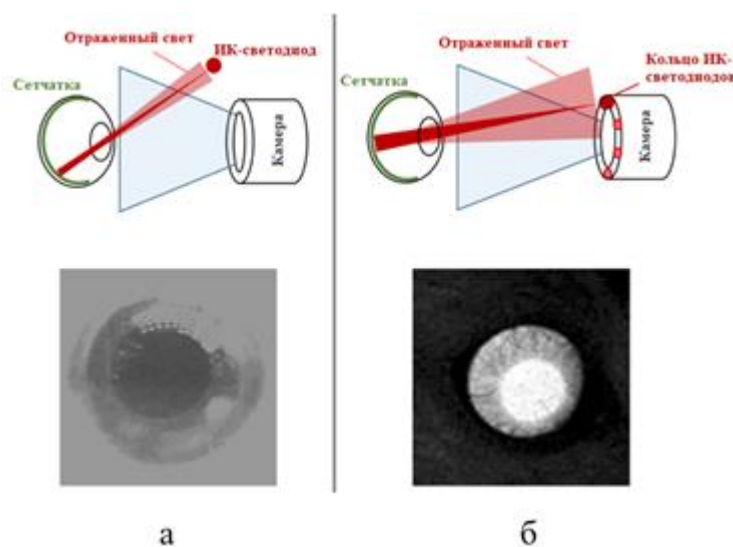


Рисунок 2 – Изображения зрачков: а) до улучшения качества; б) после улучшения качества

Во-вторых, с целью увеличения точности измерения диаметра зрачков была проведена сегментация изображений с использованием нейронной сети. На вход сети поступало полученное с камеры изображение зрачка; выходом сети являлось черно-белое изображение с выделенным зрачком (размер обучающего набора – 2546 изображений; размер дополнительного тестового набора – 329 изображений). На рисунке 3 приведены некоторые экземпляры из обучающего набора данных: верхний ряд изображений соответствует необработанным фотографиям зрачков, нижний ряд – предварительно размеченным изображениям с выделенными зрачками.

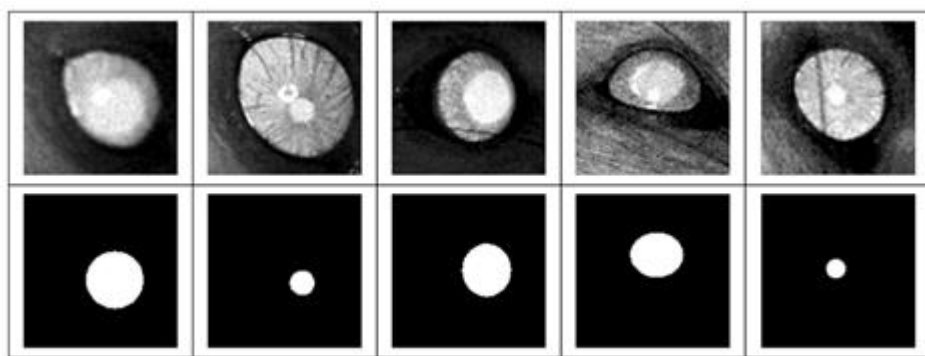


Рисунок 3 – Обучающий набор данных

Архитектура сети представлена на рисунке 4, результаты обучения сети – на рисунке 5.

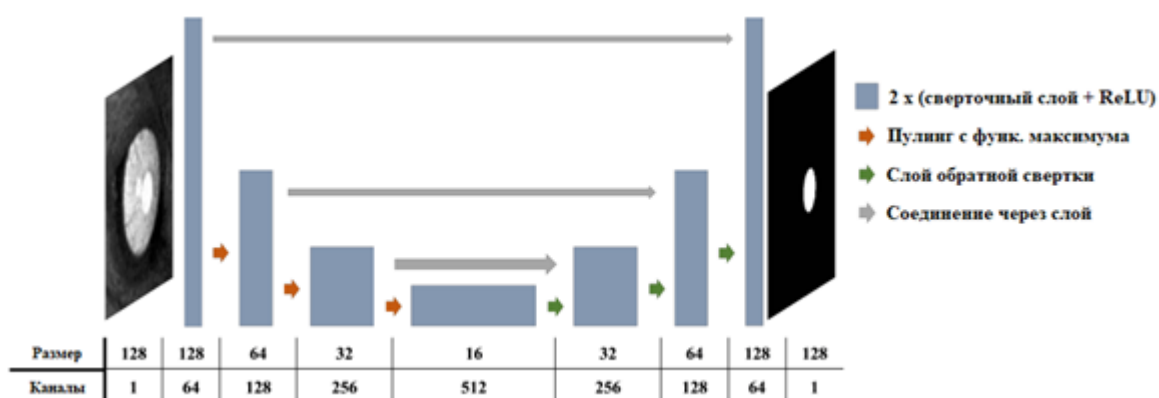


Рисунок 4 – Архитектура нейронной сети для сегментации зрачков

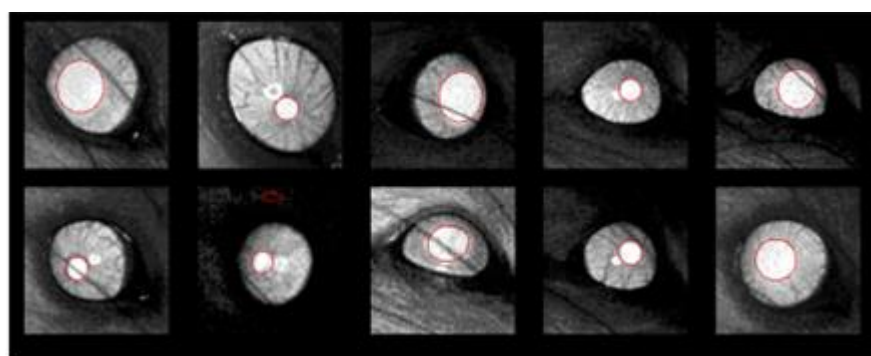


Рисунок 5 – Результаты обучения нейронной сети

Область изображения на рисунке 5, которую нейронная сеть восприняла как зрачок, выделена красной окружностью. Можно заметить, что нейронная сеть не всегда адекватно определяла местоположение зрачка.

Тем не менее благодаря использованию нейронной сети авторам исследования удалось значительно увеличить точность измерения диаметра зрачка (медианная относительная ошибка измерения оказалась равной 4%) и, как следствие, точность классификации крыс по группам (однако следует отметить, что конкретные числовые значения в исследовании не приводятся).

Данная научно-исследовательская работа далека от своего завершения, однако полученные результаты уже обладают определенной практической ценностью. Однако следует принять во внимание тот факт, что применение даже частично автоматизированной пупиллометрии по отношению к операторам БПЛА (и военнослужащим вообще) требует решения некоторых организационных вопросов, связанных, прежде всего, с обеспечением правильного уровня освещения.

2.2 Анализ движений глаз

2.2.1 Нейронная сеть iTracker

Еще один метод распознавания психоэмоционального состояния на основе глаз заключается в анализе направления взгляда. Нарушения движений глаз может свидетельствовать об умственном напряжении и усталости, а также о более тяжелых состояниях, таких как сотрясения мозга, инсульт и т.д.

Еще в 2016 году сотрудниками Массачусетского технологического института и университета Джорджии была предложена архитектура нейронной сети для определения координат взгляда под названием iTracker [3]. Данная архитектура представлена на рисунке 6.

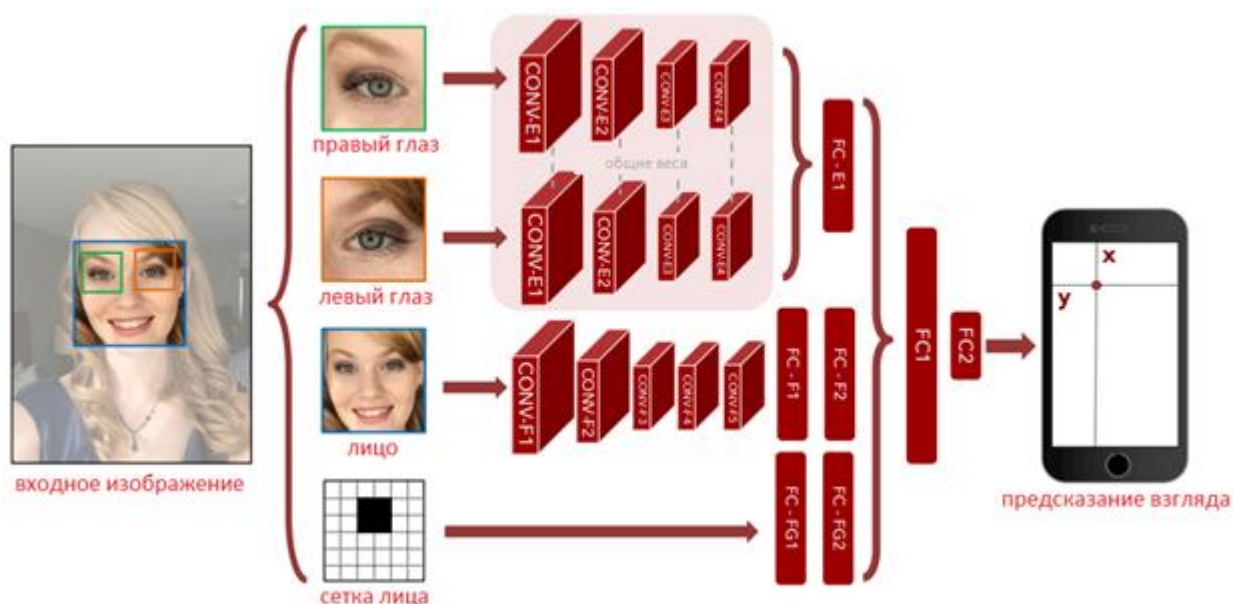


Рисунок 6 – Нейронная сеть для определения координат взгляда iTracker:

Conv – сверточный слой, FC – полносвязный слой

Для эффективного определения координат с помощью нейронной сети iTracker необходимо выполнить предварительную обработку входного изображения. Прежде всего, требуется осуществить детектирование областей изображения, содержащих лицо человека и каждый глаз в отдельности; полученные области приводятся к размерам 224 x 224 пикселей. Дополнительно формируется т.н. сетка лица (размером 25 x 25), которая представляет собой двоичную маску, отражающую положение лица внутри входного изображения. Выходными данными нейронной сети является пара координат (X; Y), соответствующая определенной точке на экране устройства (смартфона либо планшетного компьютера).

Исследователями был подготовлен собственный набор обучающих данных, включающий в себя 2445504 изображений 1471 человека. На рисунке 7 представлено множество всех целевых координат, содержащихся в наборе данных (оно же – пространство возможных предсказаний нейронной сети для сформированного набора данных); положение камеры, выполнявшей съемку людей в процессе сбора данных, приведено к нулевым координатам.

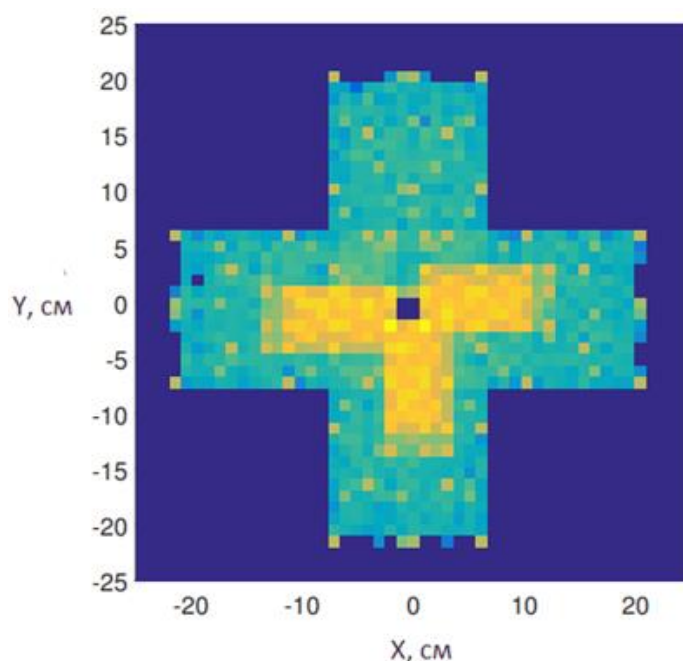


Рисунок 7 – Пространство всех возможных предсказаний нейронной сети для сформированного набора данных

По утверждению исследователей, на подготовительном этапе обучения сети iTracker выполнить успешное детектирование лица и глаз удалось для 1490959 изображений из набора данных.

Для оценки точности определения координат были использованы 2 метрики: ошибка определения координат для единичного изображения (в сантиметрах) и ошибка определения координат для потока изображений, относящихся к одному человеку при взгляде на одну и ту же точку экрана (т.н. точечная ошибка; также в сантиметрах).

Наименьшие значения ошибки были получены для нейронной сети с тонкой настройкой на тип и ориентацию устройства:

- для смартфонов – ошибка для единичного изображения 1,71 см, точечная ошибка – 1,53 см;
- для планшетных компьютеров – ошибка для единичного изображения 2,53 см, точечная ошибка 2,38 см.

Улучшить полученные результаты позволила калибровка, которая заключалась в переобучении iTracker на определенном количестве фиксированных точек с

последующих обучением регрессии опорных векторов на выходных признаках слоя FC1 нейронной сети. Ниже приведены значения ошибок при калибровке на 13 точках:

- для смартфонов – ошибка для единичного изображения 1,34 см, точечная ошибка – 1,04 см;

- для планшетных компьютеров – ошибка для единичного изображения 2,12 см, точечная ошибка 1,69 см.

При анализе архитектуры нейронной сети обращает на себя внимание некоторая избыточность входных данных: помимо целого изображения лица, iTracker также обрабатывает двоичную маску и отдельные изображения глаз. Также важную роль в повышении качества работы iTracker играет увеличение количества человек, участвующих в сборе данных (рисунок 8).

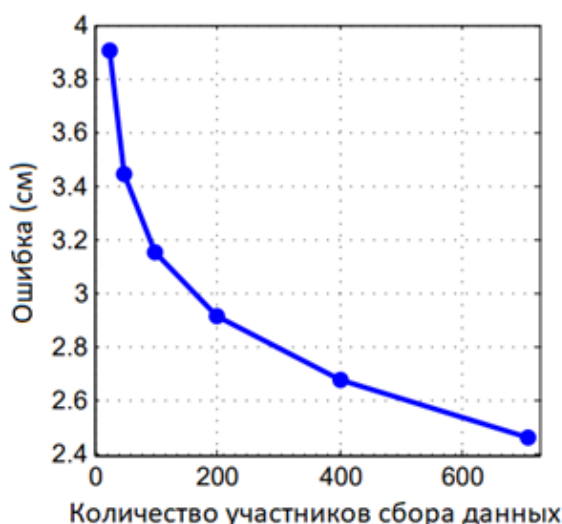


Рисунок 8 – Влияние количества участников сбора данных на ошибку определения координат

Увеличение количества участников сбора данных позволило значительно уменьшить ошибку определения координат.

2.2.2 Модель оценки усталости Google Research

Значительно упростить архитектуру нейронной сети для определения координат взгляда удалось сотрудникам Google Research [4].

В ходе экспериментов исследователями была спроектирована нейронная сеть для отслеживания взгляда посредством фронтальной камеры смартфона. Архитектура нейронной сети изображена на рисунке 9.

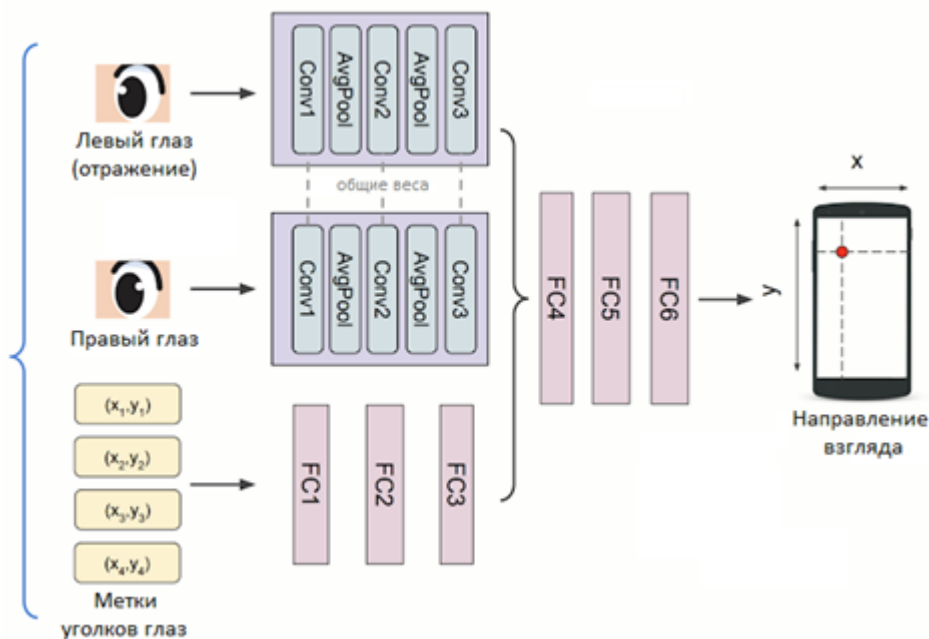


Рисунок 9 – Архитектура нейронной сети для отслеживания направления взгляда: Conv – сверточный слой, AvgPool – пулинг с функцией вычисления среднего, FC – полносвязный слой

Фронтальная камера формировала фотографию лица с метками в уголках глаз, после чего производилась обрезка фотографии до областей размером 128 x 128 пикселей, содержащих правый и левый глаз. Обрезанные изображения глаз и координаты меток служили входными данными для нейронной сети. Выходными данными сети также являлись координаты вида (X; Y), соответствовавшие определенной точке на экране смартфона. Как и в случае iTracker, точность определения координат (X; Y) можно повысить путем персонализации и тонкой настройки под нужды конкретного пользователя.

Авторами исследования утверждается, что точность отслеживания взгляда при помощи их разработки сопоставима с точностью специализированных ай-трекеров (ошибка определения координат нейронной сетью с персонализацией составила

около 0,46 см; при расстоянии между лицом и смартфоном 25-40 см эта величина соответствует угловой ошибке 0,6-1°).

Исследователи также смогли воспроизвести результаты предыдущих работ по изучению движения глаз в нейробиологии и психологии, включая стандартные глазодвигательные задачи (для проверки основных зрительных функций мозга) и задачи на естественное понимание изображений.

Разработанная нейронная сеть в дальнейшем была использована для выявления отклонений в движении глаз при накоплении умственной усталости [5]. Группе испытуемых предлагалось в течение определенного времени сопровождать взглядом движущийся по круговой траектории на экране смартфона объект. По мере увеличения усталости испытуемых траектория взгляда становилась все более беспорядочной (рисунок 10).



Рисунок 10 – Траектория взгляда: а) при отсутствии умственной усталости; б) при наличии умственной усталости

Авторами исследования также была создана модель оценки усталости на основе машины опорных векторов. В качестве входных данных модели использовались нормализованные признаки, связанные с выполнением задач (точность и среднее время обнаружения целевого объекта) и отслеживаемым взглядом (энтропия взгляда, рассчитанная как энтропия Шеннона по тепловым картам взгляда; математическое ожидание и стандартное отклонение ошибки взгляда; математическое ожидание и стандартное отклонение ошибки определения координат); выходными данными модели являлась вероятность того, что участник эксперимента находится в утомленном

состоянии. выходными данными модели являлась вероятность того, что участник эксперимента находится в утомленном состоянии. По завершении каждого блока заданий испытуемым также предлагалось пройти опрос по методике установления уровня утомления (BFI), в ходе которого требовалось указать уровень своей усталости. На рисунке 11 приведен график прогрессии усталости после выполнения нескольких блоков однотипных заданий по отслеживанию объекта; в качестве предсказаний модели на графике показана средняя оценка для всех участников эксперимента.

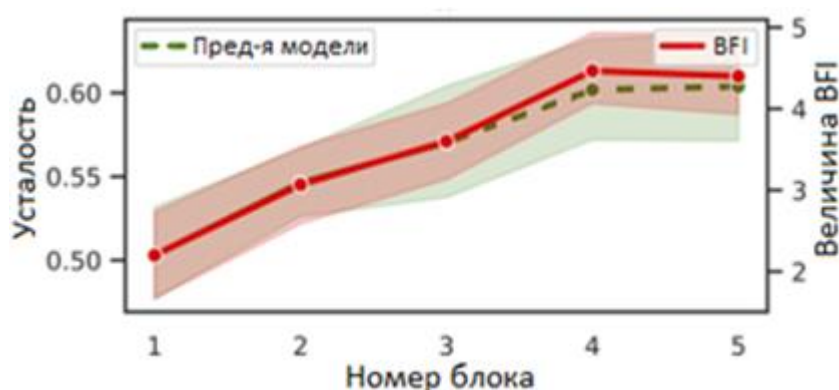


Рисунок 11 – График прогрессии усталости: пунктирная линия – предсказания модели оценки усталости; сплошная линия – оценка усталости согласно результатам опросов

Можно заметить, что предсказания модели оценки усталости имеют высокую степень совпадения с результатами, полученными в ходе опросов. Таким образом, исследование сотрудников Google Research представляет большой интерес с точки зрения распознавания отрицательных психоэмоциональных состояний (в данном случае – состояния усталости), позволяя автоматизировать не только сбор данных, но и их предварительный анализ.

Тем не менее, качественное распознавание усталости возможно только при наличии некоей эталонной траектории движения взгляда, относительно которой и анализируются отклонения. В случае, когда оператор отслеживает быстро движущийся объект или работает с несколькими мониторами, траектория его взгляда будет довольно беспорядочной, что порождает существенные проблемы при мониторинге психоэмоционального состояния в таких ситуациях.

2.3 Метод виброизображений

Широкое распространение в России получила методика анализа психоэмоционального состояния человека при помощи т.н. виброизображений [6]. Автором метода и разработчиком соответствующих программных продуктов является российская компания «Элсис».

Виброизображение – это изображение, которое отражает параметры движения и вибрации того или иного объекта. Виброизображения могут быть получены путем обработки видеоизображений. На начальном этапе осуществляется запись объекта на видео. Впоследствии проводится обработка полученного видеоряда, в ходе которой для каждого пикселя объекта определяются параметры вибрации (частота и амплитуда). На итоговое виброизображение наносится интегральная информация о перемещении пикселей.

Утверждается, что существенных результатов по распознаванию психоэмоционального состояния можно достичь, формируя и анализируя виброизображения головы человека. Данный подход основан на предположении, согласно которому каждому эмоциональному состоянию соответствует определенный уровень непроизвольных механических колебаний (вибраций) головы, вызванных работой вестибулярной системы, причем параметры вибрации стабильны во времени и изменяются только после изменения психоэмоционального состояния при условии, что человек не совершает дополнительных движений, т.е. спокойно сидит либо стоит. Зная частоту и амплитуду вибрации, можно однозначно распознать текущее психоэмоциональное состояние, психофизиологические параметры, уровень работоспособности и т.д.

Авторы методики проводили исследования с двумя типами виброизображений: внутренними и внешними. Внутреннее виброизображение представляет собой псевдоцветное изображение лица, на котором отражено распределение амплитуд либо частот колебаний головы; в зависимости от этого внутренние изображения подразделяются на амплитудные и частотные (рисунок 12).

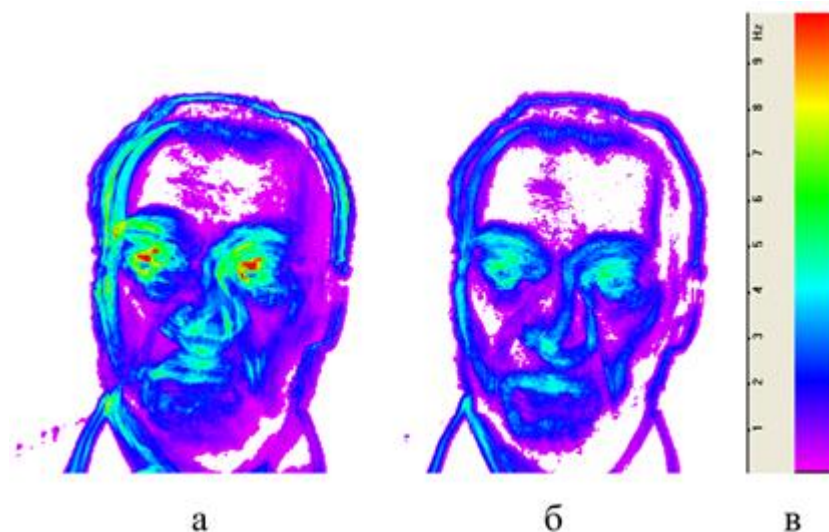


Рисунок 12 – Внутреннее виброизображение: а) амплитудное; б) частотное; в) псевдоцветовая шкала распределения частот

Внешнее виброизображение (рисунок 13) формируется вокруг головы и соответствует ее колебаниям в пространстве с учетом их частоты.

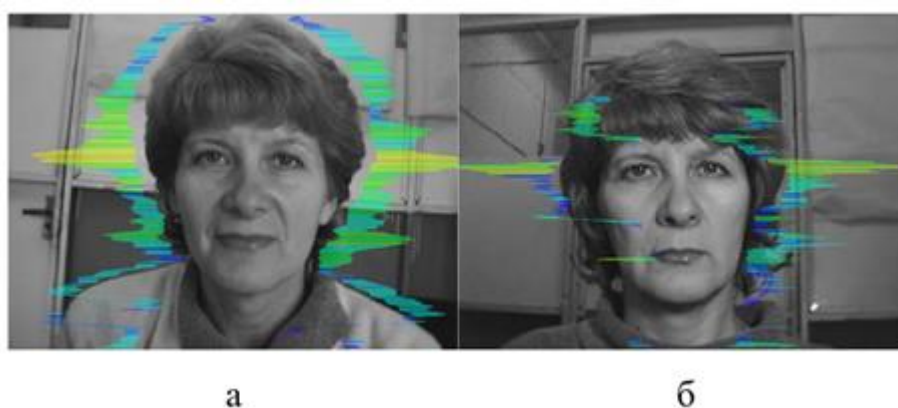


Рисунок 13 – Внешнее виброизображение человека: а) в спокойном состоянии; б) в состоянии стресса

Виброизображения данного типа состоят из совокупности строк, для каждой из которых вычислена максимальная частота и средняя амплитуда вибрации. Указанные выше параметры отображаются в виде псевдоцветных полос: цвет полосы соответствует частоте вибрации (при этом цвет для внешнего изображения кодируется той же шкалой, что и для внутреннего), а длина полосы – средней амплитуде вибрации.

Авторами сделан вывод о том, что внешние виброизображения более наглядно отображают психоэмоциональное состояние человека.

На рисунках 13, а и 13, б приведены примеры внешних виброизображений человека в нормальном и стрессовом состоянии соответственно. Можно заметить, что нормальное состояние человека характеризуется более равномерным внешним виброизображением, в то время как для стрессового состояния характерны большая пространственная и цветовая неравномерность внешнего виброизображения.

Возможна конвертация виброизображений в другие формы представления данных. На рисунке 14 приведена частотная гистограмма движения головы для всех точек изображения за временной период, равный 20 секундам; в течение эксперимента человек находился в 2 экстремальных состояниях: сильной усталости и ярости.

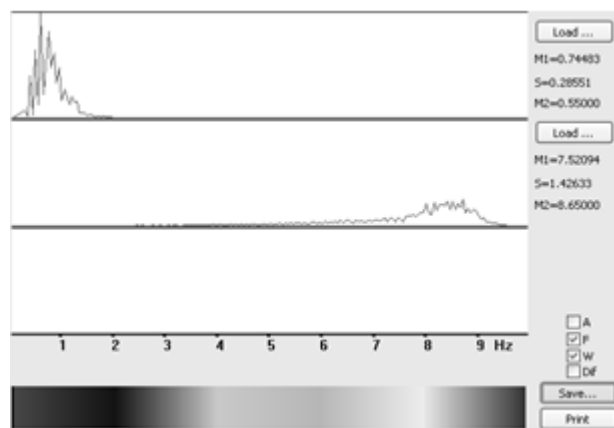


Рисунок 14 – Частотная гистограмма вибраций для человека в экстремальном психофизиологическом состоянии, верхний график характеризует сильную усталость, нижний график – ярость

На рисунке 15 представлен спектр сигнала виброизображения для человека в нормальном (спокойном) и тревожном состоянии.

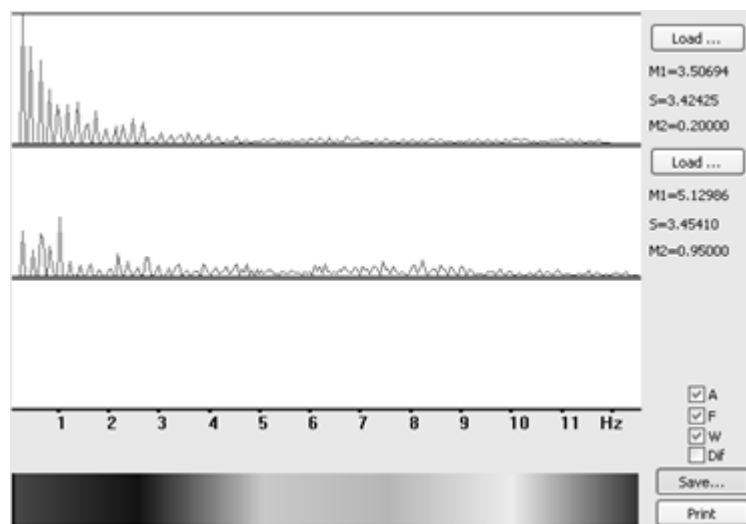


Рисунок 15 – Спектр сигнала виброизображения для человека в нормальном состоянии (верхний график) и в тревожном состоянии (нижний график)

По утверждению авторов, системы виброизображений позволяют решать множество задач, среди которых:

- обеспечение безопасности в местах массового скопления людей, выявление в толпе субъектов, потенциально опасных для окружающих, постоянная психоэмоциональная идентификация большого потока людей;
- детекция лжи, профайлинг, оценка способностей и психоэмоциональной устойчивости кандидатов при наборе персонала;
- диагностика злокачественных новообразований и др. заболеваний;
- проведение специальных исследований в медицине.

Известно, что системы виброизображений используются в Московском метрополитене и аэропорте Пулково. Также технология виброизображений применялась для распознавания психоэмоционального возбуждения посетителей Олимпийских и Паралимпийских игр в Сочи. Некоторое распространение виброизображения получили в странах Восточной Азии.

Использование произвольных движений головы в качестве маркера изменений психоэмоционального состояния (при учете индивидуальных особенностей человека) представляется относительно понятным и легко реализуемым методом

мониторинга, хотя некоторые исследователи сомневаются в его эффективности или считают его способным нанести вред общественной безопасности [7].

2.4 Анализ движений тела. Мультимодальные системы

2.4.1 Алгоритм CABERA

Для распознавания психоэмоциональных состояний представляется возможным использовать анализ движений тела. В этом направлении ведутся многочисленные исследования, однако акцент в них, как правило, сделан именно на эмоциональной составляющей. Среди множества разработок выделяется класс комплексных (мультимодальных) систем, позволяющих производить одновременную обработку лица человека, его позы и окружающей обстановки с целью получения наиболее полной информации об испытываемых эмоциях. К такому классу систем принадлежит CABERA (Complex Activity Based Emotion Recognition Algorithm) – комплексный алгоритм распознавания эмоций на основе двигательной активности [8].

Система CABERA предназначена для определения 6 основных типов эмоций (страх, грусть, счастье, гнев, удивление, отвращение), испытываемых человеком в ходе совершения тех или иных видов деятельности. CABERA была спроектирована по четырехуровневой схеме, предназначенной для анализа индивидуального и межличностного поведения. Первый уровень системы обрабатывает информацию о движениях тела, полученную при помощи датчиков Kinect (данные о позе скелета, RGB-кадры). Датчики Kinect, используемые в системе CABERA, позволяют определить местоположение 20 точек человеческого тела (рисунок 16).

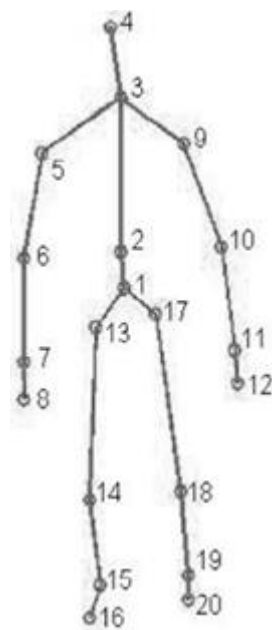


Рисунок 16 – Точки человеческого тела

Второй уровень выполняет классификацию движений на движения верхней и нижней части тела, а также анализирует первичную информацию о социальном взаимодействии. Третий уровень системы рассматривает движения частей тела в контексте окружающей обстановки. Наконец, четвертый уровень более полно анализирует социальное взаимодействие, обобщая информацию для двух человек, находящихся в одном кадре.

Потенциально CABERA может обрабатывать звуковую информацию, однако дополнительных сведений в исследовании не приводится.

По утверждению разработчиков, алгоритм CABERA, испытанный в рамках комплекса «умный дом», позволил установить достоверную взаимосвязь между типом испытываемой эмоции и типом выполняемых человеком движений (движения верхней либо нижней части тела), а также рассчитать вероятность появления положительной эмоции при совершении сложносоставных повседневных действий (готовка, просмотр телевизора и т.д.) с учетом более простых (атомарных) действий и атрибутов окружающей обстановки.

Авторы исследования заявляют, что CABERA может использоваться при выявлении аффективных состояний, при анализе поведения и отслеживании психического состояния пациентов в системах реабилитации и т.д. Вместе с тем показатели

качества SABERA, приведенные в работе, не позволяют сделать однозначный вывод об эффективности данного алгоритма.

2.4.2 Мультимодальная система на основе сетей каскадного внимания

Заслуживает упоминания мультимодальная система, предназначенная для определения эмоций, испытываемых на уровне коллектива (т.н. групповых эмоций) [9]. Данная система представляет собой ансамбль сверточных нейронных сетей, проводящий многоэтапную обработку изображений. Система состоит из следующих модулей:

- модуль обработки лиц;
- модуль обработки позы;
- модуль обработки изображения в целом.

Структура вышеописанной мультимодальной системы приведена на рисунке 17.

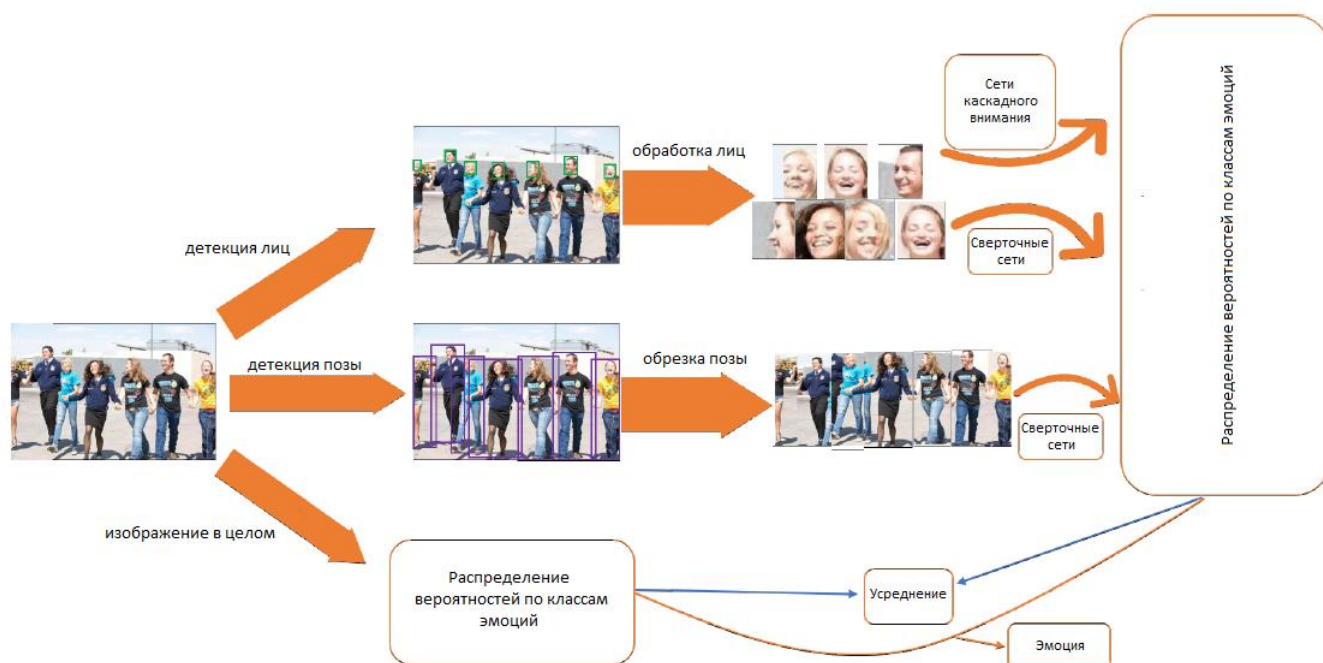


Рисунок 17 – Мультимодальная система на основе сетей каскадного внимания

В каждом модуле осуществляется расчет распределения вероятностей по классам эмоций (всего задано 3 класса: положительная эмоция, нейтральная эмоция, отрицательная эмоция), после чего происходит вычисление средних значений вероятностей и предсказание наиболее вероятной эмоции.

В данной мультимодальной системе используются стандартные архитектуры нейронных сетей. Исключение представляет модуль обработки лиц, где обработка изображений выполняется по 2 параллельным потокам:

- через последовательность стандартных сверточных нейронных сетей, которые анализируют каждое лицо в отдельности;
- через сети каскадного внимания, которые анализируют несколько лиц одновременно.

На выходе модуля обработки лиц формируется 2 распределения вероятностей (по числу потоков обработки).

На рисунке 18 представлена архитектура сети каскадного внимания.

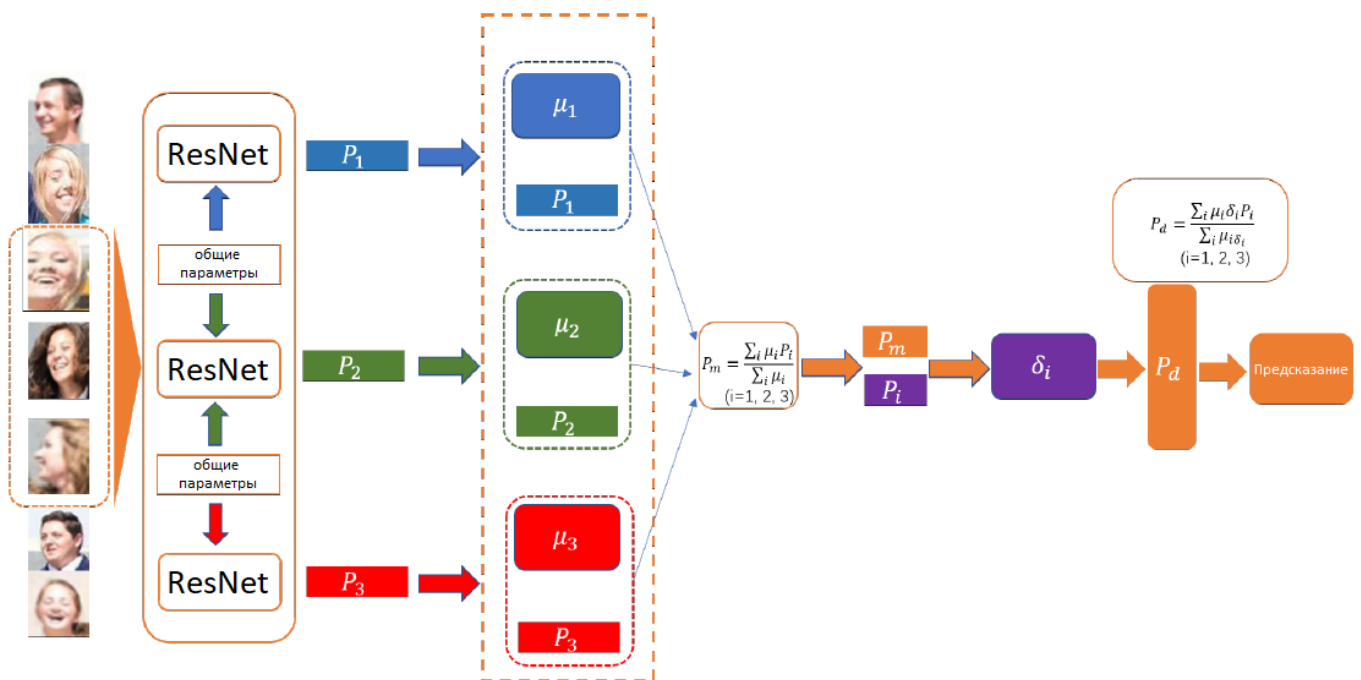


Рисунок 18 – Сеть каскадного внимания

Сеть каскадного внимания выполняет двухэтапную обработку лиц с целью установления важности того или иного лица для определения групповой эмоции. На

первом этапе стандартными нейронными сетями (в данном случае – ResNet18 с выходным усредняющим пулингом) извлекаются числовые признаки лиц, которые впоследствии передаются в полносвязные слои (слои «само-внимания», или self-attention) с одномерным выходом. В дальнейшем выполняется расчет взвешенного среднего для всех анализируемых лиц

$$P_m = \frac{\sum \mu_i P_i}{\sum \mu_i}, \quad (1)$$

где P_i – совокупность числовых признаков i -го лица;

μ_i – выходное значение слоя «само-внимания» для i -го лица.

На втором этапе происходит конкатенация взвешенного среднего P_m с признаками лиц P_i и передача полученной совокупности чисел в новый полносвязный слой с одномерным выходом (слой контекстного внимания). Итоговая числовая характеристика анализируемых лиц, на основе которой сеть каскадного внимания определяет распределение вероятностей групповых эмоций, рассчитывается по формуле

$$P_d = \frac{\sum \mu_i \delta_i P_i}{\sum \mu_i \delta_i}, \quad (2)$$

где δ_i – выходное значение слоя контекстного внимания для i -го лица.

Рассмотренная мультимодальная система была протестирована в рамках соревнования EmotiW 2018. Набор данных был разбит на тренировочную, валидационную и тестовую выборки, содержащие 9136, 4346 и 3011 изображений соответственно. Наибольшая точность, которой удалось достичь на тестовой выборке, составила около 67,48%. Данное обстоятельство позволило авторам исследования занять второе место в зачете систем распознавания групповых эмоций.

Возможность применения мультимодальных систем (как описанной выше, так и любых других) в области мониторинга психоэмоционального состояния требует подробного изучения и с целью определения условий, при которых данные разработки продемонстрируют максимальную эффективность.

3 Описание разработанного программного обеспечения

3.1 Разработка системного решения

В таблице 1 представлена краткая характеристика каждого из рассмотренных ранее методов мониторинга психоэмоционального состояния.

Таблица 1 – Методы мониторинга психоэмоционального состояния человека

Метод мониторинга	Достоинства	Недостатки
Пупиллометрия, анализ движений глаз	Признанная эффективность метода при диагностике патологических состояний и нарушений психики	Невозможность качественной дифференциации психоэмоциональных состояний. В случае пупиллометрии – организационные трудности при мониторинге. В случае анализа движений глаз – отсутствие эталонной траектории движения взгляда видеооператора
Метод виброизображений	Универсальность метода. Возможность распознавания широкого спектра состояний (психические, эмоциональные состояния, наличие заболеваний и т.д.) и их дифференциации	Непрозрачность технологии. Отсутствие независимых проверок качества и работоспособности
Анализ движений тела, мультимодальные системы	Учет различных аспектов получаемых фото- и видеоизображений. Возможность включения в состав мультимодальной	Сложность системы. Необходимость правильного отбора наиболее эффективных методов рас-

Продолжение таблицы 1

Метод мониторинга	Достоинства	Недостатки
	системы всех рассмотренных ранее методов	познавания психоэмоционального состояния при проектировании мультимодальных систем

Обобщая сведения, приведенные в таблице 1, следует отметить, что наибольшей эффективностью потенциально обладают основанные на анализе движений тела методики, в первую очередь мультимодальные системы. Однако реализация таких систем в рамках выполняемой работы вызывает заметные трудности ввиду отсутствия достаточных вычислительных мощностей. Удовлетворительную альтернативу представляют собой системы анализа движений глаз, поскольку:

- методика анализа движений глаз имеет качественное научное обоснование;
- по тематике анализа движений глаз и определения траектории взгляда существует большое количество наборов обучающих данных;
- возможно построение простого и экономичного по вычислительным ресурсам алгоритма (нейронной сети или ансамбля нейронных сетей), требующего для своей работы незначительных объемов входных данных; например, входом подобного алгоритма могут служить изображения глаз малого размера или последовательности двумерных координат опорных точек глаза.

Таким образом, при разработке программного обеспечения распознавания психоэмоционального возбуждения оператора БПЛА представляется разумным использовать метод анализа движений глаз.

Рассмотрим структуру разрабатываемой системы. Очевидно, задача мониторинга психоэмоционального состояния человека по характеру движений его глаз включает в себя следующие подзадачи:

- распознавание лица человека;

- определение координат взгляда в текущий момент времени и формирование траектории взгляда;

- анализ траектории взгляда и принятие решения о психоэмоциональном состоянии человека.

Учитывая вышеизложенное, поделим разрабатываемую систему распознавания психоэмоционального возбуждения на 3 составные части: модуль распознавания лиц; модуль определения траектории взгляда; модуль анализа движений глаз (рисунок 19).

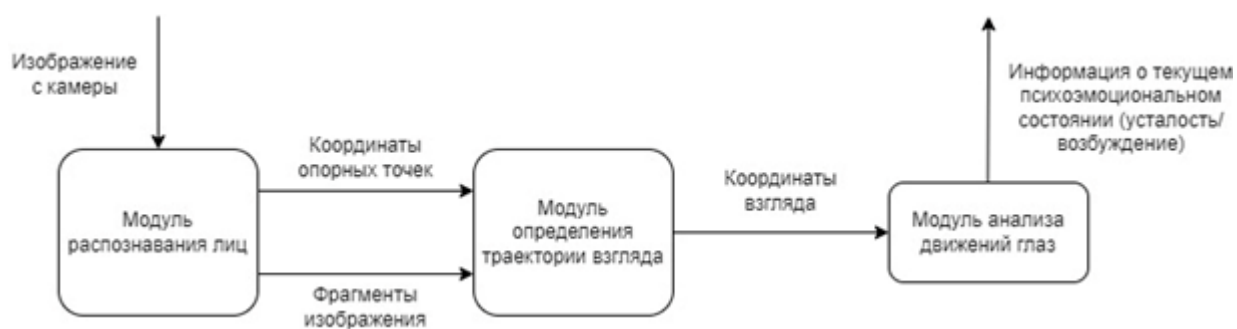


Рисунок 19 – Структура разрабатываемой системы распознавания психоэмоционального возбуждения

Модуль распознавания лиц отвечает за определение опорных точек лица с целью выделения фрагментов изображения, на которых запечатлены глаза. Модуль распознавания лиц также выполняет функцию контроля за концентрацией оператора, поскольку при значительном отклонении положения головы от нормального положения (например, если оператор отвернулся от дисплея системы видеонаблюдения) опорные точки лица не могут быть распознаны; в этом случае модуль подает соответствующий сигнал.

Модуль определения траектории взгляда принимает на вход координаты опорных точек глаз человека, рассчитанные относительно изображения камеры, и выделенные согласно этим координатам фрагменты изображения. Выходными данными модуля служат координаты взгляда, т.е. точка на дисплее, на которую смотрит оператор.

Модуль анализа движений глаз обрабатывает информацию о глазодвигательной активности, накопленную за определенный промежуток времени, и принимает

решение о текущем психоэмоциональном состоянии оператора (нормальное состояние или чрезмерное психоэмоциональное возбуждение).

Поскольку разработка программного обеспечения (в частности, модуля определения траектории взгляда) осуществляется с применением алгоритмов глубокого обучения и нейронных сетей, система распознавания психоэмоционального возбуждения нуждается в наборе обучающих данных. Используемый в ходе выполнения работы набор данных, а также процесс его предварительной обработки будут описаны ниже.

3.2 Поиск и предобработка набора обучающих данных

3.2.1 Набор данных OperatorEY EVP

Упомянутый ранее набор данных, использованный при обучении нейронной сети iTracker, предоставляет все необходимые возможности для разработки качественной системы распознавания психоэмоционального возбуждения. Однако этот набор находится в закрытом доступе, что вынуждает искать альтернативу. Одной из возможных альтернатив является OperatorEY EVP – набор экспериментальных данных, содержащий сведения о глазодвигательной активности и степени усталости 15 человек [10].

По утверждению исследователей, ответственных за сбор и публикацию OperatorEY EVP, запись данных осуществлялась 3 раза в день (в 9:00, 13:00 и 18:00) в течение 8 дней. В ходе записи участникам экспериментов предлагалось выполнить ряд заданий, которые можно разделить на 2 типа:

- задания на контроль вовлеченности и уровня внимания (чтение текста в научном стиле);
- задания, требующие согласованной двигательной реакции (тест на время реакции выбора, корректурная проба «Кольца Ландольта», игра «Тетрис»).

Тест на время реакции выбора (CRT-тест) представлял собой перечень из 70 вопросов общей продолжительностью примерно 6 минут, в ходе которых участникам экспериментов на экране компьютера или ноутбука демонстрировался либо красный, либо зеленый кружок диаметром 2 см. Задача заключалась в том, чтобы нажать

правильную кнопку при появлении кружка красного цвета. В ходе CRT-теста регистрировались следующие параметры: среднее время реакции, стандартное отклонение и количество ошибок. В связи с тем, что у участников экспериментов могла наблюдаться разная степень утомления в начале и в конце сеанса записи данных, было решено проводить CRT-тест дважды (в начале и в конце сеанса).

После прохождения первого CRT-теста участникам предлагалось прочитать текст в научном стиле, чтобы симитировать повседневные рабочие задачи. Задание выполнялось в течение 15 минут с использованием экрана ноутбука или компьютера.

По истечении времени, отведенного для чтения, участники проходили корректурную пробу «Кольца Ландольта» – тест, применяемый для измерения остроты зрения. В качестве стимула для участников выступали ряды колец, имеющих зазор и напоминающих букву «С»; ширина зазора составляла одну пятую его общего диаметра кольца, при этом зазор мог находиться в одном из 8 местоположений. Участник должен был за 5 минут выделить в таблице размером 30 x 30 все кольца, для которых местоположение зазора совпадало с заданным по условию теста. Регистрировались следующие параметры: затраченное время (если участник выполнил задание менее чем за 5 минут), количество целевых стимулов, количество обнаруженных и пропущенных стимулов, количество ошибок, количество просмотренных элементов и строк. На основании этих параметров были рассчитаны показатели продуктивности внимания, точности работы, стабильности концентрации внимания, а также коэффициент умственной продуктивности, работоспособности, концентрации внимания, объема зрительной информации и скорости ее обработки.

Предпоследним заданием каждого сеанса записи данных являлась игра «Тетрис», которая использовалась для определения зрительно-моторной координации. Участникам было предложено достичь наилучшего игрового результата за 15 минут. Исследователями было записано количество игр, набранных очков и т.д.

Участники эксперимента также 1 раз в день (перед началом сеансов записи данных) заполняли анкету качества сна.

Регистрация данных о движении глаз, запись видео и запись частоты сердечных сокращений (ЧСС) осуществлялись одновременно.

Оборудование для записи данных включало в себя:

- ай-трекер в виде очков Pupil Invisible частотой 200 Гц со встроенной камерой частотой 30 Гц, гироскопом и акселерометром;
- оборудование для измерения ЧСС в виде пульсометра Polar Verity Sense со встроенным акселерометром, гироскопом и магнитометром;
- веб-камеру с минимальным разрешением 640 x 480 и кадровой частотой 30 кадров в секунду;
- монитор с диагональю 15,6” и разрешением 1920 x 1080.

На рисунке 20 представлена структура набора данных OperatorEYEVР.

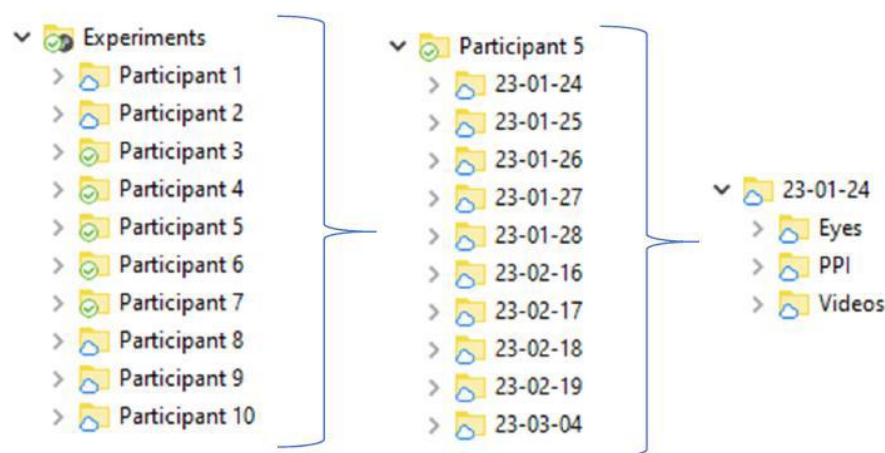


Рисунок 20 – Структура набора данных OperatorEYEVР

Весь набор данных хранится в каталоге «Experiments». Этот каталог содержит подкаталог каждого участника экспериментов. В свою очередь, подкаталоги участников экспериментов содержат папки сеансов записи данных; название папки соответствует дате исполнения сеансов. Папки сеансов записи данных содержат подпапки «Eyes», «PPI» и «Videos», в которых хранятся данные ай-трекера, данные ЧСС в виде интервалов между сердцебиениями (PPI) и видеозаписи участников экспериментов с веб-камеры соответственно. Дополнительно в подкаталоге каждого участника экспериментов сохранены файлы «Metadata» и «Summary». Файл «Metadata» включает в себя таблицу с результатами выполнения CRT-теста (время реакции, стандартное отклонение, количество ошибок), результатами корректурной пробы (19 параметров), результаты игры в «Тетрис» (продолжительность игры, общее количество очков и

т.д.), параметры вариабельности сердечного ритма (58 параметров), результаты заполнения участником специальных анкет, характеризующих его психоэмоциональное состояние, а также общие сведения об участнике (возраст, пол). В файле «Summary» собрана вся приведенная выше информация и дополнительно – специализированные количественные характеристики взгляда.

На рисунке 21 представлена структура папки «Eyes».

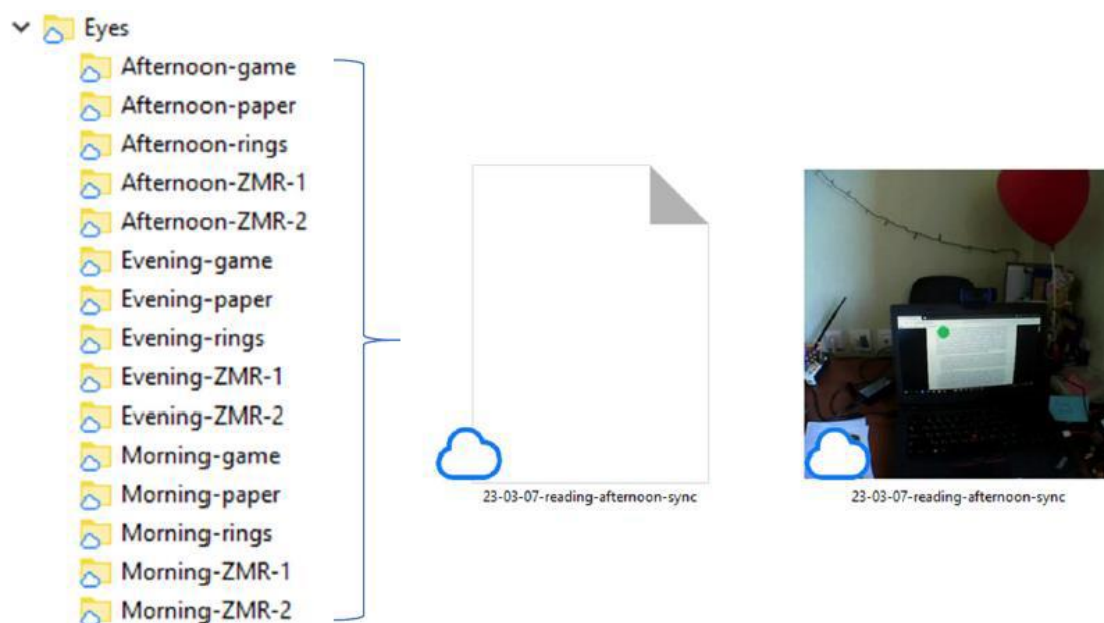


Рисунок 21 – Структура папки «Eyes»

Папка «Eyes» содержит подпапки с результатами ай-трекинга для каждого сеанса записи данных (по 5 заданий для утреннего, дневного и вечернего сеансов). Каждая подпапка содержит файл .csv с нормированными координатами взгляда и файл .mp4 с отснятым материалом. Файл .csv представляет собой таблицу с меткой времени фиксации взгляда, номером кадра, горизонтальной и вертикальной координатами взгляда (рисунок 22).

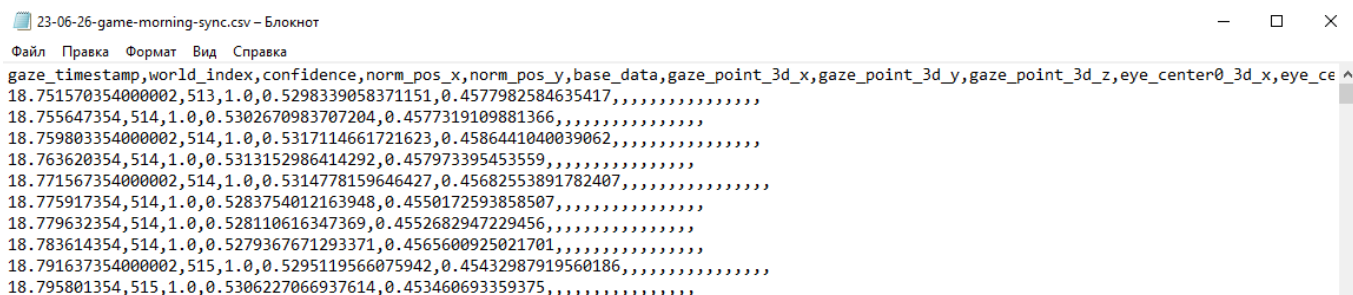


Рисунок 22 – Содержание файла с координатами взгляда

На рисунке 23 приведен кадр из видеозаписи, на которой участник экспериментов выполняет задание по чтению; текст отображается на экране ноутбука, а зеленый кружок представляет собой точку, куда в текущий момент времени направлен взгляд участника.

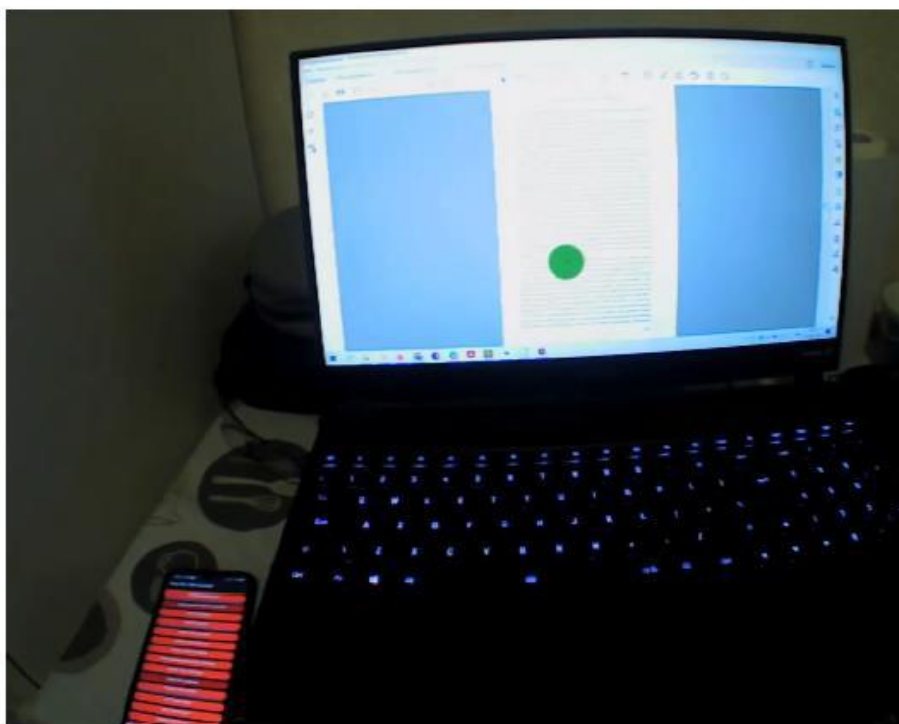


Рисунок 23 – Пример кадра видеозаписи с нанесенными координатами взгляда

На рисунке 24 показана структура папки «Videos», содержащей результаты утреннего, дневного и вечернего сеансов записи данных соответственно. В папках в формате .mp4 хранятся видеозаписи веб-камеры.

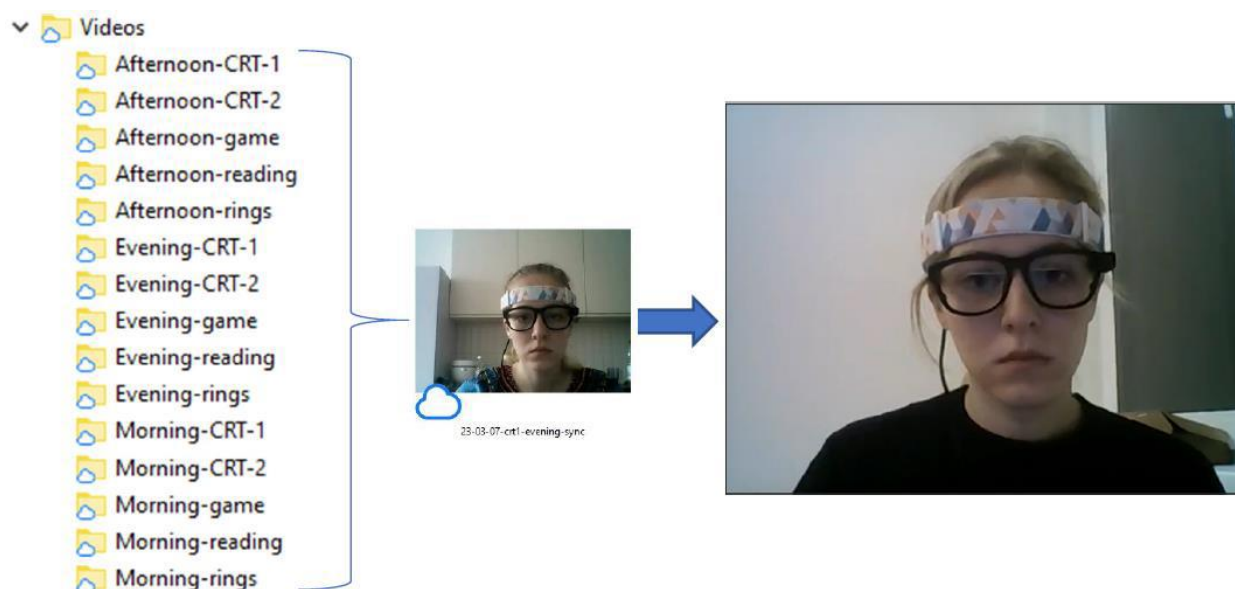


Рисунок 24 – Структура папки «Videos»

Остальные каталоги, содержащиеся в OperatorEYEVР, не представляют существенного интереса с точки зрения разработки системы распознавания психоэмоционального возбуждения, поэтому исключим их из рассмотрения в рамках данной работы.

Безусловными достоинствами OperatorEYEVР являются большое количество собранных данных и разнообразие измеренных показателей. Тем не менее этот набор данных не лишен 2 существенных недостатков:

- не для всех участников экспериментов опубликован полный перечень показателей;
- в ходе сбора данных участники использовали ай-трекер в виде очков, вследствие чего на некоторых видеозаписях с веб-камеры глаза оказываются скрыты за отражением экрана компьютера или ноутбука в стеклах ай-трекера (рисунок 25).



Рисунок 25 – Примеры некачественных видеозаписей с веб-камеры
(глаза участников экспериментов скрыты за отражением экрана)

Последнее обстоятельство представляет собой серьезную проблему и вынуждает проводить вторичный отбор данных вручную.

В ходе разработки системы распознавания психоэмоционального возбуждения было выявлено некоторое несоответствие OperatorEYEVР поставленным задачам. Прежде всего, целевые координаты взгляда, содержащиеся в OperatorEYEVР, заданы относительно натальной камеры, ведущей съемку от первого лица, а не экрана устройства, что не позволяет качественно определить взгляд пользователя на экране. Тем не менее на этом наборе данных было проведено предварительное обучение нейронных сетей, которое продемонстрировало приемлемые результаты (рисунок 26); сам процесс обучения нейронных сетей будет описан в подразделе 3.4 выпускной квалификационной работы.

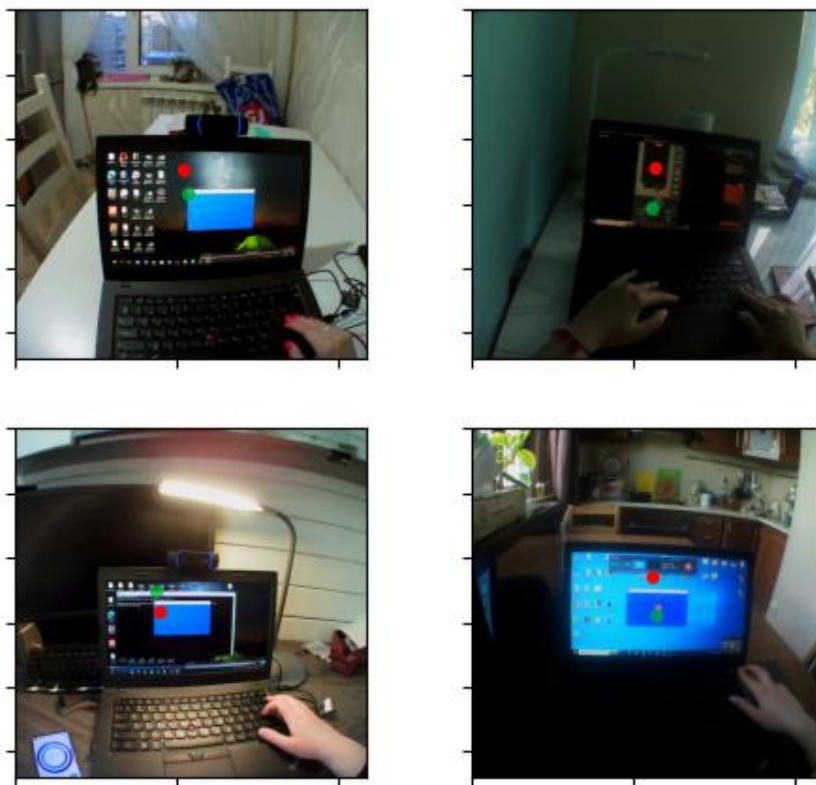


Рисунок 26 – Результаты обучения одной из архитектур нейронной сети:

зеленый кружок – истинное положение точки взгляда;

красный кружок – положение точки взгляда, предсказанное нейронной сетью

Ввиду низкой практической ценности более подробные результаты обучения нейронных сетей на наборе OperatorEYEVР не приводятся.

3.2.2 Набор данных MPiIGaze

Существует еще один открытый набор данных, содержащий информацию о направлении взгляда, под названием MPiIGaze [11]. MPiIGaze включает в себя 213659 изображений лиц анфас, собранных в течение нескольких месяцев от 15 человек во время повседневного использования ноутбука. Примеры изображений из набора данных приведены на рисунке 27.

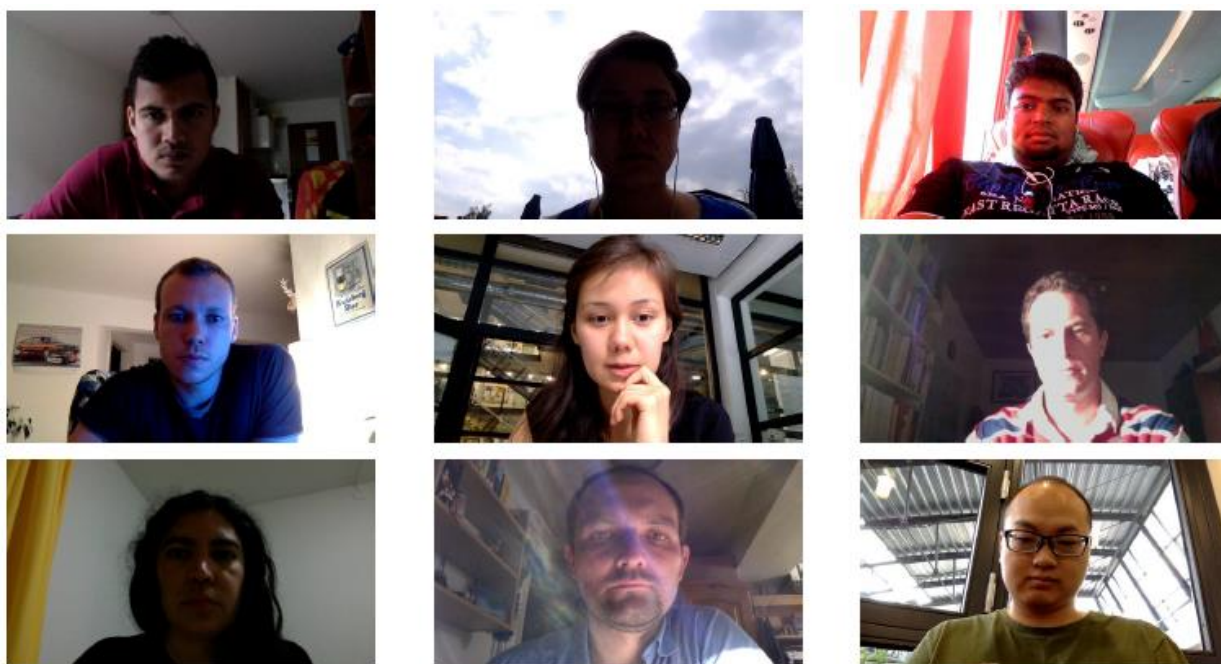


Рисунок 27 – Примеры изображений из набора данных MPIIGaze

Исследователи попытались обеспечить максимальную вариативность характеристик изображений (окружающей обстановки в кадре, уровня освещенности и т.д.).

Дополнительно для 37667 изображений из MPIIGaze были размечены точки, соответствующие зрачкам, а также уголкам рта и глаз. Разметка производилась в 2 этапа: стандартный алгоритм распознавания лиц определял совокупность опорных точек лица, а затем исследователи вносили правки в ручном режиме (рисунок 28; зрачки размечались исключительно вручную).

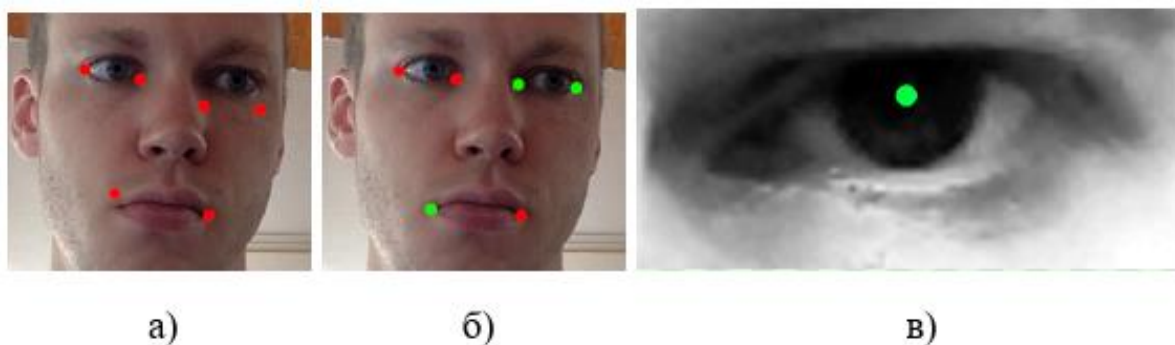


Рисунок 28 – Разметка опорных точек лица: а) до внесения правок;
б) после внесения правок; в) разметка зрачка

Структура набора данных MPIIGaze представлена на рисунке 29.

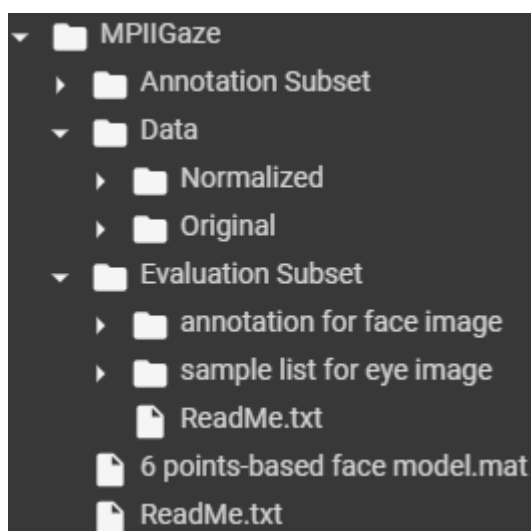


Рисунок 29 – Структура набора данных MPIIGaze

Рассмотрим основные структурные элементы набора MPIIGaze. Папка «Data» включает себя подпапки «Original» и «Normalized», которые содержат соответственно оригинальные и нормализованные (т.е. преобразованные по координатам и цвету) изображения. Подпапки «Original» и «Normalized» разделены на каталоги участников сбора данных, каталоги участников – на подкаталоги дней записи данных (в каталогах участников также содержится информация о калибровке камеры). В папках «Annotation Subset» и «Evaluation Subset» хранятся аннотации (в виде списка опорных точек лица, целевых характеристик взгляда и т.д.) для каждого изображения. Также отдельный файл аннотации .txt содержится в каждом подкаталоге дней записи данных.

Как и в случае OperatorEYEVN, принципиальный недостаток набора MPIIGaze заключается в формате представления целевых данных (координат взгляда). Поскольку данные для MPIIGaze были собраны с различных моделей ноутбуков (имеющих соответственно различные размеры и разрешение экранов), авторами исследования было принято решение преобразовать точки взгляда на экране в трехмерные позиции в системе координат записывающей камеры посредством серии калибровок. Иными словами, нейронная сеть, обученная на наборе MPIIGaze, позволяет предсказывать угол направления взгляда, но не координаты на экране устройства (рисунок 30).



Рисунок 30 – Пример работы системы, обученной с использованием набора данных MPIIGaze

MPIIGaze также следует исключить из рассмотрения в ходе поиска набора обучающих данных

3.2.3 Сбор и анализ характеристик собственного набора данных

Таким образом, имеющиеся в открытом доступе наборы данных не позволяют эффективно оценить психоэмоциональное состояние человека по его глазодвигательной активности на экране устройства, в связи с чем было принято решение сформировать собственный набор данных.

С целью первоначального сбора данных была разработана программа, выполняющая фотографирование лица человека и запись координат взгляда на дисплее устройства (в данном случае – ноутбука HP Laptop 17-by2xxx, разрешение экрана – 1600 x 900 пикселей) в виде csv-файла. На рисунке 31 представлен интерфейс разработанной программы.

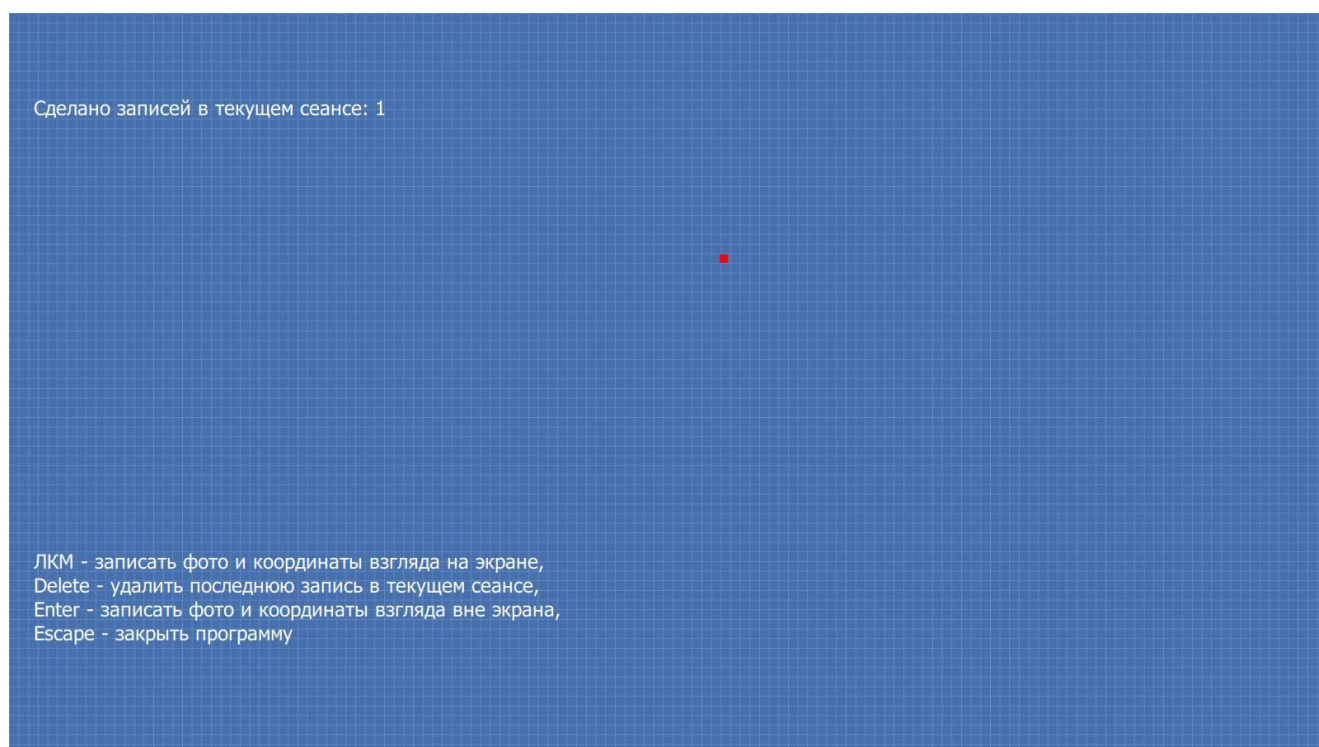


Рисунок 31 – Интерфейс программы для сбора данных

В верхнем текстовом поле («Сделано записей в текущем сеансе...») приведено количество фотографий, сохраненных за один сеанс записи данных; нижнее текстовое поле содержит справочную информацию о функционале приложения.

При нажатии левой кнопкой мыши в произвольной точке экрана происходит фотографирование лица человека и запись координат точки, в которой произошло нажатие (при этом происходит приведение координат к величине от 0 до 1). Последняя точка нажатия отображается на экране. Также присутствует возможность сохранения фотографии человека, смотрящего вне экрана, путем нажатия клавиши Enter (в таком случае в csv-файл вместо приведенных координат по осям X и Y записывается -1).

На рисунках 32 и 33 приведен пример фотографии и содержание csv-файла соответственно.



Рисунок 32 – Пример фотографии из сформированного набора данных

	A	B	C	D
1	jpg_name,x_norm,y_norm			
2	1700907701290.jpg,0.004375,0.00666667			
3	1700907702762.jpg,0.02625,0.00444444			
4	1700907704265.jpg,0.049375,0.00666667			
5	1700907705576.jpg,0.071875,0.00666667			
6	1700907707400.jpg,0.094375,0.00777778			
7	1700907708952.jpg,0.1175,0.00555556			
8	1700907710569.jpg,0.145625,0.00111111			
9	1700907711785.jpg,0.1775,0.00444444			
10	1700907713088.jpg,0.2025,0.00444444			
11	1700907714410.jpg,0.2225,0.00222222			
12	1700907715626.jpg,0.246875,0.00111111			
13	1700907717129.jpg,0.273125,0.00666667			
14	1700907718649.jpg,0.293125,0			
15	1700907720050.jpg,0.318125,0.00444444			
16	1700907721369.jpg,0.345625,0.00222222			
17	1700907722698.jpg,0.36875,0.00444444			
18	1700907724009.jpg,0.393125,0			

Рисунок 33 – Содержание файла с координатами взгляда из сформированного набора данных

Фотографирование лица осуществлялось при помощи веб-камеры Logitech C310 HD WebCam разрешением 1280 x 720 пикселей и кадровой частотой 30 кадров в секунду (рисунок 34).



Рисунок 34 – Веб-камера Logitech C310 HD WebCam

Фотографирование производилось на расстоянии около 70 см от экрана ноутбука. Всего в наборе данных содержится 5220 экземпляров.

Разработка программы осуществлялась на языках C++ (фреймворки Qt и OpenCV) и QML [12]. Исходный код программы приведен в приложении А.

Весь массив координат взгляда (фактически – точек на экране ноутбука) может быть отображен в виде сетки, состоящей из 12 ячеек размерами 400 x 300 пикселей. Также представляется возможным задать область вне экрана в качестве дополнительной ячейки (рисунок 35).

Область вне экрана – 13

1	2	3	4
5	6	7	8
9	10	11	12
Экран			

Рисунок 35 – Представление массива координат в виде сетки

Упомянутое выше разбиение на ячейки будет использовано при разработке тестового пользовательского интерфейса (подраздел 3.5 выпускной квалификационной работы).

На рисунке 36 представлено распределение экземпляров из набора обучающих данных по ячейкам экрана (информация для области вне экрана не приводится).

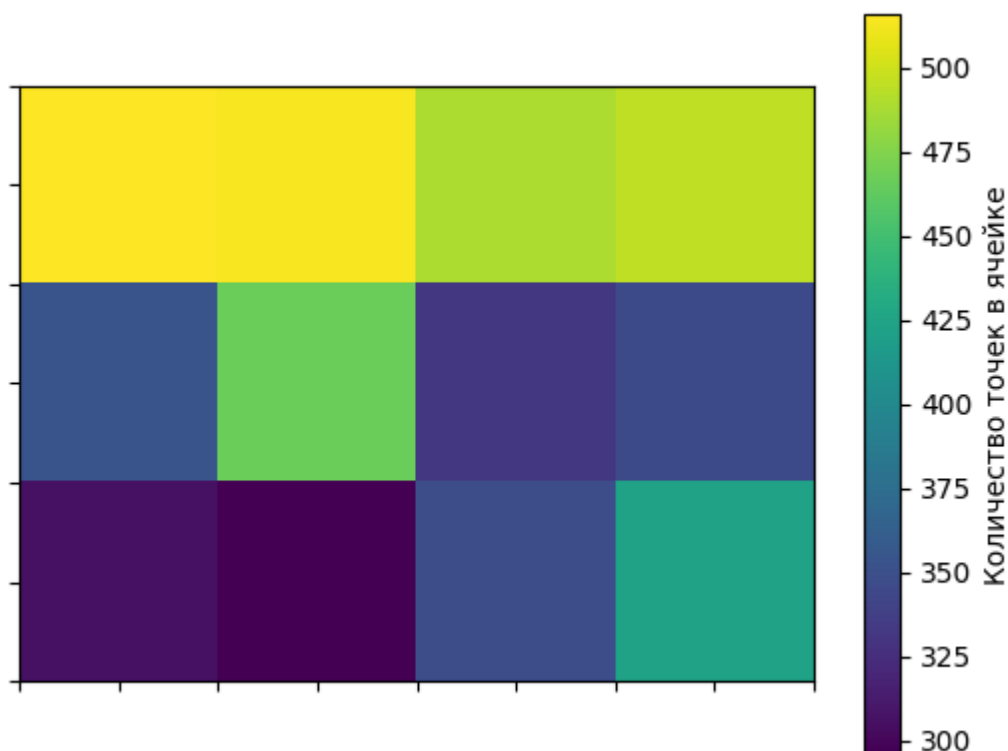


Рисунок 36 – Распределение обучающих данных по ячейкам экрана

Заметна некоторая несбалансированность координат по ячейкам; необходимо учитывать данный недостаток сформированного набора данных при обучении нейронных сетей.

3.3 Разработка модуля распознавания лиц

В настоящее время существует довольно много стандартных алгоритмов распознавания лиц, обеспечивающих высокое быстродействие и удовлетворительную точность распознавания. Однако все эти алгоритмы так или иначе допускают

определенную погрешность, что заставляет осуществлять дополнительные манипуляции над изображениями для улучшения качества обработки данных.

Рассмотрим стандартные алгоритмы, которые могут быть использованы при разработке модуля распознавания лиц в рамках выпускной квалификационной работы.

Один из наиболее популярных алгоритмов представлен в кроссплатформенной библиотеке программирования Dlib. Данный алгоритм представляет собой ансамбль предобученных моделей: детектора области изображения, на которой запечатлено лицо, и предиктора, определяющего заданное количество опорных точек лица. В рамках выпускной квалификационной работы рассматриваются 2 типа детекторов (детектор фронтального ракурса лица с использованием гистограммы направленных градиентов и линейной машины опорных векторов; детектор на основе сверточной нейронной сети) и предиктор 68 опорных точек (рисунок 37).

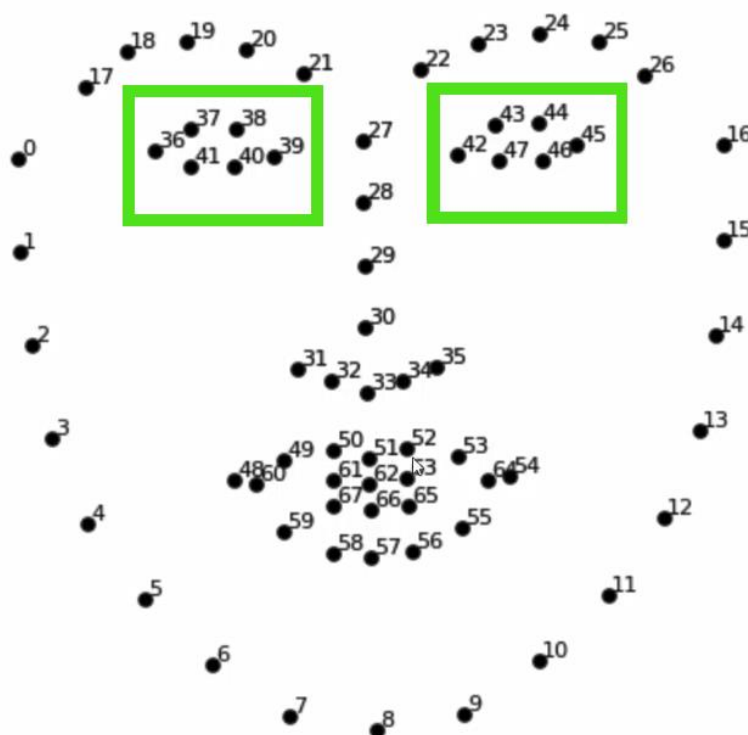


Рисунок 37 – Опорные точки лица, определяемые предиктором Dlib

Основными недостатками описанного ансамбля детектора и предиктора являются:

- невозможность детектирования лица в тех случаях, когда положение лица отклоняется от фронтального ракурса (однако данное свойство может служить преимуществом при определении ситуаций, когда оператор отвлекается от дисплея системы видеонаблюдения или управления БПЛА);

- периодически неверное определение опорных точек предиктором (однако указанный недостаток свойственен всем стандартным алгоритмам распознавания лиц).

Дополнительные возможности по распознаванию лиц может предоставить модуль определения опорных точек лица Mediapipe Face Mesh, разработанный компанией Google [13].

Главное достоинство модуля Mediapipe Face Mesh по сравнению с моделями, содержащимися в библиотеке Dlib, заключается в увеличенном количестве определяемых опорных точек (всего задано 468 точек). При этом Mediapipe Face Mesh выполняет одновременно и функцию детектора, и функцию предиктора.

Модуль Mediapipe Face Mesh доступен как составная часть библиотеки программирования Mediapipe, написанной на языке Python. Разработчиками модуля была предусмотрена возможность задания различных параметров модуля, в частности, пороговых значений уверенности в результатах детектирования и опознания опорных точек.

Опорные точки лица, определяемые Mediapipe Face Mesh, включают в себя координаты зрачков и радужной оболочки глаз: данное обстоятельство также представляет собой заметное преимущество при обработке изображений (сетка опорных точек лица показана на рисунке 38).

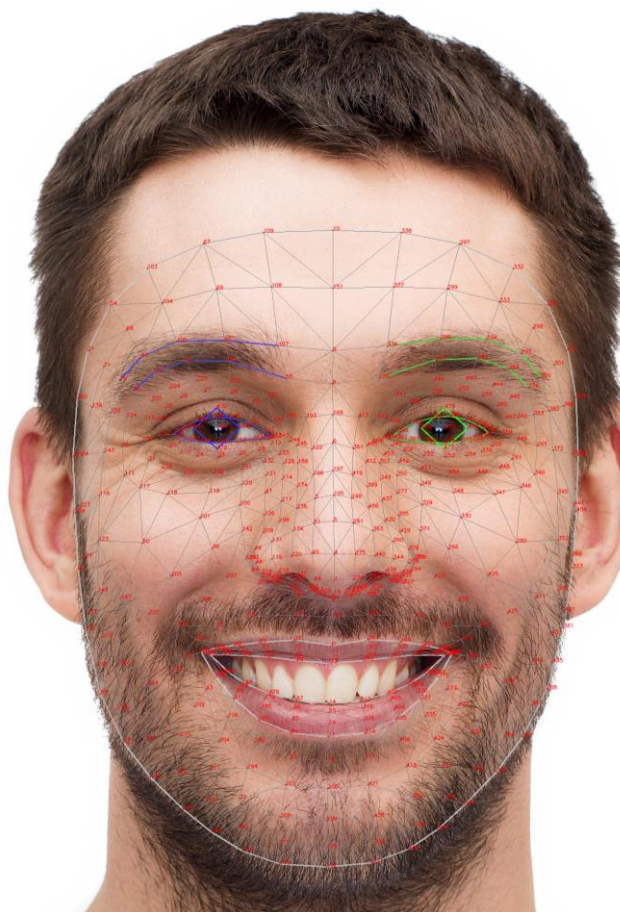


Рисунок 38 – Опорные точки лица, определяемые Mediapipe Face Mesh

Выполним сравнение описанных выше алгоритмов распознавания лиц с точки зрения качества их работы. Список алгоритмов выглядит следующим образом:

- детектор (комбинация гистограммы направленных градиентов – HOG – и линейной машины опорных векторов – SVM) и предиктор библиотеки программирования Dlib;
- детектор (сверточная нейронная сеть – CNN) и предиктор библиотеки программирования Dlib;
- модуль определения опорных точек лица Mediapipe Face Mesh.

Первый алгоритм будет протестирован в 2 вариантах: на черно-белых и RGB-изображениях. Последний алгоритм (Mediapipe Face Mesh) также будет протестирован в 2 вариантах: с пороговыми значениями уверенности 0,65 и 0,9 (пороговые значения для этапов детектирования лиц и определения опорных точек настраиваются отдельно, но в целях упрощения примем эти величины равными между собой).

Случайным образом выберем из набора обучающих данных 100 фотографий и последовательно протестируем на полученном массиве изображений каждый из перечисленных алгоритмов.

Определим основные метрики качества:

- количество изображений, для которых лицо не было распознано;
- количество изображений, для которых опорные точки (точки, соответствующие положению глаз человека) определены неверно;
- время, потраченное на обработку всего массива изображений (рассчитывается как среднее арифметическое для 5 запусков каждого алгоритма).

Тестирование всех алгоритмов осуществляется в локальной среде выполнения на центральном процессоре Intel Core i5-10210U.

Результаты тестирования алгоритмов распознавания лиц приведены в таблице 2.

Таблица 2 – Результаты тестирования стандартных алгоритмов распознавания лиц

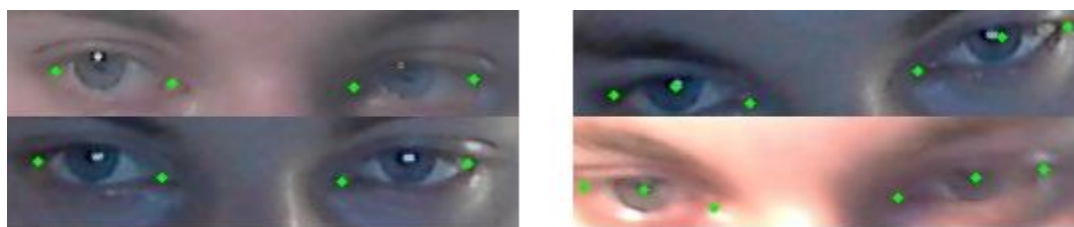
Алгоритм	Тип изображений	Количество изображений с неопознанными лицами	Количество изображений с неверно определенными опорными точками	Время, с
Детектор (HOG, SVM) и предиктор Dlib	ЧБ	11	0	18,67
Детектор (HOG, SVM) и предиктор Dlib	RGB	7	0	27,37
Детектор (CNN) и предиктор Dlib	RGB	0	0	2138,25

Продолжение таблицы 2

Алгоритм	Тип изображений	Количество изображений с неопознанными лицами	Количество изображений с неверно определенными опорными точками	Время, с
Mediapipe Face Mesh, пороговое значение 0,65	RGB	0	0	19,27
Mediapipe Face Mesh, пороговое значение 0,9	RGB	2	0	18,08

Время обработки включает в себя также время открытия и записи изображений.

Наибольшее время обработки изображений (около 2138 с) было получено для ансамбля детектора (CNN) и предиктора: по-видимому, данный детектор предназначен для запуска исключительно на графическом процессоре (среднее время обработки изображений для того же детектора на ускорителе NVIDIA Tesla T4 составило около 16 с). Наименее точным оказался детектор – комбинация HOG и SVM. При этом необходимо отметить тот факт, что и предиктор библиотеки Dlib, и модуль Mediapipe Face Mesh успешно справились с определением опорных точек: хотя расположение точек не является идеальным, представляется вполне возможным ограничить область глаз по полученным координатам (рисунок 39).



а

б

Рисунок 39 – Опорные точки, определенные: а) предиктором Dlib;
б) модулем Mediapipe Face Mesh

Таким образом, в рамках выпускной квалификационной работе представляется разумным использовать Mediapipe Face Mesh, поскольку применение данного алгоритма позволяет обеспечить ряд преимуществ:

- отсутствие какой-либо аппаратной зависимости;
- большое количество опорных точек;
- приемлемое время обработки изображений.

Однако необходимо учитывать, что Mediapipe Face Mesh предназначен для работы с изображениями высокого разрешения.

Чтобы минимизировать погрешность распознавания лиц, будем выполнять дополнительную проверку опорных точек, которую можно разделить на 2 этапа:

- этап проверки цвета зрачков и уголков глаз;
- этап определения морганий глаз.

Проверка цвета зрачков и уголков глаз задается системой неравенств

$$C_{pupil} \leq C_{left\ corner}, \quad (3)$$

$$C_{pupil} \leq C_{right\ corner}, \quad (4)$$

где C_{pupil} , $C_{left\ corner}$, $C_{right\ corner}$ – цвет опорной точки зрачка, левого уголка и правого уголка глаза в черно-белой расцветке изображения соответственно.

Проверка цвета основана на предположении, что после перевода изображения в черно-белую расцветку зрачок должен оказаться темнее либо сопоставим по цвету

с уголками глаз. Если данное условие не выполняется, опорные точки глаза определены неверно.

В свою очередь, моргания можно определить по взаимному расположению опорных точек, используя формулу соотношения сторон глаза [14]

$$EAR = \frac{\|P_2 - P_6\| + \|P_3 - P_5\|}{2\|P_1 - P_4\|}, \quad (5)$$

где $P_{1...6}$ – координаты опорных точек глаза.

На рисунке 40 показана используемая конфигурация опорных точек.

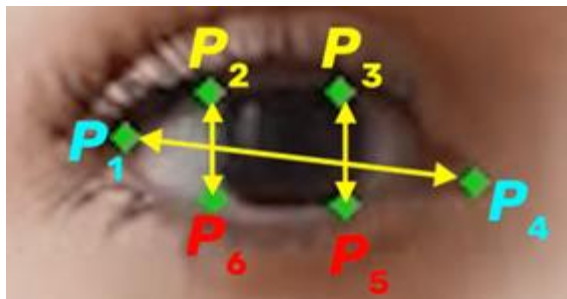


Рисунок 40 – Опорные точки глаза, используемые при определении морганий

Если величина EAR оказывается ниже пороговой, это означает, что человек моргнул (либо опорные точки определены неверно и глаз имеет неправильную форму).

В дальнейшем при определении морганий глаза будет производиться расчет усредненного значения EAR:

$$EAR_{avg} = \frac{EAR_{left} + EAR_{right}}{2}, \quad (6)$$

где EAR_{left} и EAR_{right} – значения EAR соответственно для левого и правого глаза.

Выполняемая дополнительная проверка позволит значительно повысить качество разработанного модуля распознавания лиц.

3.4 Разработка модуля определения траектории взгляда

Выполним предобработку набора данных для обучения нейронных сетей. Блок-схема алгоритма предобработки сформированного набора данных показана на рисунке 41.

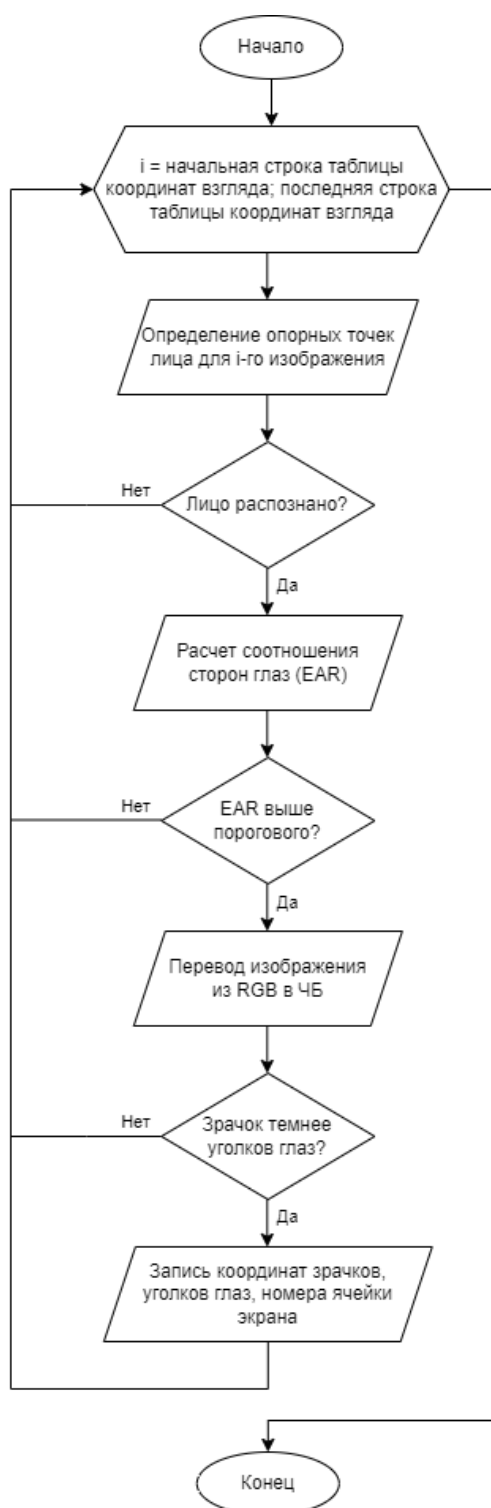


Рисунок 41 – Блок-схема алгоритма предобработки набора данных

Пороговое значение EAR принято равным 0,2.

В обучении не участвуют изображения, для которых координаты взгляда указаны вне экрана. Отобранные данные были разделены на тренировочную, валидационную и тестовую выборки количеством 3857, 203 и 214 экземпляров соответственно.

Обучение производится на графическом процессоре NVIDIA Tesla T4. Программа обучения реализована на языке Python [15-19]; в частности, использованы библиотеки NumPy и PyTorch. Исходный код программы приведен в приложении Б.

Осуществим обучение нескольких архитектур нейронных сетей:

- базовая архитектура, разработанная исследователями Google Research и описанная в подразделе 2.2.2;
- «гибридная» архитектура (iTracker с измененным форматом входных данных);
- модификация базовой архитектуры, предназначенная для обработки одного глаза (выбран правый глаз; выполнено 2 сеанса обучения);
- 2 полносвязные архитектуры (3 слоя) с максимальной размерностью скрытого пространства 256 и 512.

Ниже перечислены гиперпараметры обучения.

Функция потерь – средний квадрат ошибки определения координат

$$MSE = \frac{\sum_{i=1}^N [(x_{i \text{ target}} - x_{i \text{ output}})^2 + (y_{i \text{ target}} - y_{i \text{ output}})^2]}{N}, \quad (7)$$

где N – размер (длина) набора обучающих данных;

$x_{i \text{ target}}, y_{i \text{ target}}$ – истинные координаты взгляда для i -го экземпляра из набора обучающих данных;

$x_{i \text{ output}}, y_{i \text{ output}}$ – координаты взгляда для i -го экземпляра из набора обучающих данных, предсказанные нейронной сетью.

Размер пакета входных данных – 32.

Начальная скорость обучения: для базовой архитектуры: 0,016; для архитектуры, обрабатывающей один глаз, – 0,016 и 0,001; для гибридной архитектуры – 0,001; для полносвязных архитектур – 0,001.

Оптимизатор – алгоритм Adam с параметрами $\beta_1 = 0,9$, $\beta_2 = 0,999$.

Количество эпох обучения: для полносвязных архитектур – 20; для остальных архитектур – 30.

Метод уменьшения скорости обучения – одномоментное уменьшение: для базовой и гибридной архитектур – на 46% каждые 10 эпох; для архитектуры, обрабатывающей один глаз, – на 46% и в 10 раз каждые 10 эпох; для полносвязных архитектур – в 10 раз после первой, пятой и десятой эпохи.

Рассмотрим базовую архитектуру. На рисунке 42 показан график обучения базовой архитектуры.

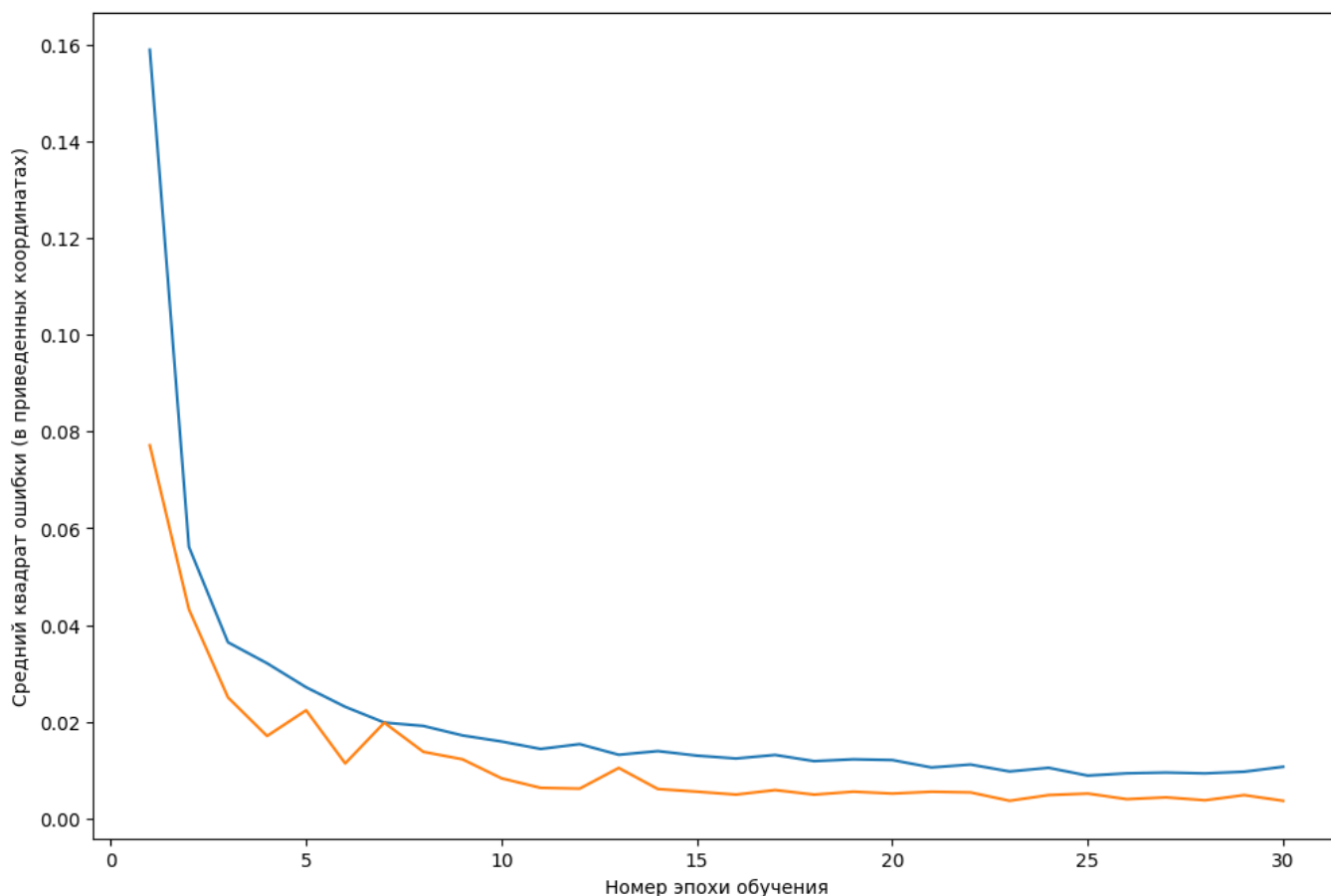


Рисунок 42 – График обучения базовой архитектуры:

синяя линия – результаты обучения для тренировочной выборки;

оранжевая линия – результаты обучения для валидационной выборки

Обучение базовой архитектуры заняло 90,5 минут, при этом общее количество обучаемых параметров оказалось равным 297390.

Величина среднего квадрата ошибки в неприведенных координатах составила 4100 пикселей в квадрате, что соответствует среднеквадратичной ошибке 64 пикселя (или 1,7 см).

На рисунке 43 отображен средний квадрат ошибки для каждой ячейки экрана.

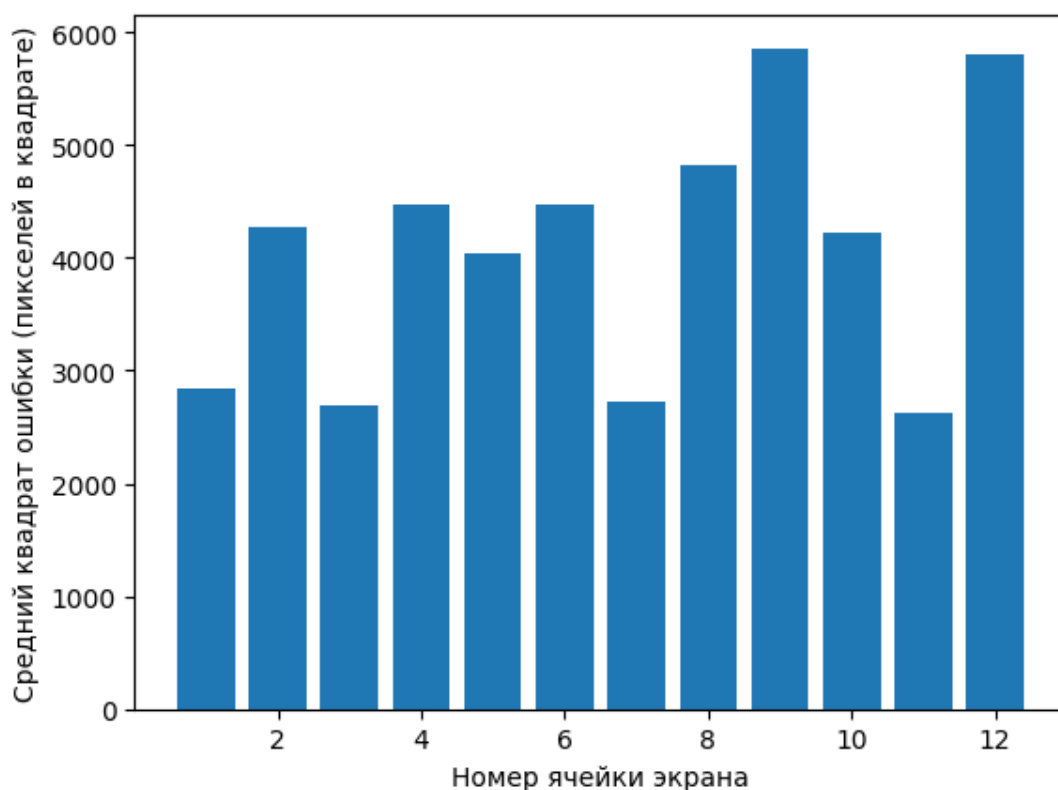


Рисунок 43 – Средний квадрат ошибки определения координат для каждой ячейки экрана (базовая архитектура)

Входные данные «гибридной» архитектуры аналогичны по формату входным данным базовой архитектуры с тем отличием, что в «гибридной» нейронной сети используются изображения размером 25 x 25 пикселей. «Гибридная» архитектура показана на рисунке 44.

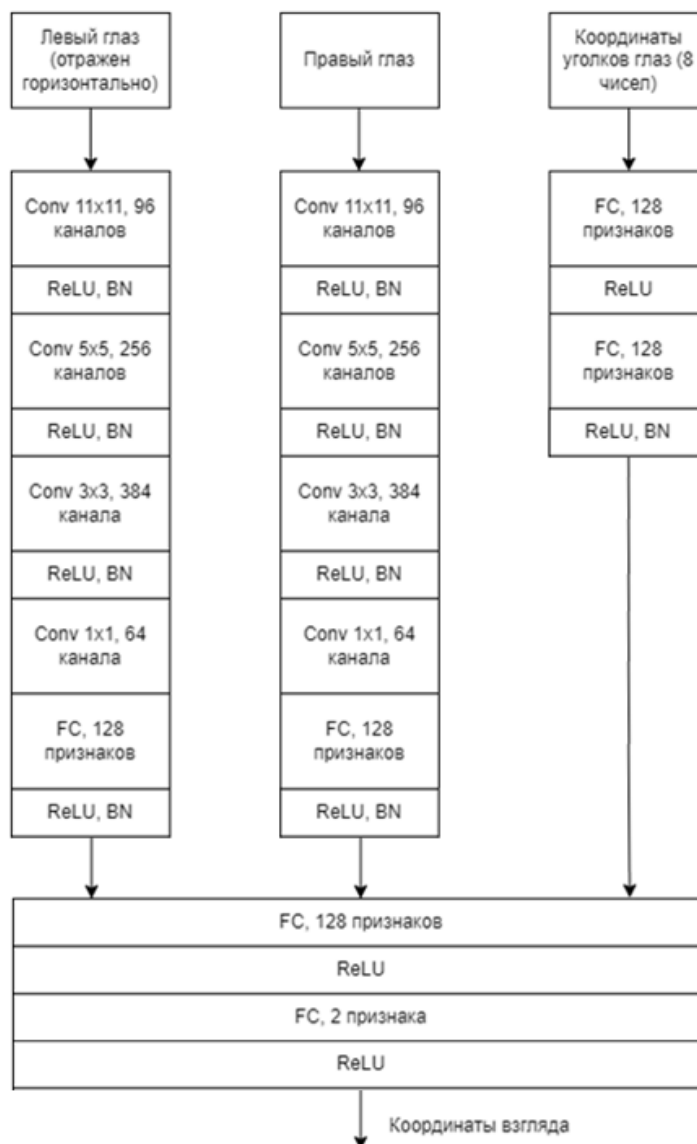


Рисунок 44 – «Гибридная» архитектура

Обучение «гибридной» архитектуры заняло 83,9 минут, при этом количество обучаемых параметров оказалось равным 4517762 (приблизительно в 15 раз больше количества параметров в базовой архитектуре). На рисунках 45 и 46 показаны график обучения и результаты тестирования (средний квадрат ошибки для ячеек экрана) «гибридной» архитектуры.

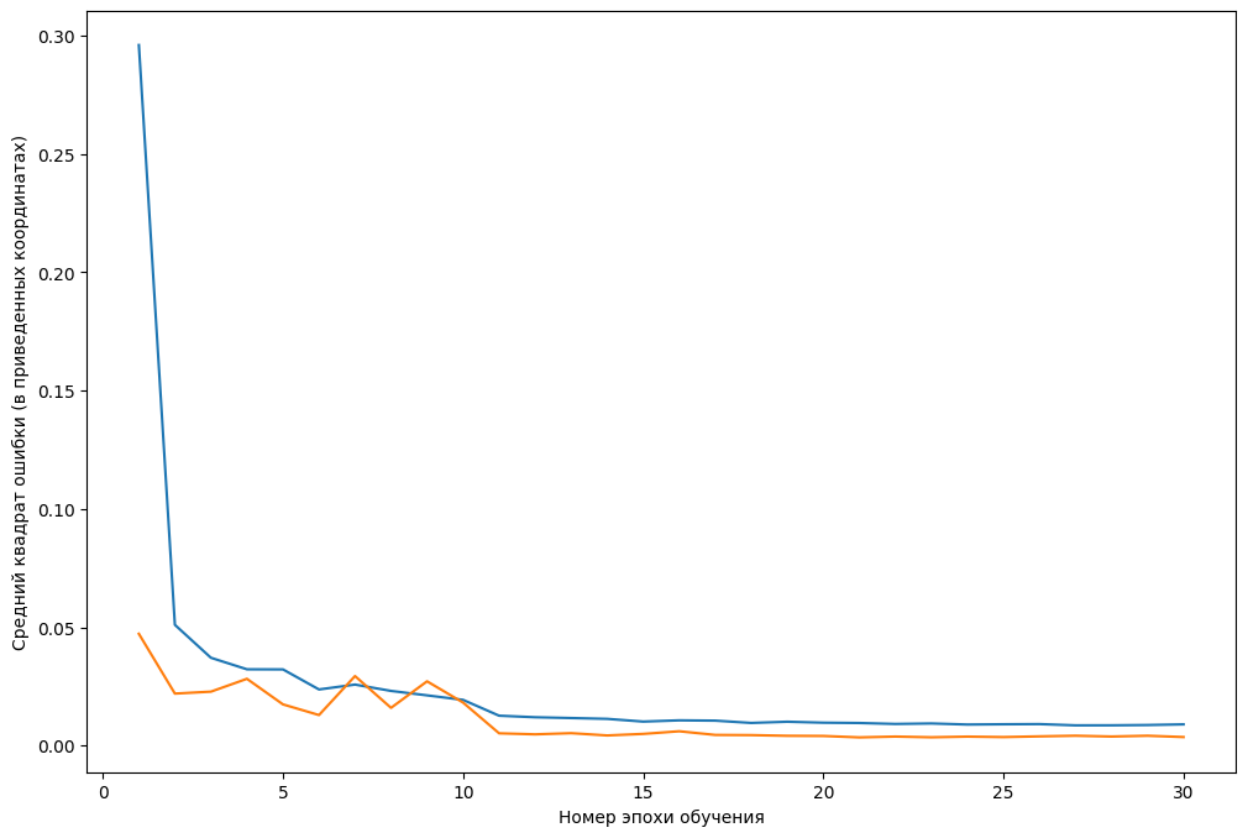


Рисунок 45 – График обучения «гибридной» архитектуры:
 синяя линия – результаты обучения для тренировочной выборки;
 оранжевая линия – результаты обучения для валидационной выборки

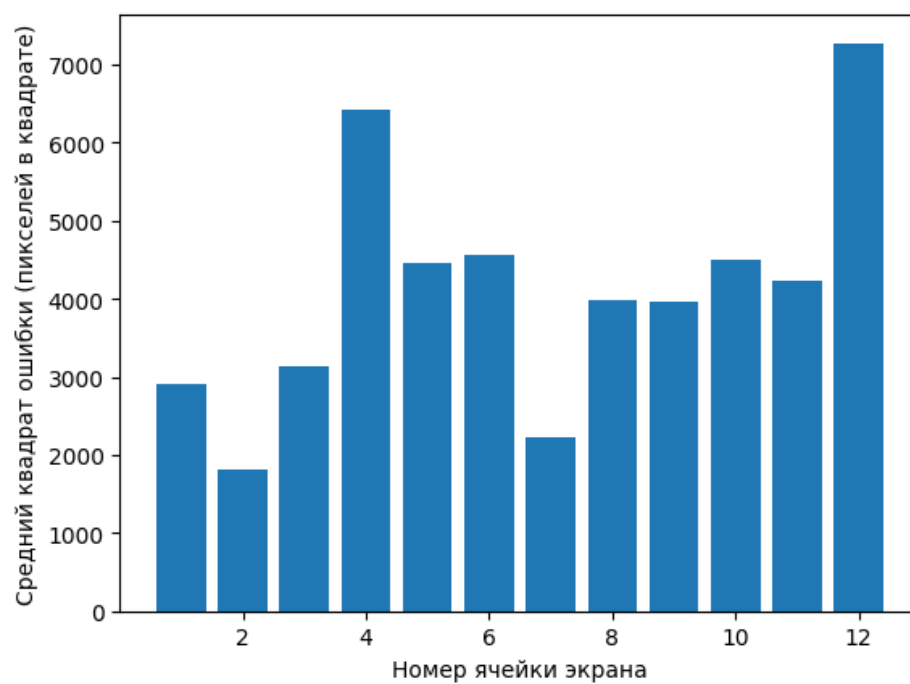


Рисунок 46 – Средний квадрат ошибки определения координат для каждой ячейки экрана («гибридная» архитектура)

В этот раз величина среднего квадрата ошибки в неприведенных координатах составила 4179 пикселей в квадрате, что соответствует среднеквадратичной ошибке 65 пикселей (или 1,7 см).

На рисунке 47 представлена архитектура для обработки одного глаза.

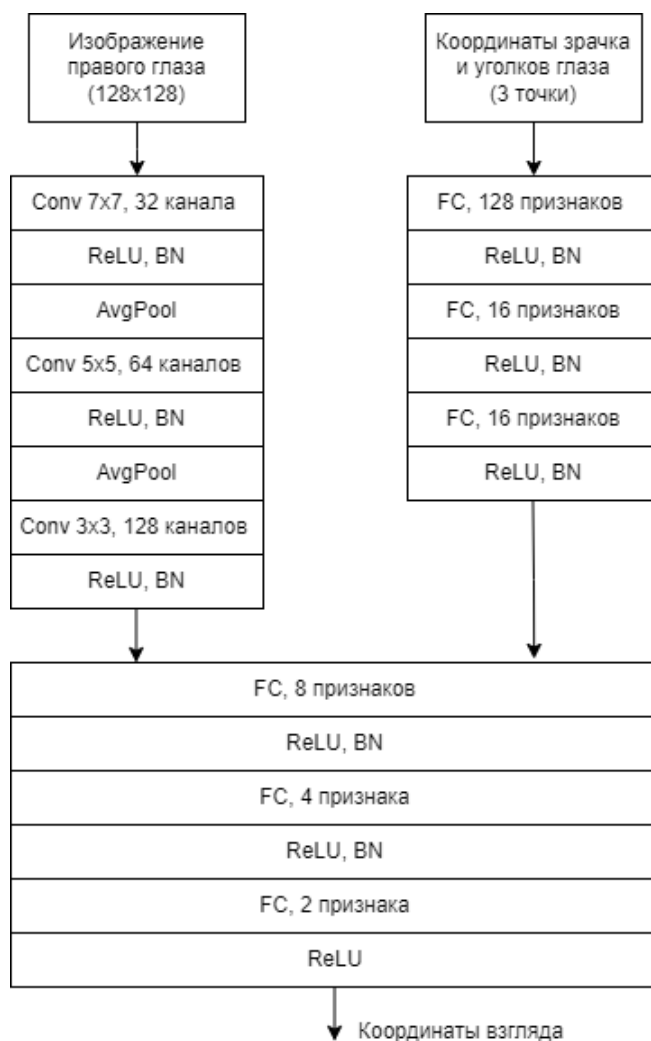


Рисунок 47 – Архитектура нейронной сети для обработки одного глаза

На рисунках 48 и 49 приведены график обучения и результаты тестирования (средний квадрат ошибки для ячеек экрана) архитектуры, обрабатывающей один глаз (первая вариация гиперпараметров: начальная скорость обучения – 0,016, скорость обучения уменьшается на 46 % каждые 10 эпох).

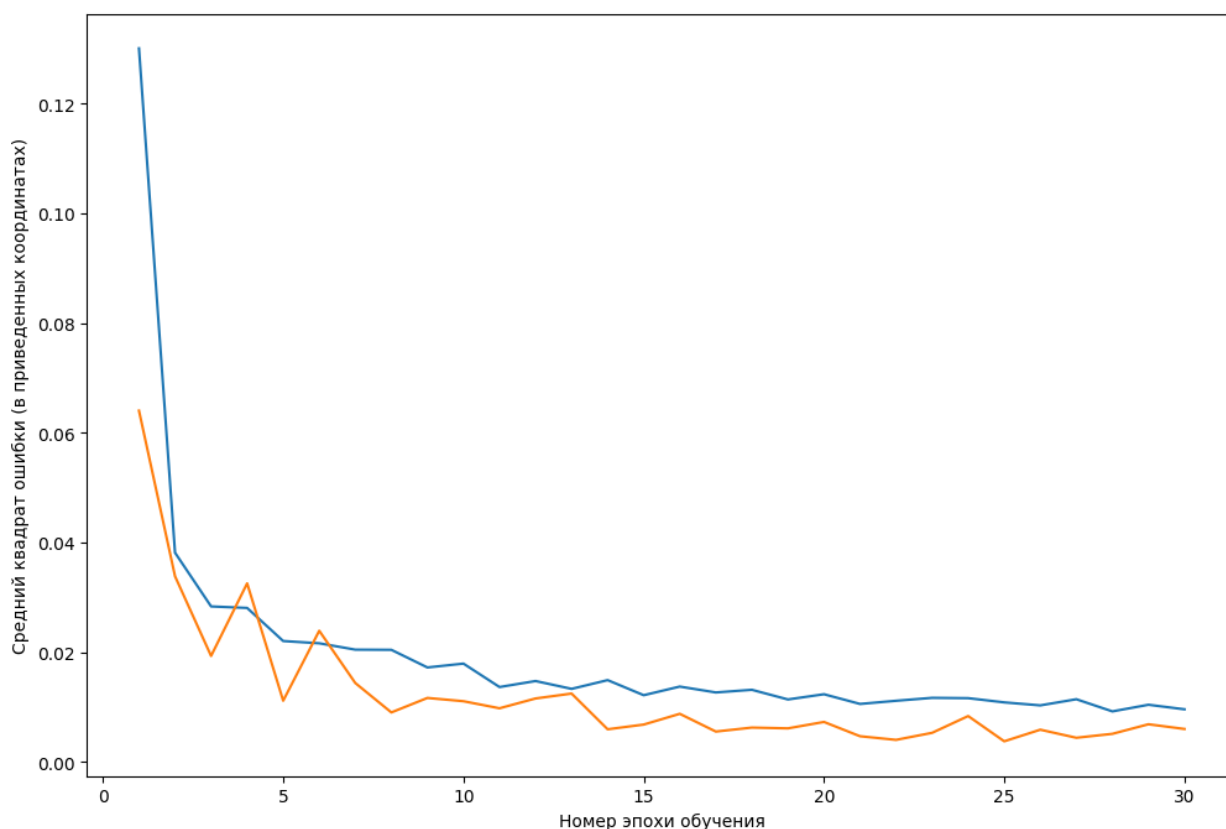


Рисунок 48 – График обучения (первая вариация гиперпараметров):
 синяя линия – результаты обучения для тренировочной выборки;
 оранжевая линия – результаты обучения для валидационной выборки

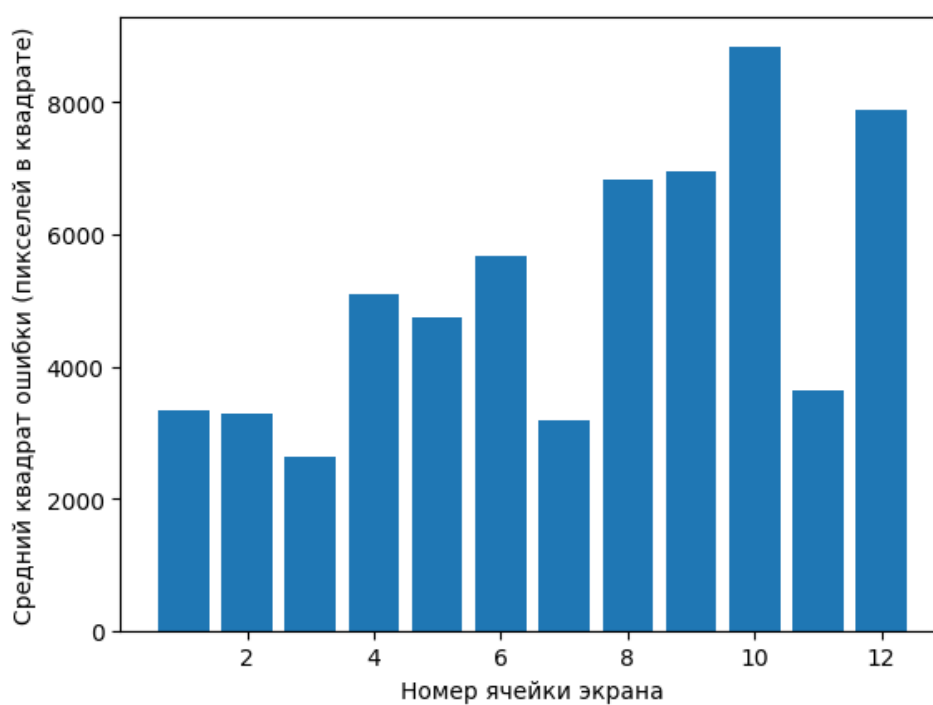


Рисунок 49 – Средний квадрат ошибки определения координат для
 каждой ячейки экрана (первая вариация гиперпараметров)

Обучение данной архитектуры заняло 93,7 минут, общее количество обучаемых параметров оказалось равным 150446. Величина среднего квадрата ошибки составила 5090 пикселей в квадрате, что соответствует среднеквадратичной ошибке 71 пиксель (или 1,8 см).

На рисунках 50 и 51 приведены график обучения и результаты тестирования (средний квадрат ошибки для ячеек экрана) архитектуры, обрабатывающей один глаз (вторая вариация гиперпараметров: начальная скорость обучения – 0,001, скорость обучения уменьшается в 10 раз каждые 10 эпох).

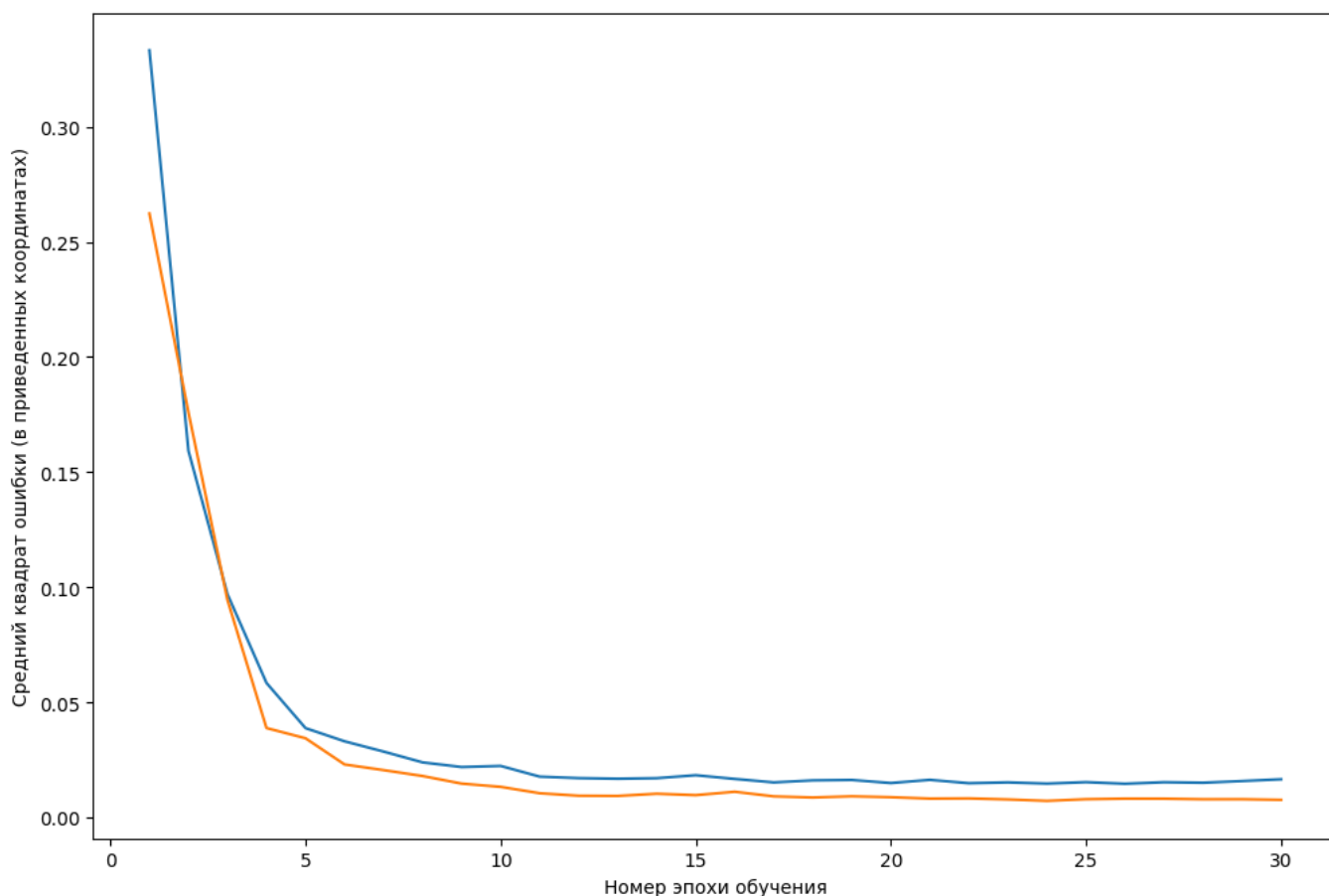


Рисунок 50 – График обучения (вторая вариация гиперпараметров):
синяя линия – результаты обучения для тренировочной выборки;
оранжевая линия – результаты обучения для валидационной выборки

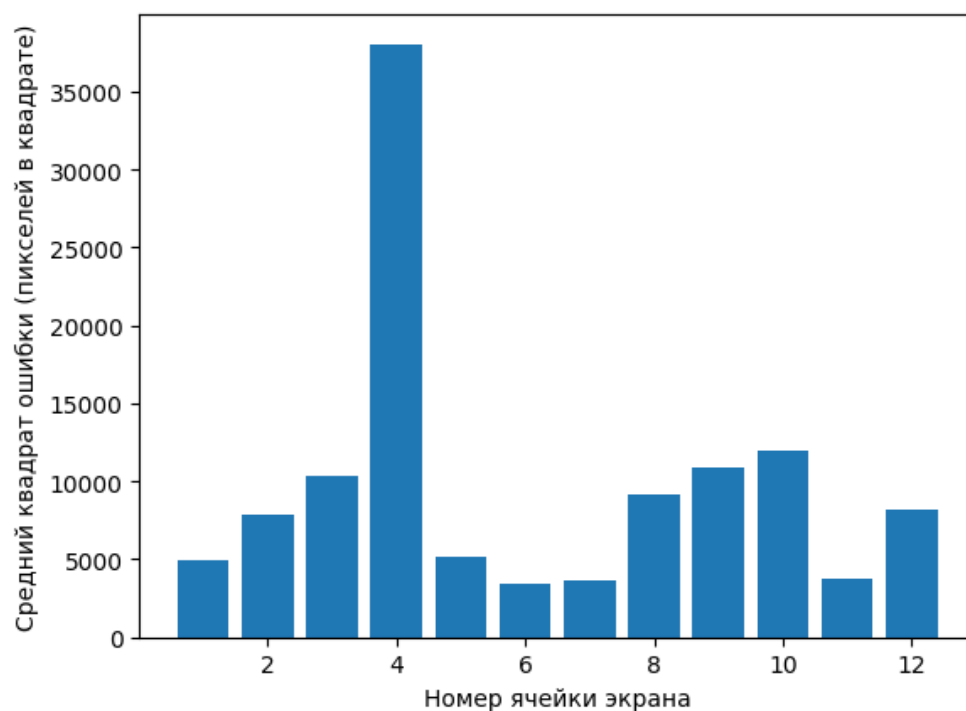


Рисунок 51 – Средний квадрат ошибки определения координат для каждой ячейки экрана (вторая вариация гиперпараметров)

Обучение данной архитектуры заняло 92,3 минут. Величина среднего квадрата ошибки составила 9868 пикселей в квадрате, что соответствует среднеквадратичной ошибке 99 пикселей (или 2,6 см).

На рисунке 52 представлена полносвязная архитектура, состоящая из 3 слоев.

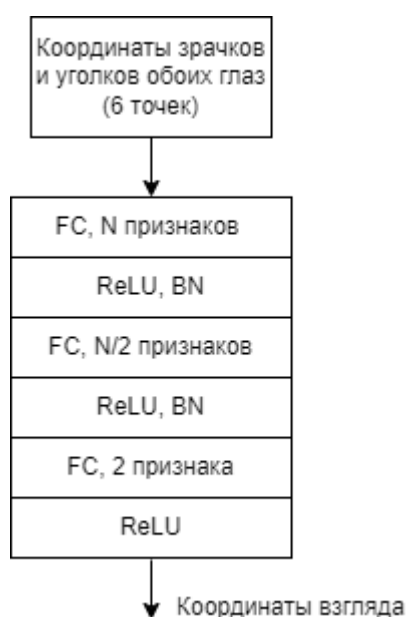


Рисунок 52 – Полносвязная архитектура

На рисунках 53 и 54 приведены график обучения и результаты тестирования (средний квадрат ошибки для ячеек экрана) полносвязной архитектуры с максимальной размерностью скрытого пространства 256.

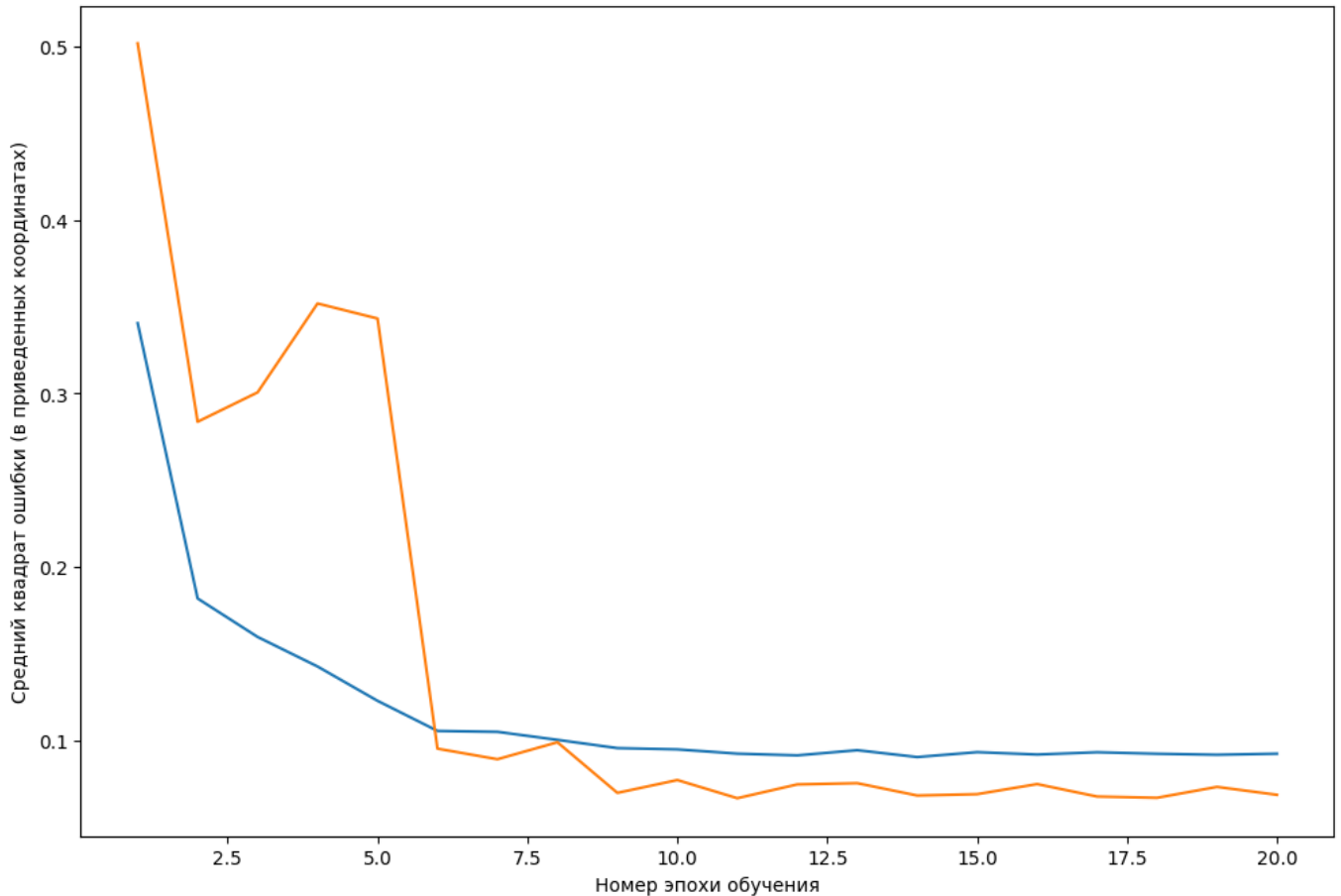


Рисунок 53 – График обучения полносвязной архитектуры с максимальной размерностью скрытого пространства 256:
синяя линия – результаты обучения для тренировочной выборки;
оранжевая линия – результаты обучения для валидационной выборки

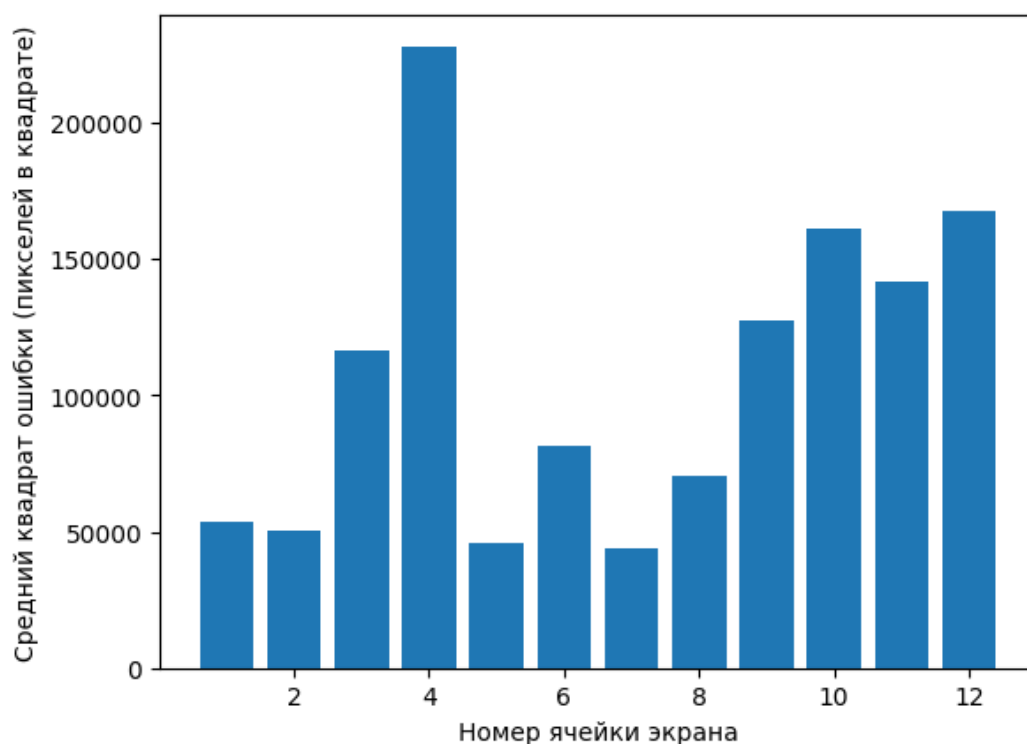


Рисунок 54 – Средний квадрат ошибки определения координат для каждой ячейки экрана (полносвязная архитектура, максимальная размерность скрытого пространства 256)

Обучение полносвязной архитектуры заняло около 2,5 минут, общее количество обучаемых параметров оказалось равным 37250. Величина среднего квадрата ошибки составила 107241 пикселей в квадрате, что соответствует среднеквадратичной ошибке 327 пикселей (или 8,5 см). Также можно отметить высокую нестабильность процесса обучения (по графику функции потерь на валидационной выборке).

На рисунках 55 и 56 приведены график обучения и результаты тестирования (средний квадрат ошибки для ячеек экрана) полносвязной архитектуры с максимальной размерностью скрытого пространства 512.

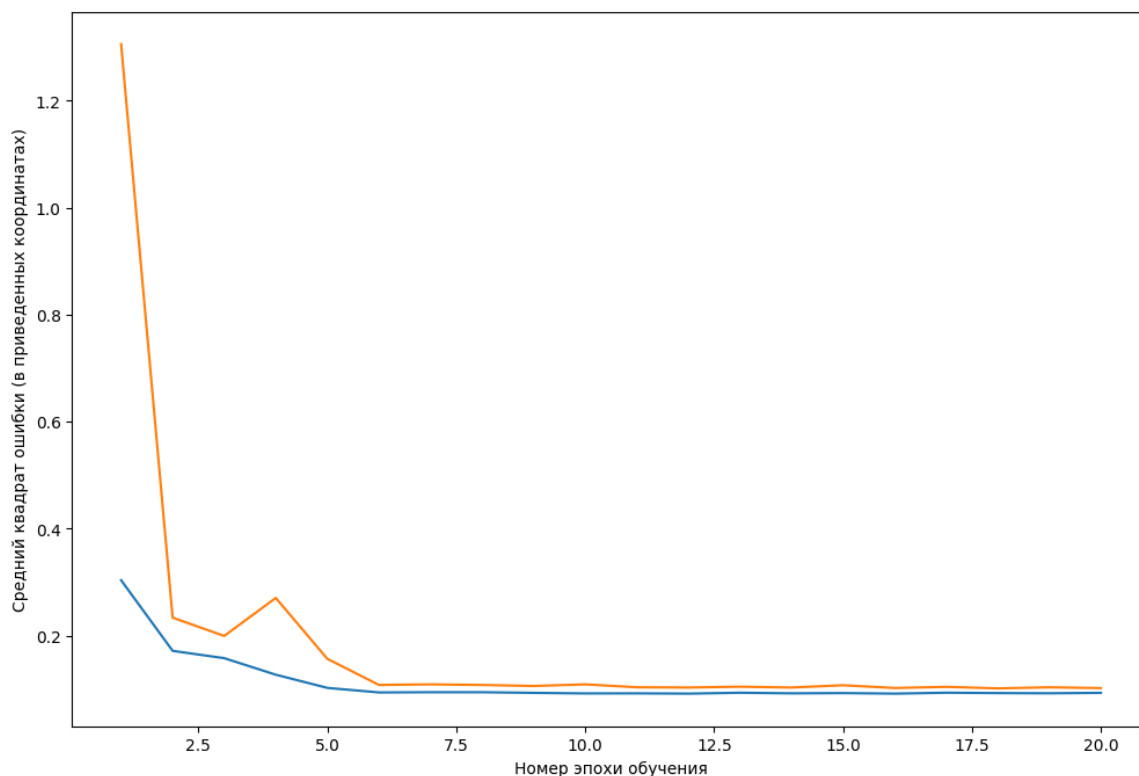


Рисунок 55 – График обучения полносвязной архитектуры с максимальной размерностью скрытого пространства 512:
синяя линия – результаты обучения для тренировочной выборки;
оранжевая линия – результаты обучения для валидационной выборки

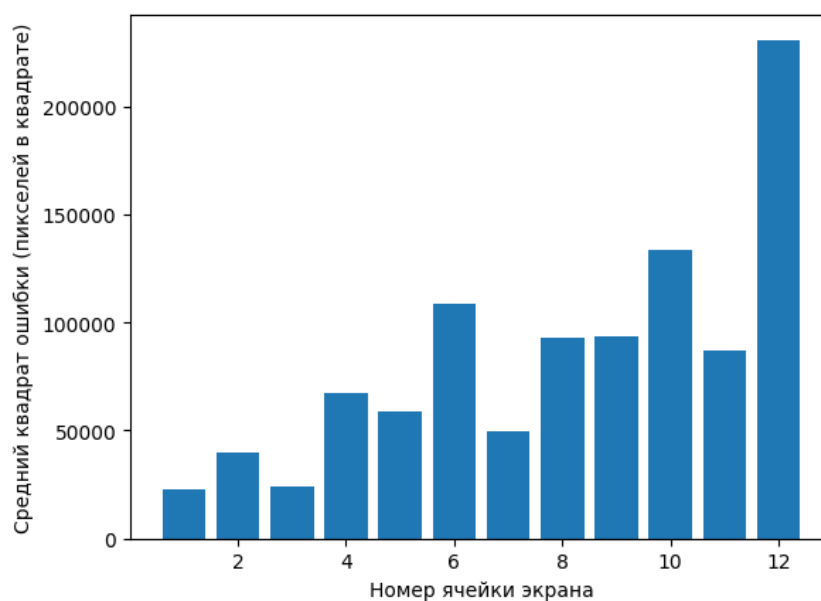


Рисунок 56 – Средний квадрат ошибки определения координат для каждой ячейки экрана (полносвязная архитектура, максимальная размерность скрытого пространства 512)

Обучение данного варианта полносвязной архитектуры заняло около 2,6 минут, количество обучаемых параметров оказалось равным 140034. Величина среднего квадрата ошибки составила 85501 пикселей в квадрате, что соответствует среднеквадратичной ошибке 292 пикселя (или 7,6 см).

Полученные результаты для каждой архитектуры сведены в таблицу 3.

Таблица 3 – Результаты обучения различных архитектур

Название архитектуры	Количество параметров	Среднеквадратическая ошибка, см	Время обучения, мин
Базовая	297390	1,7	90,5
«Гибридная»	4517762	1,7	83,9
Один глаз (первая вариация гиперпараметров)	150446	1,8	93,7
Один глаз (вторая вариация гиперпараметров)	150446	2,6	92,3
Полносвязная (размерность скрытого пространства 256)	37250	8,5	2,5
Полносвязная (размерность скрытого пространства 512)	140034	7,6	2,6

Таким образом, наименьшая среднеквадратическая ошибка, выраженная в сантиметрах, получена для базовой архитектуры. Тем не менее архитектура, обрабатывающая один глаз, позволят сократить количество параметров практически в 2 раза при сравнительно малом увеличении ошибки.

3.5 Разработка тестовой программы и модуля анализа движения глаз

С целью проверки работоспособности системы была разработана тестовая программа, позволяющая определить ячейку экрана устройства (персонального компьютера), на который в данный момент времени смотрит пользователь, и оценить

изменения глазодвигательной активности, характеризующие текущее психоэмоциональное состояние.

Алгоритм работы тестовой программы включает в себя 2 этапа:

- калибровка;
- собственно тестирование.

Калибровка заключается в выборе пользователем такого положения относительно веб-камеры, которое обеспечивает наиболее точное определение координат взгляда (в данном случае – центрального положения в кадре). На рисунке 57 представлен интерфейс калибровки.

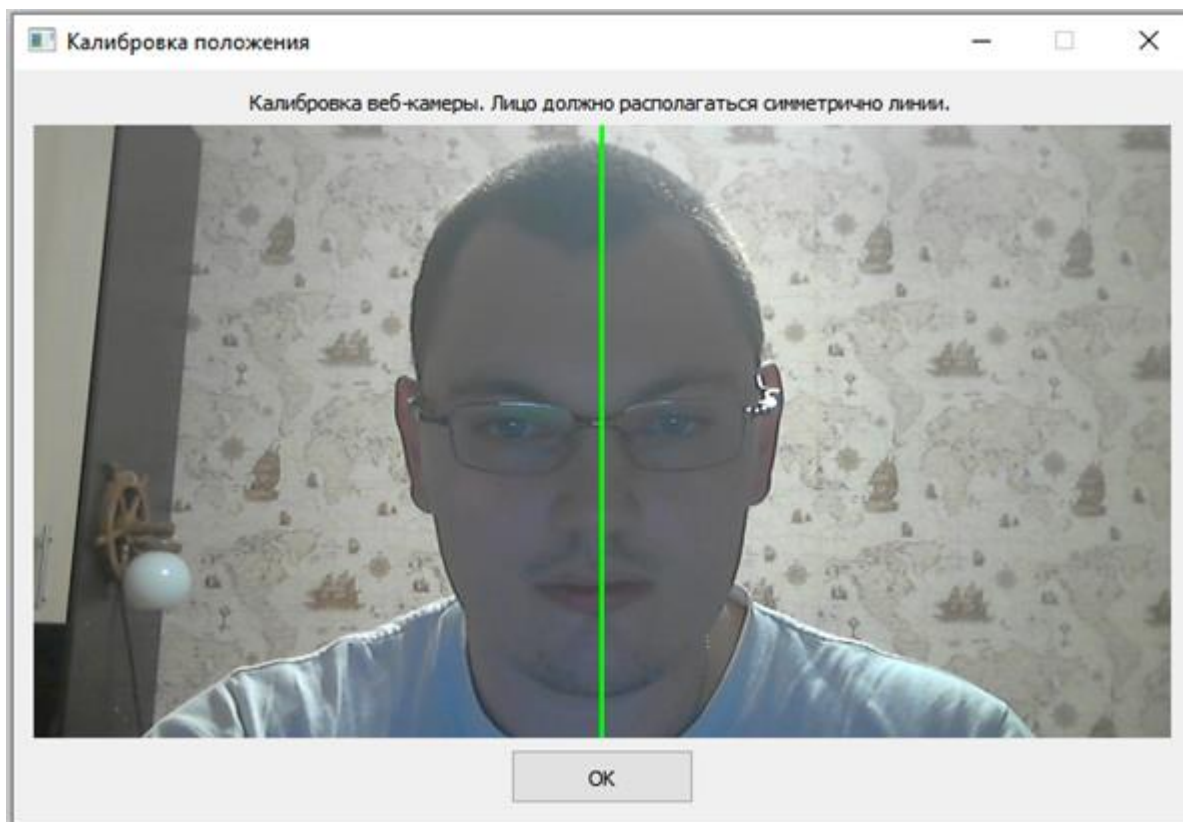


Рисунок 57 – Интерфейс калибровки

Положение, которое необходимо занять пользователю, задано в кадре при помощи линии, а также описано в сопроводительном тексте. При нажатии кнопки «ОК» в полноэкранном режиме открывается интерфейс тестирования (рисунок 58).

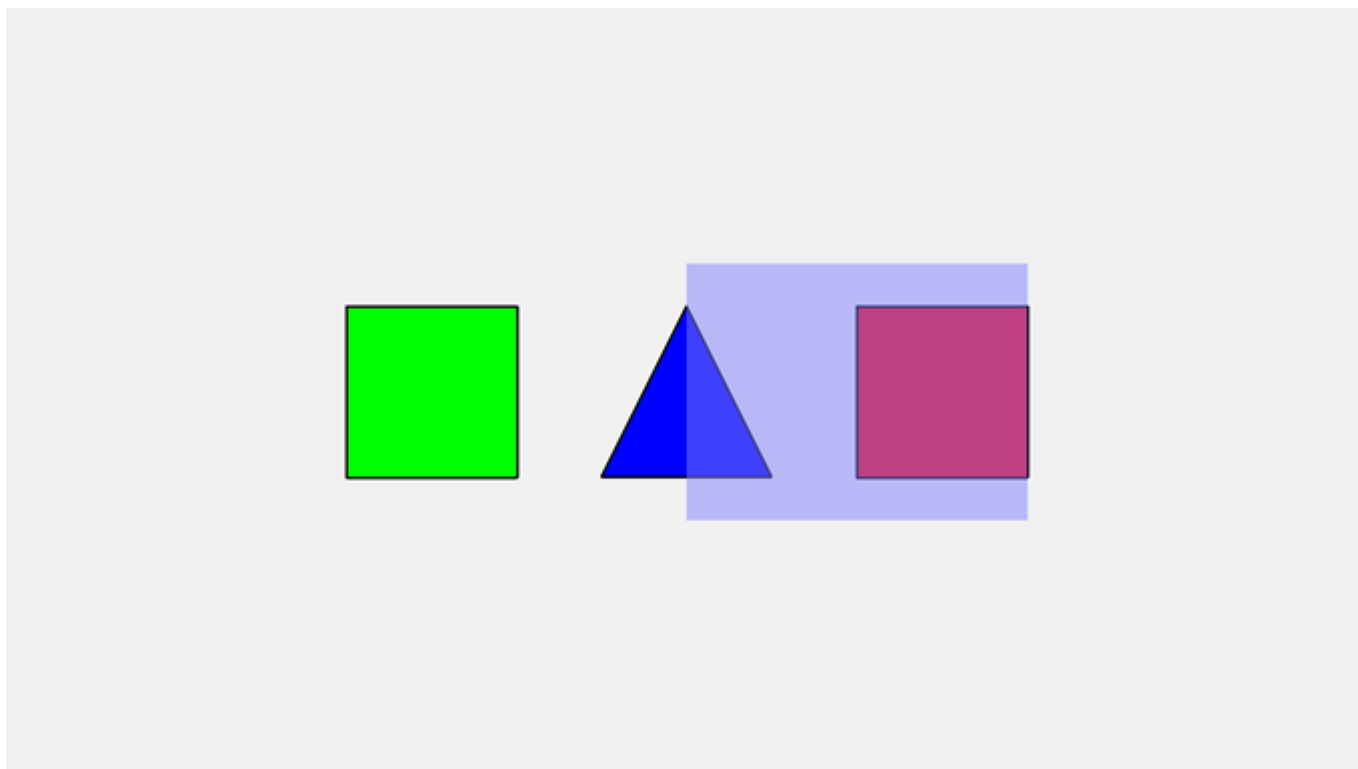


Рисунок 58 – Интерфейс тестирования

Поверх окна тестирования нанесен полупрозрачный прямоугольник, размеры которого соответствуют размерам одной ячейки экрана (400 x 300 пикселей). При изменении направления взгляда пользователя прямоугольник перемещается в ту ячейку экрана, которая соответствует рассчитанным координатам взгляда.

Если лицо или глаза в ходе тестирования распознать не удалось, окно программы окрашивается красной рамкой (рисунок 59).

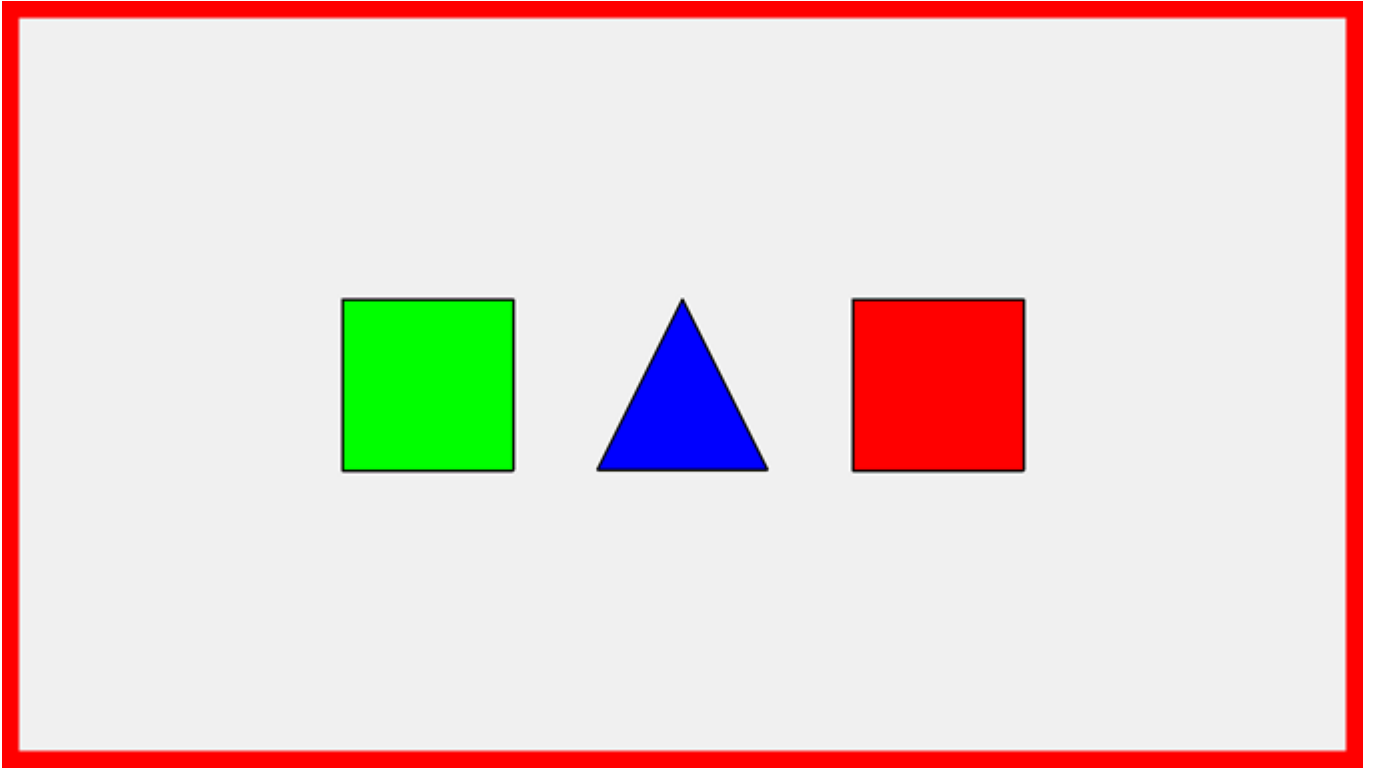


Рисунок 59 – Интерфейс тестирования в том случае, если лицо или глаза не распознаны

Изменения глазодвигательной активности определяются скоростью изменения координат взгляда

$$v_{gaze} = \frac{K_{cm} \sqrt{(x_{new} - x_{old})^2 + (y_{new} - y_{old})^2}}{t}, \quad (8)$$

где t – время смены кадра (для видеопотока 30 кадров в секунду $t = 33$ мс);

$K_{cm} = 0,026$ – коэффициент перевода пикселей в сантиметры;

x_{old}, y_{old} – прежние координаты взгляда;

x_{new}, y_{new} – текущие координаты взгляда.

Упростим формулу (8), устранив деление на временную компоненту. Получим выражение для расстояния изменения взгляда

$$\Delta_{gaze} = K_{cm} \sqrt{(x_{new} - x_{old})^2 + (y_{new} - y_{old})^2}. \quad (9)$$

Определим пороговое значение расстояния изменения взгляда, по превышении которого пользователь будет получать сообщение о повышении глазодвигательной активности (и вероятном психоэмоциональном возбуждении). В качестве порогового значения для той или иной архитектуры нейронной сети представляется разумным принять величину среднеквадратической ошибки, полученной в результате обучения данной архитектуры.

Сообщение о повышении глазодвигательной активности отображается в виде зеленой рамки окна тестовой программы (рисунок 60).

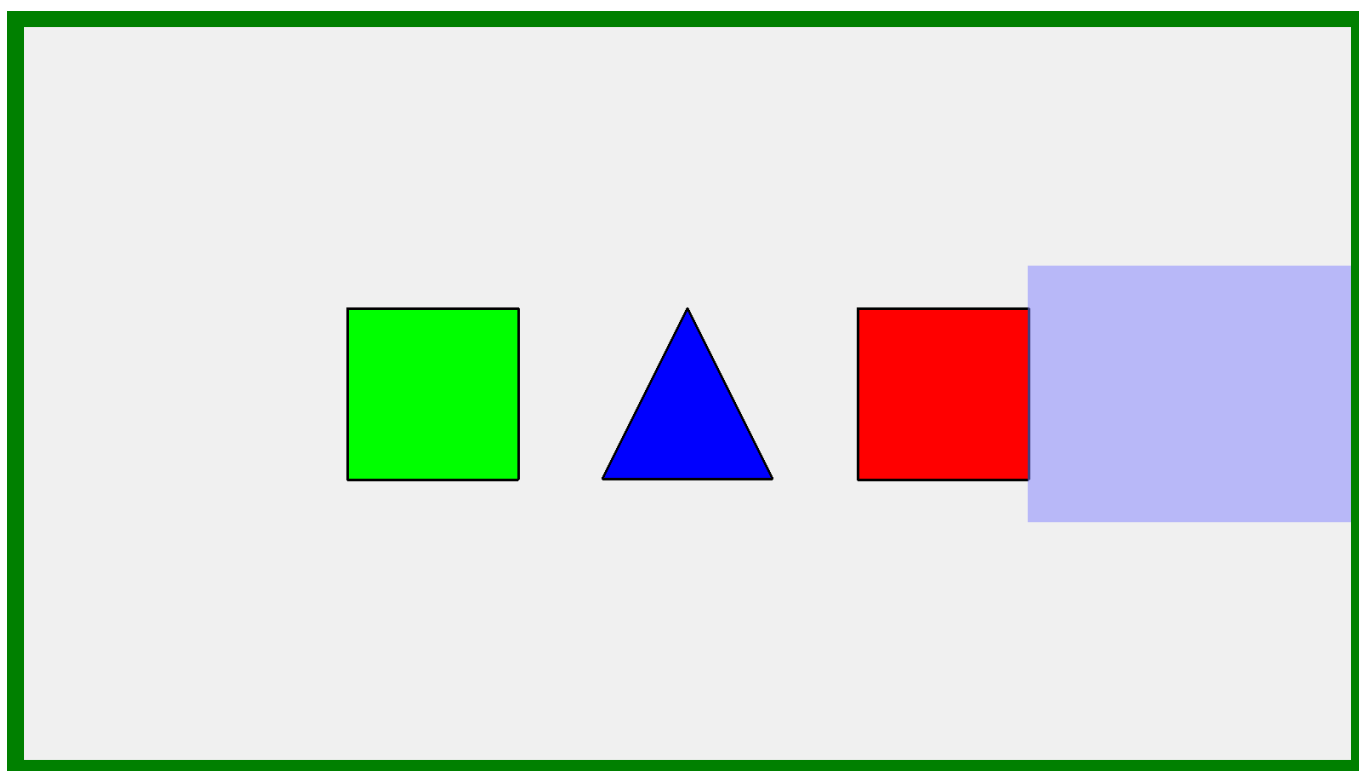


Рисунок 60 – Интерфейс тестирования в случае повышенной глазодвигательной активности

Исходный код тестовой программы приведен в приложении В. Алгоритм работы программы представлен на рисунке 61.

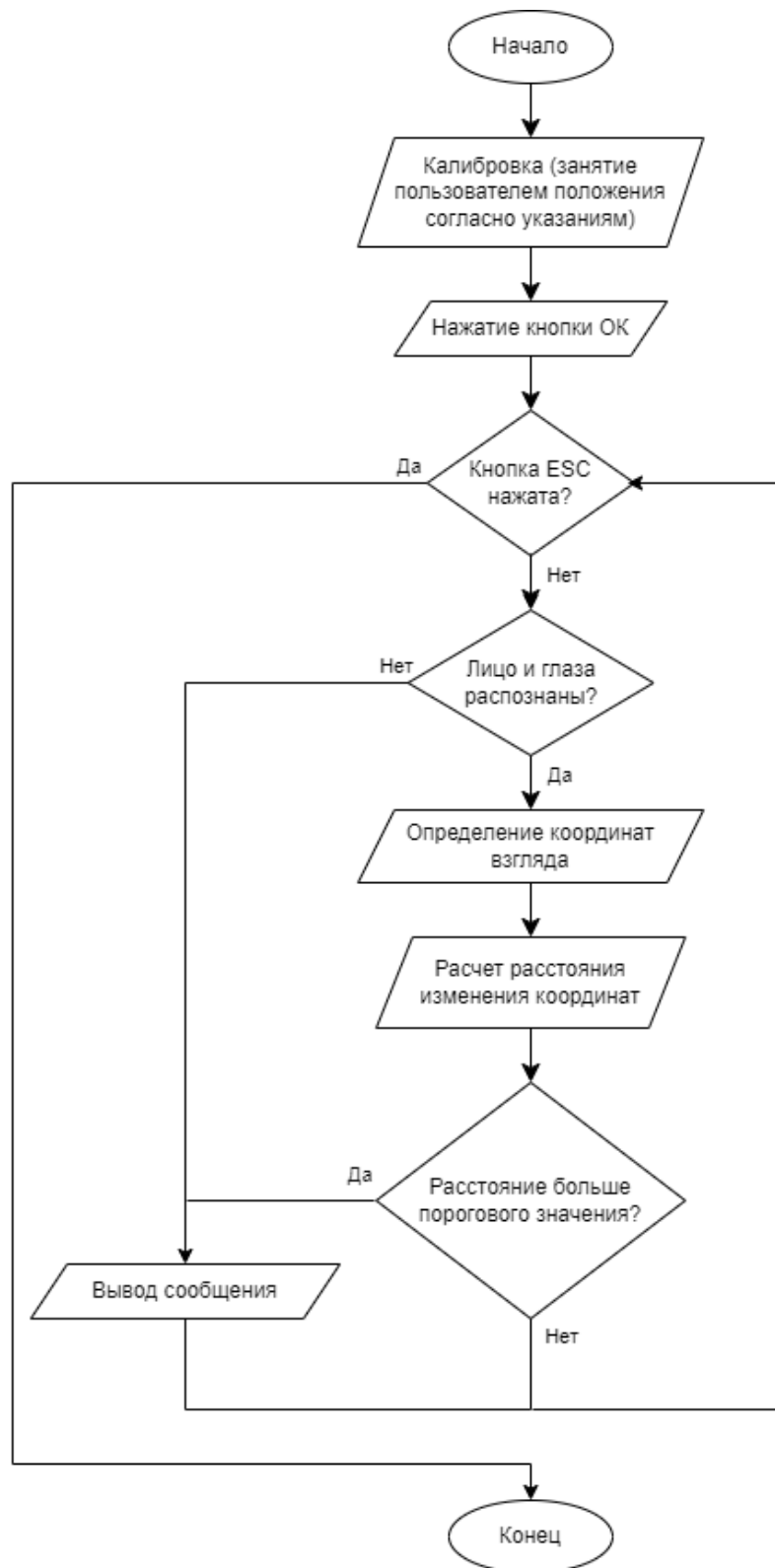


Рисунок 61 – Блок-схема работы тестовой программы

Как показывают эксперименты, разработанная система обеспечивает невысокую, но в целом приемлемую точность определения координат взгляда. Повысить точность позволило бы обучение на большем количестве данных.

4 Технико-экономическое обоснование разработки программного обеспечения

Затраты на разработку программного обеспечения (10) состоят из затрат на расходные материалы и заработную плату исполнителям, затрат на использование оборудования, затрат на организацию рабочих мест и затрат на накладные расходы

$$З = З_{\text{м}} + З_{\text{п}} + З_{\text{эвм}} + Н_{\text{р}}, \quad (10)$$

где $З_{\text{м}}$ – затраты на расходные материалы;

$З_{\text{п}}$ – заработная плата исполнителей;

$З_{\text{эвм}}$ – затраты на использование ЭВМ;

$Н_{\text{р}}$ – накладные расходы.

Ориентировочный расчет затрат на расходные материалы представлен в таблице 4.

Таблица 4 – Расчет стоимости расходных материалов

Наименование статьи расходов	Количество	Цена за ед., р.	Сумма, р.
1 Бумага, упаковка	1	349	349
2 Письменные принадлежности, набор	1	340	340
Итого 689 р.			
Транспортно-заготовительные расходы (ТРЗ) (10 %) 68,9 р.			
Всего 757,9 р.			

Заработная плата исполнителей (разработчиков задачи)

$$З_{\text{п}} = З_{\text{о}} + З_{\text{д}} + Н_{\text{с}}, \quad (11)$$

где $З_{\text{о}}$ – основная заработная плата разработчика;

$З_{\text{д}}$ – дополнительная заработная плата;

H_c – начисления на заработную плату.

Расчет основной заработной платы при дневной оплате труда исполнителей проводили на основе данных по окладам и графику занятости исполнителей (12):

$$Z_o = T \cdot O_m, \quad (12)$$

где T – число месяцев проекта;

O_m – месячный оклад исполнителя.

Для реализации программного обеспечения привлекается один профильный специалист. Заработная плата одного разработчика, необходимого для реализации поставленной задачи в установленные сроки в среднем составляет 70000 (семьдесят тысяч) рублей. Сроки реализации проекта составляют 3 месяца. Таким образом, основная заработная плата составляет

$$Z_o = 3 \cdot 70000 \text{ р.} = 210000 \text{ р.} \quad (13)$$

Расходы на дополнительную заработанную плату учитывают все выплаты непосредственно исполнителям за время, не проработанное на производстве, но предусмотренное законодательством, в том числе: оплата очередных отпусков, компенсация за недоиспользованный отпуск и др. Величина этих выплат составляет 20% от размера основной заработной платы

$$Z_d = 0,2 \cdot Z_o. \quad (14)$$

Учитывая рассчитанное значение основной заработной платы, получаем

$$Z_d = 0,2 \cdot 210000 \text{ р.} = 42000 \text{ р.} \quad (15)$$

Согласно налоговому кодексу РФ, отчисления на социальные нужды включают отчисления в пенсионный фонд РФ, фонд социального страхования, фонды

обязательного медицинского страхования (федеральный и территориальный фонды)

$$H_c = (Z_o + Z_d) \cdot H_{\text{соц}}, \quad (16)$$

где $H_{\text{соц}}$ – отчисления с заработной платы в виде единого социального налога (30 %).

Отчисления на социальные нужды составляют

$$H_c = (210000 \text{ р.} + 42000 \text{ р.}) \cdot 0,3 = 75600 \text{ р.} \quad (17)$$

Затраты, связанные с использованием вычислительной техники, определяются с учетом стоимости одного часа работы оборудования и количества отработанных на этом оборудовании часов. При расчете $Z_{\text{пк}}$ учитывали время на подготовку исходных текстов программ, их отладку и решение контрольного примера. Результаты расчета сведены в таблицу 5.

Таблица 5 – Расчет затрат на использование оборудования

Наименование оборудования	Стоимость часа, р.	Количество отработанных часов, ч.	Сумма, р.
1 Ноутбук HP Laptop 17-by2xxx, 2020 г.	58,5	77	4504,5
2 Веб-камера Logitech C310 HD 720p, 2022 г.	20,8	21	436,8
3 Виртуальный сервер Yandex DataSphere, конфигурация g1.1 с 8 CPU и 1 GPU	311,1	72	22399,2
Итого 27340,5 р.			

Накладные расходы составляют 150-180% от основной заработной платы. Приняв накладные расходы равными 150% от основной заработной платы, рассчитали сумму накладных расходов

$$P_{\text{нак}} = 1,5 \cdot 210000 \text{ р.} = 315000 \text{ р.} \quad (18)$$

Прочие прямые расходы – 7 % от основной заработной платы. Таким образом

$$P_{\text{прям}} = 0,07 \cdot 210000 \text{ р.} = 14700 \text{ р.} \quad (19)$$

Результаты расчетов сведены в таблицу 6.

Таблица 6 – Затраты на разработку программного обеспечения

Наименование статей затрат	Сумма, р.
1 Расходные материалы	757,9
2 Основная заработная плата исполнителей	210000
3 Дополнительная заработная плата	42000
4 Отчисления на социальные нужды	75600
5 Затраты на использование оборудования	27340,5
6 Накладные расходы	315000
7 Прочие прямые расходы	14700
Итого затрат на разработку	685398,4

Вывод: суммарные расходы на разработку программного обеспечения распознавания психоэмоционального состояния оператора беспилотного летательного аппарата составили 685398,4 р. (шестьсот восемьдесят пять тысяч триста девяносто восемь рублей сорок копеек).

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы был проведен обзор основных разработок в области мониторинга психоэмоционального состояния человека. Были рассмотрены программные и технические решения, реализующие метод визуального мониторинга, т.е. мониторинга по изображению, с применением алгоритмов глубокого обучения и компьютерного зрения.

В результате выполнения выпускной квалификационной работы была разработана система распознавания психоэмоционального возбуждения оператора БПЛА путем анализа его глазодвигательной активности. Данная система включает в себя следующие компоненты:

- модуль распознавания лиц;
- модуль определения траектории взгляда;
- модуль анализа движений глаз.

Модуль распознавания лиц представляет собой стандартный алгоритм определения опорных точек лица Mediarpipe Face Mesh. Данный алгоритм позволил добиться высокого быстродействия и качества распознавания лиц. Тем не менее было обнаружено, что модуль распознавания лиц допускает редкие ошибки в определении опорных точек, в связи с чем была реализована проверка результатов работы модуля, состоящая из 2 этапов:

- проверка цвета зрачков и уголков глаз (после перевода изображения в черно-белую расцветку);
- проверка соотношения сторон глаза.

Было принято решение отказаться от готовых наборов обучающих данных и сформировать собственный набор, для чего была разработана программа, осуществляющая запись изображения с веб-камеры (разрешением 1280 x 720) и координат экрана (разрешение экрана 1600 x 900) при нажатии левой кнопки мыши. Нажатие кнопки Enter позволяло записать фото с координатами взгляда вне экрана. Очевидным недостатком собранного набора данных является его малый объем (чуть более 5000 экземпляров).

Было выполнено обучение нескольких архитектур нейронных сетей, определяющих координаты взгляда на экране. Список архитектур выглядит следующим образом:

- архитектура, разработанная исследователями Google Research (базовая);
- нейронная сеть iTracker с измененным форматом входных данных («гибридная»);
- модификация базовой архитектуры для обработки только одного глаза;
- полносвязная архитектура (3 слоя) с максимальной размерностью скрытого пространства 256 и 512.

Наименьшая среднеквадратическая ошибка (1,7 см) была получена для базовой архитектуры, наибольшая (7-8 см) – для обеих полносвязных архитектур. При этом для архитектуры, обрабатывающей один глаз, было обнаружено незначительное повышение среднеквадратической ошибки (около 1 мм) при сокращении числа параметров более чем в 2 раза по сравнению с базовой архитектурой.

Для проверки работоспособности системы была реализована тестовая программа, работающая в полноэкранном режиме. При взгляде пользователя в определенную точку экрана происходит подсвечивание той ячейки экрана (всего 12 ячеек, размеры одной ячейки 400 x 300 пикселей), на которую направлен взгляд пользователя. Изменения глазодвигательной активности пользователя определяются посредством расчета скорости изменения координаты взгляда на экране: если скорость превышает некое пороговое значение (выбранное равным 3,8 см), пользователь получает соответствующее сообщение о повышении глазодвигательной активности (и, как следствие, вероятном психоэмоциональном возбуждении).

Необходимо отметить, что разрабатываемая система распознавания психоэмоционального возбуждения оператора БПЛА носит сугубо рекомендательный характер – окончательное решение обязан принимать специалист по мониторингу психоэмоционального состояния.

СПИСОК ЛИТЕРАТУРЫ

1. Катасев А.С. Нейросетевая модель оценки функционального состояния водителей в системах транспортной безопасности / А.С. Катасев, Д.В. Катасева, А.А. Сибгатуллин // Электроника, фотоника и киберфизические системы. – 2023. – Т. 3. № 1. – С. 69 – 80. – Текст: непосредственный.
2. Kalmar G. Automating, Analyzing and Improving Pupillometry with Machine Learning Algorithms / G. Kalmar et al. // Acta Cybernetica. – 2019. – № 24. – P. 197 – 209. – Текст: непосредственный.
3. Krafka K. Eye Tracking for Everyone / K. Krafka et al. // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. – 2016. – P. 2176 – 2184. – Текст: непосредственный.
4. Valliappan N. Accelerating eye movement research via accurate and affordable smartphone eye tracking / N. Valliappan et al. // Nature Communications. – 2020. – № 11. – 12 p. – Текст: непосредственный.
5. Tseng V. Digital biomarker of mental fatigue / V. Tseng et al. // NPJ Digital Medicine. – 2021. – № 4. – 5 p. – Текст: непосредственный.
6. Минкин В.А. Исследование зависимости психофизиологических характеристик человека от величины торможения вестибулярной системы методом виброизображения / В.А. Минкин, Н.Н. Николаенко // Кубанский Научный Медицинский Вестник. – 2007. – № 6. – С. 23 – 28. – Текст: непосредственный.
7. Wright J. Suspect AI: Vibraimage, Emotion Recognition Technology and Algorithmic Opacity / J. Wright // Science, Technology and Society. – 2021. – 40 p. – Текст: непосредственный.
8. Thakur N. A Complex Activity Based Emotion Recognition Algorithm for Affect Aware Systems / N. Thakur, C.Y. Han // Proceedings of IEEE 8th Annual Computing and Communication Workshop and Conference. – 2018. – P. 748 – 753. – Текст: непосредственный.
9. Wang K. Cascade Attention Networks for Group Emotion Recognition with Face, Body and Image Cues / K. Wang et al. // Proceedings of the ACM International Conference

on Multimodal Interaction. – 2018. – P. 640 – 645. – Текст: непосредственный.

10. Kovalenko S. OperatorEYEV: Operator Dataset for Fatigue Detection Based on Eye Movements, Heart Rate Data, and Video Information / S. Kovalenko et al. // Sensors. – 2023. – № 23. – 35 p. – Текст: непосредственный.

11. Zhang X. MPIIGaze: Real-World Dataset and Deep Appearance-Based Gaze Estimation / X. Zhang et al. // IEEE Transactions on Pattern Analysis and Machine Intelligence. – 2017. – Vol. 41. – P. 162 – 175. – Текст: непосредственный.

12. Шлее М. Qt 5.10. Профессиональное программирование на языке C++ / М. Шлее. – СПб.: БХВ-Петербург, 2018. – 1072 с.: ил. – Текст: непосредственный.

13. Kartynnik Y. Real-time Facial Surface Geometry from Monocular Video on Mobile GPUs / Y. Kartynnik et al. // CVPR Workshop on Computer Vision for Augmented and Virtual Reality. – 2019. – 4 p. – Текст: непосредственный.

14. Soukupova T. Real-Time Eye Blink Detection using Facial Landmarks / T. Soukupova, J. Cech // 21st Computer Vision Winter Workshop. – 2016. – 8 p. – Текст: непосредственный.

15. Златопольский Д.М. Основы программирования на языке Python / Д.М. Златопольский. – 2-е изд. – М.: ДМК Пресс, 2018. – 396 с.: ил. – Текст: непосредственный.

16. Джоши П. Искусственный интеллект с примерами на Python: пер. с англ. / П. Джоши. – СПб.: ООО «Диалектика», 2019. – 448 с. – Текст: непосредственный.

17. Бхаргава А. Грокаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих: пер. с англ. / А. Бхаргава – СПб.: Питер, 2021. – 288 с.: ил. – Текст: непосредственный.

18. Траск Э. Грокаем глубокое обучение: пер. с англ. / Э. Траск. – СПб.: Питер, 2021. – 352 с.: ил. – Текст: непосредственный.

19. Хайкин С. Нейронные сети: полный курс: пер. с англ. / С. Хайкин – 2 изд., испр. – СПб.: ООО «Диалектика», 2020. – 1104 с.: ил. – Текст: непосредственный.

ПРИЛОЖЕНИЕ А

Листинг файлов программы для сбора данных

Файл main.cpp

```
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include "dataloader.h"

int main(int argc, char *argv[])
{
    QCoreApplication::setAttribute(Qt::AA_EnableHighDpiScaling);
    QGuiApplication app(argc, argv);
    qmlRegisterType<DataLoader>("DataLoader", 1, 0, "DataLoader");
    QQmlApplicationEngine engine;
    const QUrl url(QStringLiteral("qrc:/main.qml"));
    QObject::connect(&engine, &QQmlApplicationEngine::objectCreated,
        &app, [url](QObject *obj, const QUrl &objUrl) {
        if (!obj && url == objUrl)
            QCoreApplication::exit(-1);
    }, Qt::QueuedConnection);
    engine.load(url);

    return app.exec();
}
```

Файл dataloader.h

```
#ifndef DATALOADER_H
#define DATALOADER_H

#include <QObject>
#include <opencv2/dnn/dnn.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/objdetect/face.hpp>
#include <opencv2/objdetect/objdetect.hpp>
#include <opencv2/imgcodecs/imgcodecs.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/features2d.hpp>
#include <iostream>
#include <stdio.h>
#include <QDebug>
#include <QDir>
#include <QFile>
#include <QDateTime>
#include <QMouseEvent>
```



```

#include <QGuiApplication>
#include <QTimer>

class DataLoader: public QObject
{
    Q_OBJECT
public:
    DataLoader();
    ~DataLoader();
    Q_INVOKABLE void setFrameAndCoords(double mouseX, double mouseY);
    Q_INVOKABLE void deleteLast();

private:
    cv::VideoCapture video_capture;
    cv::Mat frame;
    QString dir_path = "d://Data_collection_1280x720";
    QString csv_file_name = dir_path + "//coords.csv";
    QStringList jpg_file_names = QStringList();
    QTimer* timer;
    int lastIndex = 0;

public slots:
    void setFrameOnTimer();

signals:
    void errorOccured();
};

#endif // DATALOADER_H

```

Файл dataloder.cpp

```

#include "dataloader.h"

using namespace cv;
using namespace std;

DataLoader::DataLoader()
{
    video_capture.open(0, cv::CAP_DSHOW);
    video_capture.set(cv::CAP_PROP_FRAME_WIDTH, 1280);
    video_capture.set(cv::CAP_PROP_FRAME_HEIGHT, 720);
    QDir dir(dir_path);
    QFile csv(csv_file_name);
    if (!dir.exists())
    {
        dir.mkdir(".");
        csv.open(QIODevice::WriteOnly | QIODevice::Text);
        csv.write("jpg_name,x_norm,y_norm\n");
        csv.close();
    }
}

```

```

else
{
    csv.open(QIODevice::ReadOnly | QIODevice::Text);
    lastIndex = QString(csv.readAll()).split("\n").size()-2;
    csv.close();
}
timer = new QTimer(this);
connect(timer,&QTimer::timeout,this,&DataLoader::getFrameOnTimer);
timer->start(10);
}

void DataLoader::getFrameOnTimer()
{
    //qApp->processEvents();
    //qDebug() << "getFrameOnTimer";
    video_capture.read(frame);
}

void DataLoader::getFrameAndCoords(double mouseX, double mouseY)
{
    //qApp->processEvents();
    //timer->stop();
    video_capture.read(frame);
    //qDebug() << "getFrameAndCoords" << frame.size().width << frame.size().height;
    QString jpg_name = QString::number(QDateTime::currentMSecsSinceEpoch())+".jpg";
    jpg_file_names.append(dir_path+"/"+jpg_name);
    imwrite((dir_path+"/"+jpg_name).toStdString(), frame);
    QFile csv(csv_file_name);
    csv.open(QIODevice::Append | QIODevice::Text);
    csv.write((jpg_name+", "+QString::number(mouseX)+", "+QString::num-
ber(mouseY)+"\n").toStdString().c_str());
    //timer->start(10);
}

void DataLoader::deleteLast()
{
    if (!jpg_file_names.isEmpty())
    {
        QFile delFile(jpg_file_names.last());
        jpg_file_names.removeLast();
        delFile.remove();
        QFile csv(csv_file_name);
        csv.open(QIODevice::ReadWrite | QIODevice::Text);
        QStringList str_list = QString(csv.readAll()).split("\n");
        qDebug() <<str_list;
        str_list = str_list.mid(0,1+lastIndex+jpg_file_names.size());
        //qDebug() <<str_list << 1+lastIndex+jpg_file_names.size();
        csv.write("");
        csv.close();
        csv.open(QIODevice::WriteOnly | QIODevice::Text);
        QString update = str_list.join("\n")+"\n";
        csv.write(update.toStdString().c_str());
        csv.close();
    }
}

```

```

    }
}

DataLoader::~~DataLoader()
{
    video_capture.release();
}

```

Файл main.qml

```

import QtQuick 2.6
import QtQuick.Window 2.2
import QtQuick.Controls 1.1
import QtQuick.Layouts 1.0
import QtQuick.Dialogs 1.0
import DataLoader 1.0

ApplicationWindow {
    id: app_win
    width: 640
    height: 480
    visible: true
    visibility: "FullScreen"
    property int count: 0

    DataLoader {
        id: data_loader
    }

    Item {
        focus: true
        Keys.onEscapePressed: {app_win.close();}
        Keys.onDeletePressed: {
            if (app_win.count==0) return;
            track_point.oldX.pop();
            track_point.oldY.pop();
            app_win.count--;
            data_loader.deleteLast();
            if (track_point.oldX[app_win.count] != -1)
            {
                track_point.x = track_point.oldX[app_win.count];
                track_point.y = track_point.oldY[app_win.count];
                if (!track_point.visible) track_point.visible = true;
            }
            else
                track_point.visible = false;
        }
        Keys.onReturnPressed: {
            data_loader.getFrameAndCoords(-1,-1);
            track_point.oldX.push(-1);
            track_point.oldY.push(-1);
        }
    }
}

```

```

        track_point.visible = false;
        app_win.count++;
    }
}

```

```

MouseArea {
    id: ma_app_win
    anchors.fill: parent
    onClicked: {
        data_loader.getFrameAndCoords(mouseX / app_win.width, mouseY / app_win.height);
        if (!track_point.visible) track_point.visible = true;
        console.log(mouseX, mouseY);
        track_point.x = mouseX - track_point.width / 2;
        track_point.y = mouseY - track_point.height / 2;
        track_point.oldX.push(track_point.x);
        track_point.oldY.push(track_point.y);
        app_win.count++;
    }
}

```

```

Image {
    id: img
    source: "fon.jpg"
    width: app_win.width
    height: app_win.height

```

```

Text {
    id: countText
    text: "Сделано записей в текущем сеансе: "+Number(app_win.count)
    x: 30
    y: 100
    color: "white"
    font.pixelSize: 24
}

```

```

Text {
    id: annotationText
    visible: true
    text: "ЛКМ - записать фото и координаты взгляда на экране,\nDelete - удалить последнюю запись в текущем сеансе,\nEnter - записать фото и координаты взгляда вне экрана,\nEscape - закрыть программу"
    x: 30
    y: 650
    color: "white"
    font.pixelSize: 24
    lineHeight: 1
}
}

```

```

Rectangle {
    id: track_point
    color: "red"
    width: 10

```

```

    height: 10
    x: 0
    y: 0
    property var oldX: [-1]
    property var oldY: [-1]
    visible: false
}
}

```

Файл Data_collect.pro

QT += quick widgets

CONFIG += c++11

You can make your code fail to compile if it uses deprecated APIs.

In order to do so, uncomment the following line.

#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000 # disables all the APIs deprecated before Qt 6.0.0

SOURCES += \
 dataloader.cpp \
 main.cpp

RESOURCES += qml.qrc \
 qrc.qrc

Default rules for deployment.

qnx: target.path = /tmp/\${TARGET}/bin
 else: unix:!android: target.path = /opt/\${TARGET}/bin
 !isEmpty(target.path): INSTALLS += target

HEADERS += \
 dataloader.h

INCLUDEPATH += D:\opencv\opencv\build\include

LIBS += D:\opencv\opencv-build\bin\libopencv_core454.dll
 LIBS += D:\opencv\opencv-build\bin\libopencv_highgui454.dll
 LIBS += D:\opencv\opencv-build\bin\libopencv_imgcodecs454.dll
 LIBS += D:\opencv\opencv-build\bin\libopencv_imgproc454.dll
 LIBS += D:\opencv\opencv-build\bin\libopencv_features2d454.dll
 LIBS += D:\opencv\opencv-build\bin\libopencv_calib3d454.dll
 LIBS += D:\opencv\opencv-build\bin\libopencv_objdetect454.dll
 LIBS += D:\opencv\opencv-build\bin\libopencv_videoio454.dll
 LIBS += D:\opencv\opencv-build\bin\libopencv_dnn454.dll

ПРИЛОЖЕНИЕ Б

Листинг исходного кода для обучения нейронных сетей

Создание функций предобработки набора данных

```
mp_face_mesh = mediapipe.solutions.face_mesh
RIGHT_IRIS = [474, 475, 476, 477]
LEFT_IRIS = [469, 470, 471, 472]
R_H_LEFT = 33 # right eye right most landmark
R_H_RIGHT = 133 # right eye left most landmark
R_H_PUPIL = 468
L_H_LEFT = 362 # left eye right most landmark
L_H_RIGHT = 263 # left eye left most landmark
L_H_PUPIL = 473
LEFT_EYE_3_POINTS = [473, 263, 362]
RIGHT_EYE_3_POINTS = [468, 133, 33]
LEFT_EYE_6_POINTS = [385, 380, 387, 373, 362, 263]
RIGHT_EYE_6_POINTS = [160, 144, 158, 153, 33, 133]

def distance(point1, point2):
    return ((point1[0] - point2[0])**2 + (point1[1] - point2[1])**2)**0.5

def get_ear(mesh, idxs):
    return (distance(mesh[idxs[0]], mesh[idxs[1]]) + distance(mesh[idxs[2]], mesh[idxs[3]]))/ 2 / distance(mesh[idxs[4]], mesh[idxs[5]])

def get_is_darker(gray_frame, mesh, idxs):
    pupils_darker = []
    pupils_darker.append(gray_frame[mesh[idxs[0]][1],mesh[idxs[0]][0]]<= gray_frame[mesh[idxs[1]][1],mesh[idxs[1]][0]])
    pupils_darker.append(gray_frame[mesh[idxs[0]][1],mesh[idxs[0]][0]]<= gray_frame[mesh[idxs[2]][1],mesh[idxs[2]][0]])
    return sum(pupils_darker)

def create_frames(input_path):
    images_list = list(Path(input_path).rglob('*.jpg'))
    csv_path = list(Path(input_path).rglob('*.csv'))[0]
    point_map = np.zeros((900, 1600))
    segment_map = np.zeros((3, 4))
    if not len(images_list) > 1:
        return map
    csv_data = pd.read_csv(csv_path).reset_index()
    input_coords_df = pd.DataFrame(columns=['jpg_name',
                                             'left_inner_x',
                                             'left_outer_x',
                                             'left_inner_y',
                                             'left_outer_y',
                                             'right_inner_x',
                                             'right_outer_x',
```

```

        'right_inner_y',
        'right_outer_y',
        'left_pupil_x',
        'left_pupil_y',
        'right_pupil_x',
        'right_pupil_y',
        'left_pupil_as_center_x',
        'left_pupil_as_center_y',
        'right_pupil_as_center_x',
        'right_pupil_as_center_y',
        'target_x_norm',
        'target_y_norm',
        'target_x',
        'target_y',
        'screen_segment'])

with mp_face_mesh.FaceMesh(
    static_image_mode=True,
    max_num_faces=1,
    refine_landmarks=True,
    min_detection_confidence=0.65,
    min_tracking_confidence=0.65) as face_mesh:
    counter = 0
    for img in images_list:
        frame = cv2.imread(str(img))
        rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        res = face_mesh.process(rgb_frame)
        if not res.multi_face_landmarks:
            csv_data = csv_data[csv_data['jpg_name'] != str(img).split('/')[-1]].reset_index(drop=True)
            os.remove(str(img))
            print(str(img).split('/')[-1]+' deleted, face not found')
            counter +=1
        else:
            gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
            h, w = frame.shape[:2]
            norm_meshpoints = np.array([np.array([p.x, p.y]).astype(float) for p in res.multi_face_landmarks[0].landmark])
            meshpoints = np.array([np.array([p.x*w, p.y*h]).astype(int) for p in res.multi_face_landmarks[0].landmark])
            is_darker = get_is_darker(gray_frame, meshpoints, LEFT_EYE_3_POINTS) +
            get_is_darker(gray_frame, meshpoints, RIGHT_EYE_3_POINTS)
            ear = get_ear(meshpoints, LEFT_EYE_6_POINTS) + get_ear(meshpoints, RIGHT_EYE_6_POINTS)
            if is_darker < 3 or ear/2 < 0.2 or int(csv_data[csv_data['jpg_name']==str(img).split('/')[-1]].reset_index(drop=True).loc[0, 'x_norm'])== -1:
                os.remove(str(img))
                print(str(img).split('/')[-1]+' deleted, pupils not found '+
                    str(round(float(csv_data[csv_data['jpg_name']==str(img).split('/')[-1]].reset_index(drop=True).loc[0, 'x_norm'])*1600))+ " "+
                    str(round(float(csv_data[csv_data['jpg_name']==str(img).split('/')[-1]].reset_index(drop=True).loc[0, 'y_norm'])*900)))
                csv_data = csv_data[csv_data['jpg_name'] != str(img).split('/')[-1]].reset_index(drop=True)
                counter +=1
            continue

```

```

print(str(img).split('/')[-1]+' accepted')
(l_cx, l_cy), l_radius = cv2.minEnclosingCircle(meshpoints[LEFT_IRIS])
(r_cx, r_cy), r_radius = cv2.minEnclosingCircle(meshpoints[RIGHT_IRIS])
center_left = np.array([l_cx, l_cy], dtype=np.int32)
center_right = np.array([r_cx, r_cy], dtype=np.int32)
tx=round(float(csv_data[csv_data['jpg_name']==str(img).split('/')[-1]].reset_index(drop=True).
loc[0, 'x_norm'])*1600)
ty=round(float(csv_data[csv_data['jpg_name']==str(img).split('/')[-1]].reset_index(drop=True).
loc[0, 'y_norm'])*900)
label = tx//400+ty//300*4 if tx != -1 and ty != -1 else 12
if tx != -1600 and ty != -900:
    point_map[ty,tx]+= 1
    segment_map[ty//300, tx//400] += 1
new_df = pd.DataFrame({'jpg_name': str(img).split('/')[-1],
    'right_inner_x':norm_meshpoints[R_H_LEFT][0],
    'right_inner_y':norm_meshpoints[R_H_LEFT][1],
    'left_inner_x':norm_meshpoints[L_H_RIGHT][0],
    'left_inner_y':norm_meshpoints[L_H_RIGHT][1],
    'right_outer_x':norm_meshpoints[R_H_RIGHT][0],
    'right_outer_y':norm_meshpoints[R_H_RIGHT][1],
    'left_outer_x':norm_meshpoints[L_H_LEFT][0],
    'left_outer_y':norm_meshpoints[L_H_LEFT][1],
    'left_pupil_x':norm_meshpoints[L_H_PUPIL][0],
    'left_pupil_y':norm_meshpoints[L_H_PUPIL][1],
    'right_pupil_x':norm_meshpoints[R_H_PUPIL][0],
    'right_pupil_y':norm_meshpoints[R_H_PUPIL][1],
    'left_pupil_as_center_x':center_left[0],
    'left_pupil_as_center_y':center_left[1],
    'right_pupil_as_center_x':center_right[0],
    'right_pupil_as_center_y':center_right[1],
    'target_x_norm':csv_data[csv_data['jpg_name']==str(img).split('/')[-1]].reset_in-
dex(drop=True).loc[0, 'x_norm'],
    'target_y_norm':csv_data[csv_data['jpg_name']==str(img).split('/')[-1]].reset_in-
dex(drop=True).loc[0, 'y_norm'],
    'target_x':tx,
    'target_y':ty,
    'screen_segment':label
    }, index=[0])
input_coords_df = pd.concat([input_coords_df, new_df], ignore_index=True)
input_coords_df.reset_index(drop=True).to_csv(input_path+"/target_data.csv")
print(f'deleted {counter} files')
return point_map, segment_map

```

Создание класса загрузчика тренировочной, валидационной и тестовой выборки
(для всех архитектур, кроме полносвязной)

```

class EyeTrackingDataset(Dataset):
    def __init__(self, images_path, size_list):
        super().__init__()
        self.images_path = sorted(images_path)

```



```

self.len_ = len(self.images_path)
self.size_list = size_list
self.path_to_input_coords = OUTPUT_PATH+'/target_data.csv'
self.input_coords = pd.read_csv(self.path_to_input_coords)

def __len__(self):
    return self.len_

def load_data(self, file):
    full_image = cv2.imread(file)
    h,w = full_image.shape[:2]
    input_coords = self.input_coords[self.input_coords['jpg_name'] == file.split('/')[-1]].reset_index(drop=True)
    left_inner_x = input_coords.loc[0, 'left_inner_x']
    left_inner_y = input_coords.loc[0, 'left_inner_y']
    right_inner_x = input_coords.loc[0, 'right_inner_x']
    right_inner_y = input_coords.loc[0, 'right_inner_y']
    left_outer_x = input_coords.loc[0, 'left_outer_x']
    left_outer_y = input_coords.loc[0, 'left_outer_y']
    right_outer_x = input_coords.loc[0, 'right_outer_x']
    right_outer_y = input_coords.loc[0, 'right_outer_y']
    left_pupil_x = input_coords.loc[0, 'left_pupil_x']
    left_pupil_y = input_coords.loc[0, 'left_pupil_y']
    right_pupil_x = input_coords.loc[0, 'right_pupil_x']
    right_pupil_y = input_coords.loc[0, 'right_pupil_y']
    left_pupil_as_center_x = input_coords.loc[0, 'left_pupil_as_center_x'] / w
    left_pupil_as_center_y = input_coords.loc[0, 'left_pupil_as_center_y'] / h
    right_pupil_as_center_x = input_coords.loc[0, 'right_pupil_as_center_x'] / w
    right_pupil_as_center_y = input_coords.loc[0, 'right_pupil_as_center_y'] / h
    left_eye = full_image[int(left_pupil_y*h-w*abs(left_outer_x-left_inner_x)/2):int(left_pupil_y*h+w*abs(left_outer_x-left_inner_x)/2),
                           int(left_outer_x*w):int(left_inner_x*w)]
    right_eye = full_image[int(right_pupil_y*h-w*abs(right_inner_x-right_outer_x)/2):int(right_pupil_y*h+w*abs(right_inner_x-right_outer_x)/2),
                           int(right_inner_x*w):int(right_outer_x*w)]
    return [cv2.cvtColor(full_image, cv2.COLOR_BGR2RGB),
            cv2.flip(cv2.cvtColor(left_eye, cv2.COLOR_BGR2RGB), 1),
            cv2.cvtColor(right_eye, cv2.COLOR_BGR2RGB),
            left_pupil_as_center_x,
            left_pupil_as_center_y,
            right_pupil_as_center_x,
            right_pupil_as_center_y,
            left_pupil_x,
            left_pupil_y,
            right_pupil_x,
            right_pupil_y,
            left_inner_x,
            left_inner_y,
            right_inner_x,
            right_inner_y,
            left_outer_x,
            left_outer_y,
            right_outer_x,

```

```

        right_outer_y,
        input_coords.loc[0,'target_x_norm'],
        input_coords.loc[0,'target_y_norm'],
        input_coords.loc[0,'screen_segment']]

def __getitem__(self, index):
    data = self.load_data(str(self.images_path[index]))
    coords_tensors = []
    for i in range(3):
        data[i] = np.array(data[i] / 255, dtype='float32')
        data[i] = self.prepare_data(data[i], self.size_list[i])
    for i in range(3, len(data)-1, 2):
        data[i] = np.array(data[i], dtype='float32')
        data[i+1] = np.array(data[i+1], dtype='float32')
        coords_tensors.append(torch.cat((torch.Tensor(data[i]).unsqueeze(-1),      torch.Tensor(data[i+1]).
unsqueeze(-1)), dim=-1))
    return [str(self.images_path[index]), data[0:3], coords_tensors, data[-1]]

def prepare_data(self, image, sizes=None):
    transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])
    if sizes is not None:
        image = cv2.resize(image, sizes, interpolation = cv2.INTER_AREA)
    image = transform(image)
    return image

```

Создание класса загрузчика тренировочной, валидационной и тестовой выборки
(для полносвязной архитектуры)

```

class EyeTrackingDatasetLinear(Dataset):
    def __init__(self, images_path):
        super().__init__()
        self.images_path = sorted(images_path)
        self.len_ = len(self.images_path)
        self.path_to_input_coords = OUTPUT_PATH+'/target_data.csv'
        self.input_coords = pd.read_csv(self.path_to_input_coords)

    def __len__(self):
        return self.len_

    def load_data(self, file):
        h, w = 720, 1280
        input_coords = self.input_coords[self.input_coords['jpg_name'] == file.split('/')[-1]].reset_index(drop=True)
        left_inner_x = input_coords.loc[0, 'left_inner_x']
        left_inner_y = input_coords.loc[0, 'left_inner_y']
        right_inner_x = input_coords.loc[0, 'right_inner_x']
        right_inner_y = input_coords.loc[0, 'right_inner_y']

```

```

left_outer_x = input_coords.loc[0, 'left_outer_x']
left_outer_y = input_coords.loc[0, 'left_outer_y']
right_outer_x = input_coords.loc[0, 'right_outer_x']
right_outer_y = input_coords.loc[0, 'right_outer_y']
left_pupil_x = input_coords.loc[0, 'left_pupil_x']
left_pupil_y = input_coords.loc[0, 'left_pupil_y']
right_pupil_x = input_coords.loc[0, 'right_pupil_x']
right_pupil_y = input_coords.loc[0, 'right_pupil_y']
left_pupil_as_center_x = input_coords.loc[0, 'left_pupil_as_center_x'] / w
left_pupil_as_center_y = input_coords.loc[0, 'left_pupil_as_center_y'] / h
right_pupil_as_center_x = input_coords.loc[0, 'right_pupil_as_center_x'] / w
right_pupil_as_center_y = input_coords.loc[0, 'right_pupil_as_center_y'] / h
return [left_pupil_as_center_x,
        left_pupil_as_center_y,
        right_pupil_as_center_x,
        right_pupil_as_center_y,
        left_pupil_x,
        left_pupil_y,
        right_pupil_x,
        right_pupil_y,
        left_inner_x,
        left_inner_y,
        right_inner_x,
        right_inner_y,
        left_outer_x,
        left_outer_y,
        right_outer_x,
        right_outer_y,
        input_coords.loc[0, 'target_x_norm'],
        input_coords.loc[0, 'target_y_norm'],
        input_coords.loc[0, 'screen_segment']]

```

```

def __getitem__(self, index):
    data = self.load_data(str(self.images_path[index]))
    coords_tensors = []
    for i in range(0, len(data)-1, 2):
        data[i] = np.array(data[i], dtype='float32')
        data[i+1] = np.array(data[i+1], dtype='float32')
        coords_tensors.append(torch.cat((torch.Tensor(data[i]).unsqueeze(-1),
        torch.Tensor(data[i+1]).
        unsqueeze(-1)), dim=-1))
    return [str(self.images_path[index]), coords_tensors, data[-1]]

```

Создание нейронных сетей

```

class BaselineNet(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.first_eye = torch.nn.Sequential(
            torch.nn.Conv2d(in_channels=3, out_channels=32, kernel_size=7, stride=2),
            torch.nn.BatchNorm2d(32),
            torch.nn.ReLU(),

```

```

torch.nn.AvgPool2d(kernel_size=2),
torch.nn.Conv2d(in_channels=32, out_channels=64, kernel_size=5, stride=2),
torch.nn.ReLU(),
torch.nn.BatchNorm2d(64),
torch.nn.AvgPool2d(kernel_size=2),
torch.nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, stride=1),
torch.nn.BatchNorm2d(128),
torch.nn.ReLU())
self.second_eye = torch.nn.Sequential(
    torch.nn.Conv2d(in_channels=3, out_channels=32, kernel_size=7, stride=2),
    torch.nn.BatchNorm2d(32),
    torch.nn.ReLU(),
    torch.nn.AvgPool2d(kernel_size=2),
    torch.nn.Conv2d(in_channels=32, out_channels=64, kernel_size=5, stride=2),
    torch.nn.ReLU(),
    torch.nn.BatchNorm2d(64),
    torch.nn.AvgPool2d(kernel_size=2),
    torch.nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, stride=1),
    torch.nn.BatchNorm2d(128),
    torch.nn.ReLU())
self.coords = torch.nn.Sequential(
    torch.nn.Linear(in_features=4*2, out_features=128),
    torch.nn.BatchNorm1d(128),
    torch.nn.ReLU(),
    torch.nn.Linear(in_features=128, out_features=16),
    torch.nn.BatchNorm1d(16),
    torch.nn.ReLU(),
    torch.nn.Linear(in_features=16, out_features=16),
    torch.nn.BatchNorm1d(16),
    torch.nn.ReLU())
self.union = torch.nn.Sequential(
    torch.nn.Linear(in_features=16+4096, out_features=8),
    torch.nn.BatchNorm1d(8),
    torch.nn.ReLU(),
    torch.nn.Linear(in_features=8, out_features=4),
    torch.nn.BatchNorm1d(4),
    torch.nn.ReLU(),
    torch.nn.Linear(in_features=4, out_features=2),
    torch.nn.ReLU())

```

```

def forward(self, imgs, coords):
    x_1 = self.first_eye(imgs[0])
    x_2 = self.second_eye(imgs[1])
    x_1 = x_1.reshape(x_1.shape[0], -1)
    x_2 = x_2.reshape(x_2.shape[0], -1)
    x_3 = self.coords(torch.cat(tuple(coords[4:]), dim=-1))
    return self.union(torch.cat((x_1, x_2, x_3), dim=-1))

```

```

class HybridNet(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.first_eye = torch.nn.Sequential(
            torch.nn.Conv2d(in_channels=3, out_channels=96, kernel_size=11),

```

```

torch.nn.BatchNorm2d(96),
torch.nn.ReLU(),
torch.nn.Conv2d(in_channels=96, out_channels=256, kernel_size=5),
torch.nn.BatchNorm2d(256),
torch.nn.ReLU(),
torch.nn.Conv2d(in_channels=256, out_channels=384, kernel_size=3),
torch.nn.BatchNorm2d(384),
torch.nn.ReLU(),
torch.nn.Conv2d(in_channels=384, out_channels=64, kernel_size=1),
torch.nn.BatchNorm2d(64),
torch.nn.ReLU())
self.first_eye_fc = torch.nn.Sequential(
    torch.nn.Linear(in_features=64*9*9, out_features=128),
    torch.nn.BatchNorm1d(128),
    torch.nn.ReLU())
self.second_eye = torch.nn.Sequential(
    torch.nn.Conv2d(in_channels=3, out_channels=96, kernel_size=11),
    torch.nn.BatchNorm2d(96),
    torch.nn.ReLU(),
    torch.nn.Conv2d(in_channels=96, out_channels=256, kernel_size=5),
    torch.nn.BatchNorm2d(256),
    torch.nn.ReLU(),
    torch.nn.Conv2d(in_channels=256, out_channels=384, kernel_size=3),
    torch.nn.BatchNorm2d(384),
    torch.nn.ReLU(),
    torch.nn.Conv2d(in_channels=384, out_channels=64, kernel_size=1),
    torch.nn.BatchNorm2d(64),
    torch.nn.ReLU())
self.second_eye_fc = torch.nn.Sequential(
    torch.nn.Linear(in_features=64*9*9, out_features=128),
    torch.nn.BatchNorm1d(128),
    torch.nn.ReLU())
self.coords = torch.nn.Sequential(
    torch.nn.Linear(in_features=4*2, out_features=128),
    torch.nn.BatchNorm1d(128),
    torch.nn.ReLU(),
    torch.nn.Linear(in_features=128, out_features=128),
    torch.nn.BatchNorm1d(128),
    torch.nn.ReLU())
self.union = torch.nn.Sequential(
    torch.nn.Linear(in_features=128*3, out_features=128),
    torch.nn.BatchNorm1d(128),
    torch.nn.ReLU(),
    torch.nn.Linear(in_features=128, out_features=2))

def forward(self, imgs, coords):
    x_1 = self.first_eye(imgs[0])
    x_2 = self.second_eye(imgs[1])
    x_1 = self.first_eye_fc(x_1.reshape(x_1.shape[0], -1))
    x_2 = self.second_eye_fc(x_2.reshape(x_2.shape[0], -1))
    x_3 = self.coords(torch.cat(tuple(coords[4:]), dim=-1))
    return self.union(torch.cat((x_1, x_2, x_3), dim=-1))

```

```

class LinearNet(torch.nn.Module):
    def __init__(self, hid_dim):
        super().__init__()
        self.fc1 = torch.nn.Sequential(
            torch.nn.Linear(12, hid_dim),
            torch.nn.ReLU(),
            torch.nn.BatchNorm1d(hid_dim)
        )
        self.fc2 = torch.nn.Sequential(
            torch.nn.Linear(hid_dim, hid_dim//2),
            torch.nn.ReLU(),
            torch.nn.BatchNorm1d(hid_dim//2)
        )
        self.fc3 = torch.nn.Sequential(
            torch.nn.Linear(hid_dim//2, 2),
            torch.nn.ReLU(),
        )

    def forward(self, coords):
        coords = torch.cat(tuple(coords[2:]),dim=-1)
        return self.fc3(self.fc2(self.fc1(coords)))

class OneEyeNetAsRegression(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.first_eye = torch.nn.Sequential(
            torch.nn.Conv2d(in_channels=3, out_channels=32, kernel_size=7, stride=2),
            torch.nn.BatchNorm2d(32),
            torch.nn.ReLU(),
            torch.nn.AvgPool2d(kernel_size=2),
            torch.nn.Conv2d(in_channels=32, out_channels=64, kernel_size=5, stride=2),
            torch.nn.ReLU(),
            torch.nn.BatchNorm2d(64),
            torch.nn.AvgPool2d(kernel_size=2),
            torch.nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, stride=1),
            torch.nn.BatchNorm2d(128),
            torch.nn.ReLU())
        self.coords = torch.nn.Sequential(
            torch.nn.Linear(in_features=6, out_features=128),
            torch.nn.BatchNorm1d(128),
            torch.nn.ReLU(),
            torch.nn.Linear(in_features=128, out_features=16),
            torch.nn.BatchNorm1d(16),
            torch.nn.ReLU(),
            torch.nn.Linear(in_features=16, out_features=16),
            torch.nn.BatchNorm1d(16),
            torch.nn.ReLU())
        self.union = torch.nn.Sequential(
            torch.nn.Linear(in_features=16+2048, out_features=8),
            torch.nn.BatchNorm1d(8),
            torch.nn.ReLU(),
            torch.nn.Linear(in_features=8, out_features=4),
            torch.nn.BatchNorm1d(4),

```

```

torch.nn.ReLU(),
torch.nn.Linear(in_features=4, out_features=2),
torch.nn.ReLU())

```

```

def forward(self, imgs, coords):
    x_1 = self.first_eye(imgs[1])
    x_1 = x_1.reshape(x_1.shape[0], -1)
    x_2 = self.coords(torch.cat(tuple([coords[3], coords[5], coords[7]]), dim=-1))
    return self.union(torch.cat((x_1, x_2), dim=-1))

```

Создание функции процесса обучения (для всех архитектур, кроме полносвязной)

```

def train_model(model, criterion, optimizer, scheduler, wts_path, num_epochs=25, model_name='base-
line'):
    since = time.time()
    best_model_wts = model.state_dict()
    best_loss = np.inf
    losses = {'train': [], 'val': []}
    for epoch in range(num_epochs):
        plt.close()
        for phase in ['train', 'val']:
            if phase == 'train':
                model.train(True)
            else:
                model.eval()
            running_loss = 0.0
            for data in dataloaders[phase]:
                _, imgs, coords, _ = data
                imgs = [img.to(device) for img in imgs[1:]]
                coords = [coord.to(device) for coord in coords]
                if phase=="train":
                    optimizer.zero_grad()

            if phase == "eval":
                with torch.no_grad():
                    outputs = model(imgs, coords[:-1])
            else:
                outputs = model(imgs, coords[:-1])
            loss = criterion(outputs, coords[-1])
            if phase == 'train':
                loss.backward()
                optimizer.step()
            running_loss += loss.item()
        if phase=='train':
            scheduler.step()
        epoch_loss = running_loss / dataset_sizes[phase]
        losses[phase].append(epoch_loss)
        with open('./losses_'+phase+'_'+model_name, 'w') as loss_file:
            loss_file.write(' '.join([str(loss_value) for loss_value in losses[phase]]))
        if phase == 'val' and epoch_loss < best_loss:
            best_loss = epoch_loss

```

```

        best_model_wts = model.state_dict()
        torch.save(best_model_wts, './best_'+wts_path.split('/')[1])
    output.clear()
    plt.figure(figsize=(12, 8))
    epochs = [_ for _ in range(1, epoch+2)]
    time_elapsed = time.time() - since
    plt.title('Epoch '+str(epoch+1)+' , elapsed time: '+str(time_elapsed/60))
    if epoch>=1:
        plt.plot(epochs, losses['train'], label='train')
        plt.plot(epochs, losses['val'], label='val')
    else:
        plt.scatter(epochs, losses['train'], label='train')
        plt.scatter(epochs, losses['val'], label='val')
    plt.legend()
    plt.xlabel("Номер эпохи обучения")
    plt.ylabel("Средний квадрат ошибки (в приведенных координатах)")
    plt.show()
    torch.save(model.state_dict(), './last_'+wts_path.split('/')[1])
    return model, losses

```

Создание функции процесса обучения (для полносвязной архитектуры)

```

def train_model_linear(model, criterion, optimizer, scheduler, wts_path, num_epochs=25,
model_name='baseline'):
    since = time.time()
    best_model_wts = model.state_dict()
    best_loss = np.inf
    losses = {'train': [], "val": []}
    for epoch in range(num_epochs):
        plt.close()
        for phase in ['train', 'val']:
            if phase == 'train':
                model.train(True)
            else:
                model.eval()
            running_loss = 0.0
            for data in dataloaders[phase]:
                _, coords, _ = data
                coords = [coord.to(device) for coord in coords]
                if phase=="train":
                    optimizer.zero_grad()

            if phase == "eval":
                with torch.no_grad():
                    outputs = model(coords[:-1])
            else:
                outputs = model(coords[:-1])
            loss = criterion(outputs, coords[-1])
            if phase == 'train':
                loss.backward()
                optimizer.step()

```



```

        running_loss += loss.item()
    if phase=='train':
        scheduler.step()
    epoch_loss = running_loss / dataset_sizes[phase]
    losses[phase].append(epoch_loss)
    with open('./losses_'+phase+'_'+model_name, 'w') as loss_file:
        loss_file.write(' '.join([str(loss_value) for loss_value in losses[phase]]))
    if phase == 'val' and epoch_loss < best_loss:
        best_loss = epoch_loss
        best_model_wts = model.state_dict()
        torch.save(best_model_wts, './best_'+wts_path.split('/')[1])
output.clear()
plt.figure(figsize=(12, 8))
epochs = [_ for _ in range(1,epoch+2)]
time_elapsed = time.time() - since
plt.title('Epoch '+str(epoch+1)+' , elapsed time: '+str(time_elapsed/60))
if epoch>=1:
    plt.plot(epochs, losses['train'], label='train')
    plt.plot(epochs, losses['val'], label='val')
else:
    plt.scatter(epochs, losses['train'], label='train')
    plt.scatter(epochs, losses['val'], label='val')
plt.legend()
plt.xlabel("Номер эпохи обучения")
plt.ylabel("Средний квадрат ошибки (в приведенных координатах)")
plt.show()
torch.save(model.state_dict(), './last_'+wts_path.split('/')[1])
return model, losses

```

Создание функций оценки качества обучения нейронных сетей (для всех архитектур, кроме полносвязной)

```

def evaluate_test(model, criterion, phase='test'):
    sum_loss = 0
    model.eval()
    with torch.no_grad():
        for i in range(len(test_dataset)):
            _, imgs, coords, segment = test_dataset[i]
            imgs = [img.unsqueeze(0).to(device) for img in imgs[1:]]
            coords = [coord.unsqueeze(0).to(device) for coord in coords]
            outputs = model(imgs, coords[:-1])
            outputs[:,0] = outputs[:,0] * 1600
            outputs[:,1] = outputs[:,1] * 900
            coords[-1][:,0] = coords[-1][:,0] * 1600
            coords[-1][:,1] = coords[-1][:,1] * 900
            loss = criterion(outputs, coords[-1])
            sum_loss += loss.item()
    return sum_loss / len(test_dataset)

def get_sample(model):

```

```

model.eval()
smap = np.zeros((3,4))
with torch.no_grad():
    random_character = int(np.random.uniform(0,len(test_dataset)))
    file, imgs, coords, segment = test_dataset[random_character]
    #print(segment)
    smap[segment//4, segment%4] += 100
    imgs = [img.unsqueeze(0).to(device) for img in imgs[1:]]
    coords = [coord.unsqueeze(0).to(device) for coord in coords]
    outputs = model(imgs, coords[:-1]).squeeze()
    outputs[0] = abs(torch.round(1600*outputs[0]))
    outputs[1] = abs(torch.round(900*outputs[1]))
    coords[-1][0][0] = abs(torch.round(1600*coords[-1][0][0]))
    coords[-1][0][1] = abs(torch.round(900*coords[-1][0][1]))
    print(outputs, coords[-1])
    smap[min(899,int(outputs[1]))//300,min(1599,int(outputs[0]))//400] += 50
plt.imshow(smap)

```

```

def get_bars(model, criterion):
    model.eval()
    sample_nums = torch.zeros((12))
    sum_mse = torch.zeros((12))
    for i in range(len(test_dataset)):
        file, imgs, coords, segment = test_dataset[i]
        imgs = [img.unsqueeze(0).to(device) for img in imgs[1:]]
        coords = [coord.unsqueeze(0).to(device) for coord in coords]
        with torch.no_grad():
            outputs = model(imgs, coords[:-1]).squeeze()
            outputs[0] = abs(torch.round(1600*outputs[0]))
            outputs[1] = abs(torch.round(900*outputs[1]))
            coords[-1][0][0] = abs(torch.round(1600*coords[-1][0][0]))
            coords[-1][0][1] = abs(torch.round(900*coords[-1][0][1]))
            sum_mse[segment] += criterion(outputs, coords[-1][0]).item()
            sample_nums[segment] += 1
    bar_data = torch.div(sum_mse, sample_nums).detach().cpu().numpy()
    plt.xlabel("Номер ячейки экрана")
    plt.ylabel("Средний квадрат ошибки (пикселей в квадрате)")
    plt.bar([_ for _ in range(1,13)], bar_data)

```

Создание функций оценки качества обучения нейронных сетей (для полносвязной архитектуры)

```

def evaluate_test_linear(model, criterion, phase='test'):
    sum_loss = 0
    model.eval()
    with torch.no_grad():
        for i in range(len(test_dataset)):
            _, coords, segment = test_dataset[i]
            coords = [coord.unsqueeze(0).to(device) for coord in coords]
            outputs = model(coords[:-1])

```

```

    outputs[:,0] = outputs[:,0] * 1600
    outputs[:,1] = outputs[:,1] * 900
    coords[-1][:,0] = coords[-1][:,0] * 1600
    coords[-1][:,1] = coords[-1][:,1] * 900
    loss = criterion(outputs, coords[-1])
    sum_loss += loss.item()
return sum_loss / len(test_dataset)

```

```

def get_sample_linear(model):
    model.eval()
    smap = np.zeros((3,4))
    with torch.no_grad():
        random_character = int(np.random.uniform(0,len(test_dataset)))
        file, coords, segment = test_dataset[random_character]
        #print(segment)
        smap[segment//4, segment%4] += 100
        coords = [coord.unsqueeze(0).to(device) for coord in coords]
        outputs = model(coords[:-1]).squeeze()
        outputs[0] = abs(torch.round(1600*outputs[0]))
        outputs[1] = abs(torch.round(900*outputs[1]))
        coords[-1][0][0] = abs(torch.round(1600*coords[-1][0][0]))
        coords[-1][0][1] = abs(torch.round(900*coords[-1][0][1]))
        print(outputs, coords[-1])
        smap[min(899,int(outputs[1]))//300,min(1599,int(outputs[0]))//400] += 50
    plt.imshow(smap)

```

```

def getBars_linear(model, criterion):
    model.eval()
    sample_nums = torch.zeros((12))
    sum_mse = torch.zeros((12))
    for i in range(len(test_dataset)):
        file, coords, segment = test_dataset[i]
        coords = [coord.unsqueeze(0).to(device) for coord in coords]
        with torch.no_grad():
            outputs = model(coords[:-1]).squeeze()
            outputs[0] = abs(torch.round(1600*outputs[0]))
            outputs[1] = abs(torch.round(900*outputs[1]))
            coords[-1][0][0] = abs(torch.round(1600*coords[-1][0][0]))
            coords[-1][0][1] = abs(torch.round(900*coords[-1][0][1]))
            sum_mse[segment] += criterion(outputs, coords[-1][0]).item()
            sample_nums[segment] += 1
    bar_data = torch.div(sum_mse, sample_nums).detach().cpu().numpy()
    plt.xlabel("Номер ячейки экрана")
    plt.ylabel("Средний квадрат ошибки (пикселей в квадрате)")
    plt.bar([_ for _ in range(1,13)], bar_data)

```

ПРИЛОЖЕНИЕ В

Листинг файлов тестовой программы

Файл main.cpp

```
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include <mainwindow.h>
#include <QApplication>
#include <QGuiApplication>

int main(int argc, char *argv[])
{
    QCoreApplication::setAttribute(Qt::AA_EnableHighDpiScaling);

    QApplication app(argc, argv);

    MainWindow *mw = new MainWindow;

    return app.exec();
}
```

Файл camerahandler.h

```
#ifndef CAMERAHANDLER_H
#define CAMERAHANDLER_H

#include <QObject>
#include <opencv2/dnn/dnn.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/objdetect/face.hpp>
#include <opencv2/objdetect/objdetect.hpp>
#include <opencv2/imgcodecs/imgcodecs.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/features2d.hpp>
#include <QImage>
#include <QPixmap>

class CameraHandler : public QObject
{
    Q_OBJECT
public:
    explicit CameraHandler(QObject *parent = nullptr);
    ~CameraHandler();
    cv::Mat captureFrame();
};
```

```

    cv::Mat getFrame();
    void releaseCap();
    bool isDone;

public slots:
    void started();
    void finished();

private:
    cv::VideoCapture cap;
    cv::Mat frame;

signals:
    void sendPixmap(QPixmap);
};

#endif // CAMERAHANDLER_H

```

Файл camerahandler.cpp

```

#include "camerahandler.h"

CameraHandler::CameraHandler(QObject *parent) : QObject(parent)
{

}

CameraHandler::~CameraHandler()
{

}

void CameraHandler::releaseCap()
{
    isDone = true;
    cap.release();
}

void CameraHandler::started()
{
    isDone = false;
    cap.open(0, cv::CAP_DSHOW);
    cap.set(cv::CAP_PROP_FRAME_WIDTH, 1280);
    cap.set(cv::CAP_PROP_FRAME_HEIGHT, 720);
    while (true)
    {
        if (isDone) return;
        cap >> frame;
        cv::imwrite("1.jpg", frame);
        cv::cvtColor(frame, frame, cv::COLOR_BGR2RGB);
        cv::line(frame, cv::Point(1280/2-1,0), cv::Point(1280/2-1,719), cv::Scalar(0,255,0),4);
    }
}

```

```

        cv::resize(frame, frame, cv::Size(1280/2,720/2));
        QImage img= QImage((uchar*) frame.data, frame.cols, frame.rows, frame.step, QImage::For-
mat_RGB888);
        emit sendPixmap(QPixmap::fromImage(img));
    }
}

void CameraHandler::finished()
{
    cap.release();
}

```

Файл mainwindow.h

```

#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QDebug>
#include <QLabel>
#include <QPushButton>
#include <QApplication>
#include <QTimer>
#include <QObject>
#include <QVBoxLayout>
#include <QQmlEngine>
#include <QQmlComponent>
#include <QHBoxLayout>
#include <QPainter>
#include <QKeyEvent>
#include <QThread>
#include <QProcess>
#include <QDateTime>
#include "camerahandler.h"
#include "messagewidget.h"

class MainWindow : public QWidget
{
    Q_OBJECT
public:
    explicit MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private:
    QThread* thread;
    QProcess *proc;
    QLabel *label;
    bool isStarted = false;
    CameraHandler *handler;
    cv::Mat frame;
    int currentSegment = -3;

```

```

int currentX = -3;
int currentY = -3;
int currentTime = -3;
double gazeVelocity = -3;
MessageWidget *msg;

public slots:
    void setCalibrationImage(QPixmap);
    void calibrationDone();
    void predictGaze();
    void forceClose();

protected:
    void paintEvent(QPaintEvent* event);
    void keyPressEvent(QKeyEvent* event);
};

#endif // MAINWINDOW_H

```

Файл mainwindow.cpp

```

#include "mainwindow.h"

MainWindow::MainWindow(QWidget *parent) : QWidget(parent)
{
    thread = new QThread;
    handler = new CameraHandler;
    handler->moveToThread(thread);
    connect(thread, SIGNAL(started()), handler, SLOT(started()));
    connect(thread, SIGNAL(finished()), handler, SLOT(finished()));
    connect(handler, &CameraHandler::sendPixmap, this, &MainWindow::setCalibrationImage);
    setWindowTitle("Калибровка положения");
    setFixedSize(1280/2,410);
    label = new QLabel(this);
    label->setAlignment(Qt::AlignCenter);
    QVBoxLayout *layout = new QVBoxLayout;
    QPushButton *button = new QPushButton(this);
    button->setText("OK");
    button->setFixedSize(100,30);
    QLabel *txt = new QLabel("Калибровка веб-камеры. Лицо должно располагаться симметрично ли-
нии.");
    txt->setAlignment(Qt::AlignCenter);
    QHBoxLayout *hl = new QHBoxLayout;
    hl->addStretch();
    hl->addWidget(button);
    hl->addStretch();
    layout->addWidget(txt);
    layout->addWidget(label);
    layout->addLayout(hl);
    setLayout(layout);
    thread->start();
}

```

```

    show();
    connect(button, &QPushButton::clicked, this, &MainWindow::calibrationDone);
}

void MainWindow::setCalibrationImage(QPixmap pixmap)
{
    label->setPixmap(pixmap);
}

void MainWindow::predictGaze()
{
    if (msg->isVisible()) msg->close();
    QString reply = QString(proc->readAll());
    qDebug() << reply;
    QString segment = reply.split(",").at(0);
    QString x = reply.split(",").at(1);
    QString y = reply.split(",").at(2).split(" ").at(0);
    qDebug() << "SXY" << segment << x << y;
    if (segment.at(0) != "(") return;
    if (currentSegment > 0)
        gazeVelocity = sqrt(pow(round(x.toDouble()*1600)-currentX,2) + pow(round(y.toDouble()*900)-currentY,2))*0.26/10;
    else
        gazeVelocity = 0;
    currentSegment = segment.remove(0,1).toInt();
    currentX = round(x.toDouble()*1600);
    currentY = round(y.toDouble()*900); //y.remove(" ").toInt();
    update();
}

void MainWindow::calibrationDone()
{
    msg = new MessageWidget;
    msg->setFixedSize(300,50);
    msg->setWindowFlags(Qt::CustomizeWindowHint);
    connect(msg, &MessageWidget::timeToClose, this, &MainWindow::forceClose);
    disconnect(handler, &CameraHandler::sendPixmap, this, &MainWindow::setCalibrationImage);
    qDebug() << "NOOOOOOOOOOOOOOOOOOOOO";
    isStarted = true;
    qDeleteAll(findChildren<QWidget *>(QString(), Qt::FindDirectChildrenOnly));
    proc = new QProcess;
    proc->start("python predict.py");
    proc->waitForStarted();
    connect(proc, &QProcess::readyReadStandardOutput, this, &MainWindow::predictGaze);
    setWindowState(Qt::WindowFullScreen);
    showFullScreen();
    msg->move(width()/2-msg->width()/2, height()/2);
    msg->show();
    handler->releaseCap();
    thread->quit();
    thread->wait();
}

```



```

MainWindow::~MainWindow()
{
    thread->quit();
    thread->wait();
    if (proc)
    {
        QProcess::execute("taskkill /IM python.exe /F");
        qDebug() << "kill";
        proc->kill();
        delete proc;
    }
    if (handler) handler->~CameraHandler();
}

void MainWindow::paintEvent(QPaintEvent *event)
{
    if (isStarted)
    {
        QPainter painter(this);
        QPen pen;
        pen.setColor("black");
        pen.setWidth(3);
        painter.setPen(pen);
        QPolygon triangle;
        triangle << QPoint(width()/2-100,height()/2+100) << QPoint(width()/2+100,height()/2+100) <<
QPoint(width()/2,height()/2-100);
        QPainterPath path;
        path.addPolygon(triangle);
        painter.fillPath(path, Qt::blue);
        painter.drawPolygon(triangle);
        QRect square1;
        square1.setCoords(triangle.at(0).x()-300, height()/2-100, triangle.at(0).x()-100, height()/2+100);
        painter.fillRect(square1, Qt::green);
        painter.drawRect(square1);
        QRect square2;
        square2.setCoords(triangle.at(1).x()+100, height()/2-100, triangle.at(1).x()+300, height()/2+100);
        painter.fillRect(square2, Qt::red);
        painter.drawRect(square2);
        if (currentSegment >= 0)
        {
            int x = currentSegment%4;
            int y = currentSegment/4;
            QRect rect;
            rect.setCoords(x*400, y*300, x*400+400, y*300+300);
            painter.setPen(QPen(QBrush(),0));
            //painter.setBrush(QBrush(QColor(128, 128, 255, 128)));
            //painter.drawEllipse(QPoint(currentX, currentY), 146, 146);
            painter.fillRect(rect, QBrush(QColor(128, 128, 255, 128)));
            painter.drawRect(rect);
            if (gazeVelocity > 1.7)
            {
                QPolygon activity;

```

```

        activity << QPoint(0,0) << QPoint(0,height()) << QPoint(width(),height()) << QPoint
(width(),0);
        pen.setColor("green");
        pen.setWidth(40);
        painter.setPen(pen);
        painter.setBrush(QBrush());
        painter.drawPolygon(activity);
    }
}
else if (currentSegment == -1 || currentSegment == -2)
{
    QPolygon screen_frame;
    screen_frame << QPoint(0,0) << QPoint(0,height()) << QPoint(width(),height()) << QPoint
(width(),0);
    pen.setColor("red");
    pen.setWidth(40);
    painter.setPen(pen);
    painter.drawPolygon(screen_frame);
}
}
}

```

```

void MainWindow::keyPressEvent(QKeyEvent* event)
{
    if (isStarted && event->key() == Qt::Key::Key_Escape)
    {
        if (proc)
        {
            qDebug() << "kill";
            proc->kill();
            proc->startDetached("taskkill /IM python.exe /F");
            proc->waitForFinished();
            proc->kill();
            delete proc;
        }
        if (handler) handler->~CameraHandler();
        close();
    }
}

```

```

void MainWindow::forceClose()
{
    msg->close();
    if (proc)
    {
        qDebug() << "kill";
        proc->kill();
        proc->startDetached("taskkill /IM python.exe /F");
        proc->waitForFinished();
        proc->kill();
        delete proc;
    }
    if (handler) handler->~CameraHandler();
}

```

```
    close();  
}
```

Файл messagewidget.h

```
#ifndef MESSAGEWIDGET_H  
#define MESSAGEWIDGET_H  
  
#include <QWidget>  
#include <QLabel>  
#include <QHBoxLayout>  
#include <QKeyEvent>  
  
class MessageWidget : public QWidget  
{  
    Q_OBJECT  
public:  
    explicit MessageWidget(QWidget *parent = nullptr);  
  
protected:  
    void keyPressEvent(QKeyEvent *event);  
  
signals:  
    void timeToClose();  
};  
  
#endif // MESSAGEWIDGET_H
```

Файл messagewidget.cpp

```
#include "messagewidget.h"  
  
MessageWidget::MessageWidget(QWidget *parent) : QWidget(parent)  
{  
    QLabel *label = new QLabel;  
    QHBoxLayout *hl = new QHBoxLayout;  
    label->setText("Подгрузка модулей. Ожидайте...");  
    hl->addStretch();  
    hl->addWidget(label);  
    hl->addStretch();  
    setLayout(hl);  
}  
  
void MessageWidget::keyPressEvent(QKeyEvent *event)  
{  
    if (isVisible() && event->key() == Qt::Key::Key_Escape)  
        emit timeToClose();  
}
```

Файл predict.py

```
import torch
import numpy as np
import argparse
import cv2
import mediapipe
import torchvision

class HybridNet(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.first_eye = torch.nn.Sequential(
            torch.nn.Conv2d(in_channels=3, out_channels=96, kernel_size=11),
            torch.nn.BatchNorm2d(96),
            torch.nn.ReLU(),
            torch.nn.Conv2d(in_channels=96, out_channels=256, kernel_size=5),
            torch.nn.BatchNorm2d(256),
            torch.nn.ReLU(),
            torch.nn.Conv2d(in_channels=256, out_channels=384, kernel_size=3),
            torch.nn.BatchNorm2d(384),
            torch.nn.ReLU(),
            torch.nn.Conv2d(in_channels=384, out_channels=64, kernel_size=1),
            torch.nn.BatchNorm2d(64),
            torch.nn.ReLU())
        self.first_eye_fc = torch.nn.Sequential(
            torch.nn.Linear(in_features=64*9*9, out_features=128),
            torch.nn.BatchNorm1d(128),
            torch.nn.ReLU())
        self.second_eye = torch.nn.Sequential(
            torch.nn.Conv2d(in_channels=3, out_channels=96, kernel_size=11),
            torch.nn.BatchNorm2d(96),
            torch.nn.ReLU(),
            torch.nn.Conv2d(in_channels=96, out_channels=256, kernel_size=5),
            torch.nn.BatchNorm2d(256),
            torch.nn.ReLU(),
            torch.nn.Conv2d(in_channels=256, out_channels=384, kernel_size=3),
            torch.nn.BatchNorm2d(384),
            torch.nn.ReLU(),
            torch.nn.Conv2d(in_channels=384, out_channels=64, kernel_size=1),
            torch.nn.BatchNorm2d(64),
            torch.nn.ReLU())
        self.second_eye_fc = torch.nn.Sequential(
            torch.nn.Linear(in_features=64*9*9, out_features=128),
            torch.nn.BatchNorm1d(128),
            torch.nn.ReLU())
        self.coords = torch.nn.Sequential(
            torch.nn.Linear(in_features=4*2, out_features=128),
            torch.nn.BatchNorm1d(128),
            torch.nn.ReLU(),
```

```

        torch.nn.Linear(in_features=128, out_features=128),
        torch.nn.BatchNorm1d(128),
        torch.nn.ReLU())
self.union = torch.nn.Sequential(
    torch.nn.Linear(in_features=128*3, out_features=128),
    torch.nn.BatchNorm1d(128),
    torch.nn.ReLU(),
    torch.nn.Linear(in_features=128, out_features=2),
    torch.nn.ReLU())

```

```

def forward(self, imgs, coords):
    x_1 = self.first_eye(imgs[0].unsqueeze(0))
    x_2 = self.second_eye(imgs[1].unsqueeze(0))
    x_1 = self.first_eye_fc(x_1.reshape(x_1.shape[0], -1))
    x_2 = self.second_eye_fc(x_2.reshape(x_2.shape[0], -1))
    if coords.size(0) > 1:
        coords = torch.cat(tuple(coords), dim=-1)
    x_3 = self.coords(coords)
    return self.union(torch.cat((x_1, x_2, x_3), dim=-1))

```

```

class BaselineNet(torch.nn.Module):

```

```

    def __init__(self):
        super().__init__()
        self.first_eye = torch.nn.Sequential(
            torch.nn.Conv2d(in_channels=3, out_channels=32, kernel_size=7, stride=2),
            torch.nn.BatchNorm2d(32),
            torch.nn.ReLU(),
            torch.nn.AvgPool2d(kernel_size=2),
            torch.nn.Conv2d(in_channels=32, out_channels=64, kernel_size=5, stride=2),
            torch.nn.ReLU(),
            torch.nn.BatchNorm2d(64),
            torch.nn.AvgPool2d(kernel_size=2),
            torch.nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, stride=1),
            torch.nn.BatchNorm2d(128),
            torch.nn.ReLU())
        self.second_eye = torch.nn.Sequential(
            torch.nn.Conv2d(in_channels=3, out_channels=32, kernel_size=7, stride=2),
            torch.nn.BatchNorm2d(32),
            torch.nn.ReLU(),
            torch.nn.AvgPool2d(kernel_size=2),
            torch.nn.Conv2d(in_channels=32, out_channels=64, kernel_size=5, stride=2),
            torch.nn.ReLU(),
            torch.nn.BatchNorm2d(64),
            torch.nn.AvgPool2d(kernel_size=2),
            torch.nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, stride=1),
            torch.nn.BatchNorm2d(128),
            torch.nn.ReLU())
        self.coords = torch.nn.Sequential(
            torch.nn.Linear(in_features=4*2, out_features=128),
            torch.nn.BatchNorm1d(128),
            torch.nn.ReLU(),
            torch.nn.Linear(in_features=128, out_features=16),

```

```

torch.nn.BatchNorm1d(16),
torch.nn.ReLU(),
torch.nn.Linear(in_features=16, out_features=16),
torch.nn.BatchNorm1d(16),
torch.nn.ReLU())
self.union = torch.nn.Sequential(
    torch.nn.Linear(in_features=16+4096, out_features=8),
    torch.nn.BatchNorm1d(8),
    torch.nn.ReLU(),
    torch.nn.Linear(in_features=8, out_features=4),
    torch.nn.BatchNorm1d(4),
    torch.nn.ReLU(),
    torch.nn.Linear(in_features=4, out_features=2),
    torch.nn.ReLU())

```

```

def forward(self, imgs, coords):
    x_1 = self.first_eye(imgs[0].unsqueeze(0))
    x_2 = self.second_eye(imgs[1].unsqueeze(0))
    x_1 = x_1.reshape(x_1.shape[0], -1)
    x_2 = x_2.reshape(x_2.shape[0], -1)
    if coords.size(0) > 1:
        coords = torch.cat(tuple(coords), dim=-1)
    x_3 = self.coords(coords)
    return self.union(torch.cat((x_1, x_2, x_3), dim=-1)).squeeze()

```

```

class OneEyeNetAsRegression(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.first_eye = torch.nn.Sequential(
            torch.nn.Conv2d(in_channels=3, out_channels=32, kernel_size=7, stride=2),
            torch.nn.BatchNorm2d(32),
            torch.nn.ReLU(),
            torch.nn.AvgPool2d(kernel_size=2),
            torch.nn.Conv2d(in_channels=32, out_channels=64, kernel_size=5, stride=2),
            torch.nn.ReLU(),
            torch.nn.BatchNorm2d(64),
            torch.nn.AvgPool2d(kernel_size=2),
            torch.nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, stride=1),
            torch.nn.BatchNorm2d(128),
            torch.nn.ReLU())
        self.coords = torch.nn.Sequential(
            torch.nn.Linear(in_features=6, out_features=128),
            torch.nn.BatchNorm1d(128),
            torch.nn.ReLU(),
            torch.nn.Linear(in_features=128, out_features=16),
            torch.nn.BatchNorm1d(16),
            torch.nn.ReLU(),
            torch.nn.Linear(in_features=16, out_features=16),
            torch.nn.BatchNorm1d(16),
            torch.nn.ReLU())
        self.union = torch.nn.Sequential(
            torch.nn.Linear(in_features=16+2048, out_features=8),

```

```

torch.nn.BatchNorm1d(8),
torch.nn.ReLU(),
torch.nn.Linear(in_features=8, out_features=4),
torch.nn.BatchNorm1d(4),
torch.nn.ReLU(),
torch.nn.Linear(in_features=4, out_features=2),
torch.nn.ReLU())

```

```

def forward(self, imgs, coords):
    x_1 = self.first_eye(imgs[1].unsqueeze(0))
    x_1 = x_1.reshape(x_1.shape[0], -1)
    if coords.size(0) > 1:
        coords = torch.cat(tuple([coords[3], coords[5], coords[7]]), dim=-1)
    x_2 = self.coords(coords)
    return self.union(torch.cat((x_1, x_2), dim=-1))

```

```

RIGHT_IRIS = [474, 475, 476, 477]
LEFT_IRIS = [469, 470, 471, 472]
R_H_LEFT = 33
R_H_RIGHT = 133
R_H_PUPIL = 468
L_H_LEFT = 362
L_H_RIGHT = 263
L_H_PUPIL = 473
LEFT_EYE_3_POINTS = [473, 263, 362]
RIGHT_EYE_3_POINTS = [468, 133, 33]
LEFT_EYE_6_POINTS = [385, 380, 387, 373, 362, 263]
RIGHT_EYE_6_POINTS = [160, 144, 158, 153, 33, 133]

```

```

transform = torchvision.transforms.Compose([
    torchvision.transforms.ColorJitter(),
    torchvision.transforms.ToTensor(),
    torchvision.transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

```

```

def distance(point1, point2):
    return ((point1[0] - point2[0])**2 + (point1[1] - point2[1])**2)**0.5

```

```

def get_ear(mesh, idxs):
    return (distance(mesh[idxs[0]], mesh[idxs[1]]) + distance(mesh[idxs[2]], mesh[idxs[3]]))/ 2 / distance(mesh[idxs[4]], mesh[idxs[5]])

```

```

def get_is_darker(gray_frame, mesh, idxs):
    pupils_darker = []
    pupils_darker.append(gray_frame[mesh[idxs[0]][1], mesh[idxs[0]][0]] <= gray_frame[mesh[idxs[1]][1], mesh[idxs[1]][0]])

```

```

pupils_darker.append(gray_frame[mesh[idxs[0]][1],mesh[idxs[0]][0]]<= gray_frame[mesh[idxs[2]][1],
mesh[idxs[2]][0]])
return sum(pupils_darker)

```

```

def setupModels():

```

```

    cap = cv2.VideoCapture(0, cv2.CAP_DSHOW)
    cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 720)
    cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1280)
    mp_face_mesh = mediapipe.solutions.face_mesh
    device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')
    model = HybridNet().to(device)
    model.load_state_dict(torch.load('./best_hybrid.pt', map_location=device))
    model.eval()
    return model, mp_face_mesh, cap

```

```

def predictGazeInLoop(model, mesh, cap):

```

```

    ret, frame = cap.read()
    mp_face_mesh = mediapipe.solutions.face_mesh
    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    with mesh.FaceMesh(
        max_num_faces=1,
        refine_landmarks=True,
        min_detection_confidence=0.65,
        min_tracking_confidence=0.65) as face_mesh:
        res = face_mesh.process(frame)
        if not res.multi_face_landmarks:
            return -1,-1,-1
        else:
            gray_frame = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)
            h, w = frame.shape[:2]
            norm_meshpoints = np.array([np.array([p.x, p.y]) for p in res.multi_face_landmarks[0].landmark])
            meshpoints = np.array([np.array([p.x*w, p.y*h]).astype(int) for p in res.multi_face_
landmarks[0].landmark])
            is_darker = get_is_darker(gray_frame, meshpoints, LEFT_EYE_3_POINTS) + get_is_darker(gray
_frame, meshpoints, RIGHT_EYE_3_POINTS)
            ear = get_ear(meshpoints, LEFT_EYE_6_POINTS) + get_ear(meshpoints, RIGHT_EYE
_6_POINTS)
            if is_darker < 3 or ear / 2 < 0.2:
                return -2,-2,-2
            else:
                left_eye = frame[int(meshpoints[L_H_PUPIL][1]-abs(meshpoints[L_H_LEFT][0]-
meshpoints[L_H_RIGHT][0])/2):
int(meshpoints[L_H_PUPIL][1]+abs(meshpoints[L_H_LEFT][0]-meshpoints[L_H_RIGHT][0])/2),
                meshpoints[L_H_LEFT][0]:meshpoints[L_H_RIGHT][0]]
                right_eye = frame[int(meshpoints[R_H_PUPIL][1]-abs(meshpoints[R_H_RIGHT][0]-
meshpoints[R_H_LEFT][0])/2):
int(meshpoints[R_H_PUPIL][1]+abs(meshpoints[R_H_LEFT][0]-meshpoints[R_H_RIGHT][0])/2),
                meshpoints[R_H_LEFT][0]:meshpoints[R_H_RIGHT][0]]
                cv2.imwrite('re.jpg',right_eye)
                cv2.imwrite('le.jpg', left_eye)

```



```

        left_eye = transform(cv2.resize(cv2.flip(np.array(cv2.cvtColor(left_eye, cv2.COLOR_
BGR2RGB) / 255, dtype='float32'),1), (25,25), interpolation=cv2.INTER_AREA))
        right_eye = transform(cv2.resize(np.array(cv2.cvtColor(right_eye, cv2.COLOR_BGR2RGB) /
255, dtype='float32'), (25,25), interpolation=cv2.INTER_AREA))
        coords = torch.cat([torch.FloatTensor([norm_meshpoints[L_H_RIGHT][0],norm_mesh-
points[L_H_RIGHT][1]]),
        torch.FloatTensor([norm_meshpoints[R_H_LEFT][0],norm_mesh-
points[R_H_LEFT][1]]),
        torch.FloatTensor([norm_meshpoints[L_H_LEFT][0],norm_mesh-
points[L_H_LEFT][1]]),
        torch.FloatTensor([norm_meshpoints[R_H_RIGHT][0],norm_mesh-
points[R_H_RIGHT][1]])], dim=-1)
        device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')
        left_eye = left_eye.to(device)
        right_eye = right_eye.to(device)
        coords = coords.to(device).unsqueeze(0)
        with torch.no_grad():
            outputs = model([left_eye, right_eye], coords).squeeze()
            segment = int(torch.nn.functional.relu(outputs[1])*900)//300*4 + int(torch.nn.functional.
relu(outputs[0])*1600)//400
            return min(11, segment), float(outputs[0]), float(outputs[1])

if __name__ == "__main__":
    model, mesh, cap = setupModels()
    k = 0
    while True:
        print(predictGazeInLoop(model,mesh,cap), k, flush=True)
        k+=1

```

Файл test_model.pro

```

QT += quick \
    widgets

CONFIG += c++11

# You can make your code fail to compile if it uses deprecated APIs.
# In order to do so, uncomment the following line.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000

SOURCES += \
    camerahandler.cpp \
    main.cpp \
    mainwindow.cpp \
    messagewidget.cpp

RESOURCES += \
    qrc.qrc

# Additional import path used to resolve QML modules in Qt Creator's code model

```

```
QML_IMPORT_PATH =
```

```
# Additional import path used to resolve QML modules just for Qt Quick Designer  
QML_DESIGNER_IMPORT_PATH =
```

```
# Default rules for deployment.
```

```
qnx: target.path = /tmp/${TARGET}/bin  
else: unix:!android: target.path = /opt/${TARGET}/bin  
!isEmpty(target.path): INSTALLS += target
```

```
HEADERS += \  
    camerahandler.h \  
    mainwindow.h \  
    messagewidget.h
```

```
INCLUDEPATH += D:\opencv\opencv\build\include
```

```
LIBS += D:\opencv\opencv-build\bin\libopencv_core454.dll  
LIBS += D:\opencv\opencv-build\bin\libopencv_highgui454.dll  
LIBS += D:\opencv\opencv-build\bin\libopencv_imgcodecs454.dll  
LIBS += D:\opencv\opencv-build\bin\libopencv_imgproc454.dll  
LIBS += D:\opencv\opencv-build\bin\libopencv_features2d454.dll  
LIBS += D:\opencv\opencv-build\bin\libopencv_calib3d454.dll  
LIBS += D:\opencv\opencv-build\bin\libopencv_objdetect454.dll  
LIBS += D:\opencv\opencv-build\bin\libopencv_videoio454.dll  
LIBS += D:\opencv\opencv-build\bin\libopencv_dnn454.dll
```