Департамент образования и науки города Москвы

Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет

«Московский институт электронной техники»

**Конкурс проектов «Высокие цели» 2022/2023**

**Исследовательская работа**

на тему:  
 «Разработка программного модуля для контроля за вниманием водителя на основе технологий компьютерного зрения и методов машинного обучения»

Автор работы:

Алексеев Александр Сергеевич

Студент 4 курса НИУ МИЭТ группы ПИН-43

Специальность 09.03.04 Программные технологии

распределенной обработки информации

Москва, 2023

Разработка программного модуля для контроля за вниманием водителя

ПМ КСВ

[СОДЕРЖАНИЕ 2](#_Toc132279886)

[ПЕРЕЧЕНЬ СОКРАЩЕНИЙ 3](#_Toc132279887)

[ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ 4](#_Toc132279888)

[ВВЕДЕНИЕ 5](#_Toc132279889)

[ИССЛЕДОВАТЕЛЬСКИЙ РАЗДЕЛ 6](#_Toc132279890)

[1 Актуальность выбранной темы 6](#_Toc132279891)

[2 Анализ потребностей потенциальных пользователей 7](#_Toc132279892)

[3 Обзор существующих аналогов 7](#_Toc132279893)

[4 Исследование предметной области 9](#_Toc132279894)

[4.1 Анализ существующих алгоритмов обнаружения лица 9](#_Toc132279895)

[4.2 Анализ существующих алгоритмов определения области глаз 18](#_Toc132279896)

[4.3 Анализ существующих алгоритмов определения состояния глаз 23](#_Toc132279897)

[5 Цель и задачи работы 29](#_Toc132279903)

[6 Постановка задачи работы 30](#_Toc132279904)

[7 Выводы по разделу 31](#_Toc132279905)

[КОНСТРУКТОРСКИЙ РАЗДЕЛ 32](#_Toc132279906)

[1 Структура входных и выходных данных 32](#_Toc132279907)

[2 Выбор платформы и инструментальных средств 34](#_Toc132279908)

[2.1 Выбор языка программирования 34](#_Toc132279909)

[2.2 Выбор среды разработки 37](#_Toc132279912)

[2.3 Выбор фреймворка для машинного обучения 39](#_Toc132279913)

[3 Программная архитектура и алгоритм работы 40](#_Toc132279914)

[3.1 Общий алгоритм работы 41](#_Toc132279915)

[3.2 Детектирование лиц 42](#_Toc132279916)

[3.3 Алгоритм трекинга 44](#_Toc132279917)

[3.4 Алгоритм определения области глаз 47](#_Toc132279918)

[3.5 Алгоритм определения состояния глаз 50](#_Toc132279919)

[3.6 Метрики в задачах классификации 51](#_Toc132279920)

[4 Выводы по разделу 53](#_Toc132279921)

[ИСПЫТАТЕЛЬНЫЙ РАЗДЕЛ 54](#_Toc132279922)

[1 Технологии программирования 54](#_Toc132279923)

[1.1 Достоинства и недостатки языка программирования Python 54](#_Toc132279924)

[1.2 Основные преимущества использования фреймворка Keras 56](#_Toc132279925)

[2 Используемые сторонние библиотеки 56](#_Toc132279926)

[2.1 Библиотека Numpy 56](#_Toc132279927)

[2.2 Библиотека OpenCV 57](#_Toc132279928)

[2.3 Библиотека Dlib 59](#_Toc132279929)

[3 Методы отладки 60](#_Toc132279930)

[4 Методы тестирования нейросетевых алгоритмов 61](#_Toc132279931)

[4.1 Подготовка репрезентативной выборки 61](#_Toc132279932)

[4.2 Результаты тестирования алгоритма определения состояния глаз 64](#_Toc132279937)

[5 Тестирование точности алгоритма трекинга 66](#_Toc132279938)

[6 Тестирование скорости работы алгоритма детектирования лиц 67](#_Toc132279939)

[7 Выводы по разделу 67](#_Toc132279940)

[ОСНОВНЫЕ РЕЗУЛЬТАТЫ И ВЫВОДЫ 69](#_Toc132279941)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 70](#_Toc132279942)

[ПРИЛОЖЕНИЕ 1. Программный код 75](#_Toc132279943)

# 

# ПЕРЕЧЕНЬ СОКРАЩЕНИЙ

|  |  |  |
| --- | --- | --- |
| ИИ | — | Искусственный интеллект |
| ИНС | — | Искусственная нейронная сеть |
| ООП | — | Объектно-ориентированное программирование |
| ПМ КСВ | — | Программный модуль для контроля за вниманием водителя |
| ПО | — | Программное обеспечение |
| HOG | — | Histogram of Oriented Gradients |
| SVM | — | Support Vector Machine – Метод опорных векторов |
| ML | — | Machine Learning – Машинное обучение |
| MTCNN | — | Multi-task Cascaded Convolutional Networks |
|  |  |  |

# ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

Дерево решений – это метод представления решающих правил в иерархической структуре, состоящей из элементов двух типов: узлов (node) и листьев (leaf).

Интерпретатор – программа (разновидность транслятора), выполняющая интерпретацию.

Интерпретация — построчный анализ, обработка и выполнение исходного кода программы или запроса (в отличие от компиляции, где весь текст программы, перед запуском, анализируется и транслируется в машинный или байт-код, без её выполнения)

Компилятор – программа или техническое средство, выполняющие компиляцию.

Компиляция – трансляция программы с языка высокого уровня в форму, близкую к программе, на машинном языке.

Тестирование ПО – процесс исследования, испытания программного продукта, цель которого состоит в проверке соответствия между реальным поведением программы и ее ожидаемым поведением на конечном наборе тестов, выбранных определенным образом из множества, близкого к бесконечному.

Фреймворк – ПО, облегчающее разработку и сборку программного проекта.

RTSP – это real time streaming protocol, то есть потоковый протокол реального времени. По сути, это узкоспециализированный протокол, использующийся для удалённого управления потока данных с сервера и работы с мультимедийными данными.

USB – последовательный интерфейс для подключения периферийных устройств к вычислительной технике.

# ВВЕДЕНИЕ

В современном мире значительно повысился интерес к проблеме безопасности. Одни из основных проблем, существующих сегодня, это засыпания водителей за рулем автотранспортного средства, отсутствие статистики и набора данных, способных помочь в решении данной проблемы, а также большие вычислительные затраты на видеоаналитику, требующие дорогих аппаратных комплексов.

Благодаря технологиям машинного обучения проблема обеспечения безопасности дорожного движения получила универсальное решение - разработка программного модуля для контроля за вниманием водителя автотранспортного средства. Однако, предлагаемые на рынке программно-аппаратные средства обладают общим недостатками: отсутствие функции логирования событий, высокая себестоимость аппаратных комплексов, а также отсутствие возможности настройки и калибровки под конкретную камеру.

Система контроля усталости водителя призвана обнаружить наступление усталости водителя и предупредить сон за рулем. Система предлагает сделать перерыв для отдыха путем предупреждающего звукового сигнала или сигнала на панели приборов («чашка кофе»). Определение наступления усталости водителя осуществляется разными способами – оценкой действий водителя по управлению автомобилем, контролем характера движения автомобиля, наблюдением за лицом водителя с помощью видеокамеры [3].

ПМ КСВ удовлетворяет основным потребительским потребностям в области контроля внимания водителя автотранспортного средства применением технологий машинного обучения и компьютерного зрения, а также поддерживает функцию фиксации событий и логирования.

Контроль за вниманием необходим не только участникам дорожного движения. Данная система может быть актуальна в областях, где человек выполняет функции оператора. Например, данная система может быть использована в аэропортах, для контроля за вниманием диспетчера. Цена невнимательности и усталости в данном случае слишком высока.

# ИССЛЕДОВАТЕЛЬСКИЙ РАЗДЕЛ

## Актуальность выбранной темы

В современном мире значительно повысился интерес к проблеме безопасности. Системы искусственного интеллекта позволяют контролировать при помощи камер видеонаблюдения автомобильный трафик. Согласно данным с официального сайта Госавтоинспекции большую часть ДТП на начало 2023 г. составляют наезды водителей автотранспортных средств на пешеходов. Одна из причин таких происшествий связана с засыпанием водителя за рулем автомобиля. В связи с этими событиями, неудивительно, что программные решение добрались и до контроля за вниманием водителя автотранспортного средства.

Система контроля усталости водителя призвана обнаружить наступление усталости водителя и предупредить сон за рулем. Система предлагает сделать перерыв для отдыха путем предупреждающего звукового сигнала или сигнала на панели приборов («чашка кофе»). Данная система контроля усталости водителя может быть реализована на автомобилях Mercedes-Benz, Volvo, Lexus. Определение наступления усталости водителя осуществляется разными способами – оценкой действий водителя по управлению автомобилем, контролем характера движения автомобиля, наблюдением за лицом водителя с помощью видеокамеры [3].

В 2022 году ГКУ «Дирекция Транспортных Закупок» организовало тендер по оказанию услуг по оперативному мониторингу функционального состояния машинистов [1]. В 2023 году ГУП «Мосгортранс» организовал тендер по оказанию услуг по оперативному мониторингу функционального состояния водителей [2]. Такое пристальное внимание со стороны государственных организаций к функциональному мониторингу состояния водителя говорит об актуальности и востребованности решения проблемы контроля за вниманием водителя.

Контроль за вниманием необходим не только участникам дорожного движения. Данная система может быть актуальна в областях, где человек выполняет функции оператора. Например, данная система может быть использована в аэропортах, для контроля за вниманием диспетчера. Цена невнимательности и усталости в данном случае слишком высока.

## Анализ потребностей потенциальных пользователей

Потенциальными потребителями ПМ КСВ могут быть организации и компании, оказывающие услуги пассажирских или грузовых перевозок. Например, ГУП «Московский метрополитен», ГУП «Мосгортранс» и ГКУ «Дирекция Транспортных Закупок».

Информационные потребности пользователей представлены следующими требованиями:

* высокая вероятность правильного определения состояния глаз;
* простота эксплуатации;
* возможность внедрения дополнительной аналитики;
* низкие ценовые характеристики программно-аппаратного комплекса.

## Обзор существующих аналогов

В настоящее время существует ряд программно-аппаратных средств, позволяющих контролировать состояние водителя во время управления автотранспортом. Лидирующие позиции в этом направлении занимают компании:

* Яндекс. Комплекс состоит из одноплатного компьютера NanoPI M4 на процессоре Rockchip RK3399 и USB-камеры с инфракрасной подсветкой. Особенностью данного решения является то, что для анализа используется изображения 640x480 пикселей, SDK от VisionLabs позволяет получить 30 FPS для операций нахождения лица и получения ключевых точек. Для поиска лица используется технология на основе нейронных сетей. Реализован ряд алгоритмов, включая алгоритмы поиска лица, распознавания лица, получения антропометрических точек, определения положения головы, определения состояния глаз. На основе технологии IR Liveness реализована функция, определяющая живой человек перед камерой или это фотография [4].
* Dunobil Insomnia. Программно-аппаратный комплекс, состоящий из одного цельного блока, в который входит вычислитель, камера и датчики. Реализованы алгоритмы поиска лица в кадре, определения поворота головы, определения усталости. Оповещение водителя об опасной ситуации происходит при помощи звукового сигнала [5].
* Movon. Система помощи водителю представляет собой вычислитель и камеру в одном корпусе. Данное устройство умеет определять отвлечение внимания, засыпания, зевания, использование телефона, курение. Предотвращение несчастного случая происходит путем звукового оповещения водителя. Реализована функция распознавания лиц для получения уникального ID человека [6,7].
* ОКО Системс. Техническое устройство по исполнению напоминает Dunobil Insomnia. Система ОКО предназначена для предотвращения несчастных случаев и аварийных ситуаций, вызванных человеческими факторами. Осуществляется мониторинг движения зрачков, глаз, головы оператора. Отслеживает утрату концентрации внимания. Оповещает водителя предупредительным, или, при необходимости, аварийным звуковым сигналом [8].

Таблица 1.1 – Сравнительная таблица текущего технического решения и существующих программно-аппаратных аналогов

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Аналог  Критерий | Яндекс | Dunobil Insomnia | Movon | ОКО Системс | ПМ КСВ |
| Коммерческий продукт | + | + | + | + | - |
| Открытый исходный код | - | - | - | - | + |
| Реализован алгоритм детектирования лица | + | + | + | + | + |
| Реализован алгоритм определения состояния глаз | + | - | + | + | + |
| Поддержка USB и RTSP протоколов | - | - | - | - | + |

Недостатком существующих систем является отсутствие поддержки сразу двух протоколов: USB и RSTP. Стоит отметить ещё тот факт, что существующие технические решения настроены и откалиброваны под конкретное изображение. Если используется камера с инфракрасной подсветкой, как, например, у Яндекса, то изменения источника входного изображения приведут к неверной работе комплекса или к ошибочным прогнозам при классификации.

Проведенный анализ показал (таблица 1.1), что существующие программно-аппаратные комплексы в меньшей степени подходят под решение поставленных задач и достижение текущих целей, так как их использование лишает гибкости при выборе источника входного изображения.

## Исследование предметной области

* 1. Анализ существующих алгоритмов обнаружения лица

Обнаружение лица – это первый и основной этап при решении задачи контроля за вниманием человека. Тем не менее, обнаружения лица на конкретном изображении - это довольно сложная задача ввиду непостоянства масштаба, расположения и ориентации (вертикально, повернуто). Выражение лица, окклюзии и условия освещенности также меняют общий вид лица.

В своей исследовательской работе «Detecting Faces in Images: A Survey» Янг, Кригмэн и Ахаджа произвели обзор и классификацию существующих методов обнаружения лица. Согласно их классификации, все методы могут быть разделены на четыре категории, причем алгоритмы обнаружения лица могут одновременно принадлежать к двум или более категориям [9].

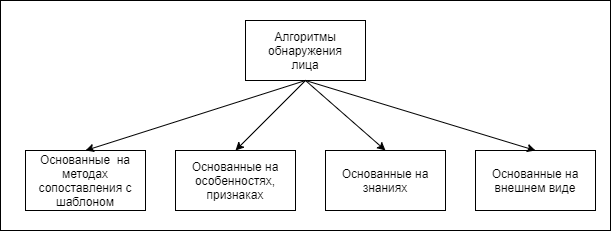


Рисунок 1.1 - Классификация алгоритмов обнаружения лица

1. Методы, основанные на сопоставлении с шаблоном

При использовании методов сопоставления с шаблоном перед разработчиком стоит задача выбора подходящего шаблона, который в действительности является «общим» лицом, имеющим все черты лица, не будучи слишком конкретным. Чаще всего в качестве шаблона используется изображение центральной части лица, содержащие его основные элементы: глаза, нос, брови и рот (Рис 1.2). Использование в качестве шаблона только центральной части лица обусловлено тем, что это позволяет предотвратить влияние наиболее непостоянных частей головы человека, содержащих прическу и (или) бороду. Обнаружение лица с помощью шаблона представляет собой проверку каждой из областей изображения на соответствие заданному шаблону.

Шаблон масштабируется согласно высоте и ширине проверяемой области. Отнесение части изображения к области, содержащей лицо, производится путем расчета корреляции между исследуемой частью изображения и шаблоном.

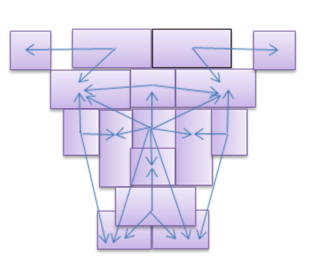


Рисунок 1.2 – Шаблон 14x16 пикселей для локализации лица на основе метода Синха. Шаблон состоит из 16 областей (серые прямоугольники) и 23 отношений (показано стрелками)

К достоинствам распознавания с помощью шаблонов можно отнести относительную простоту реализации и неплохие результаты на изображениях с не очень сложным задним фоном. А главным недостатком является необходимость калибровки шаблона вблизи с изображением лица. Большая трудоёмкость вычисления шаблонов для различных ракурсов и поворотов лица ставят под вопрос целесообразность их использования [10].

1. Методы, основанные на особенностях, признаках

Объекты отличаются своими уникальными особенностями. Глаза, уши, брови, нос, губы, цвет кожи и так далее – это признаки, которые могут быть использованы для обнаружения лица. Вместо того чтобы использовать один шаблон для представления всего лица, данные методы используют отдельные шаблоны для поиска каждого отдельного признака лица. Алгоритмы, относящиеся к данной категории, пытаются выявить закономерности и свойства изображения лица, найти инвариантные признаки лица, не зависящие от положения головы и угла наклона. Область изображения определяется как лицо, если идентифицируется подходящая комбинация черт лица. Эти методы позволяют находить не только фронтальные лица, но и лица с незначительными поворотами головы. Для обнаружения внутренних признаков лицо на изображении должно быть достаточно большого размера. Поэтому данные алгоритмы не пригодные для лиц с разрешением 50х50 пикселей.

Пример может служить алгоритм Виолы-Джонса, которой является одним из классических методов обнаружения лица.  Все человеческие лица имеют некоторые универсальные свойства человеческого лица, например, область глаз темнее соседних пикселей, а область носа ярче, чем область глаз. Поэтому в качестве признаков используются примитивы Хаара (рис 1.3). Преимущество использования признаков Хаара является наибольшая, по сравнению с остальными признаками, скорость. При использовании интегрального представления изображения, признаки Хаара могут вычисляться за постоянное время.

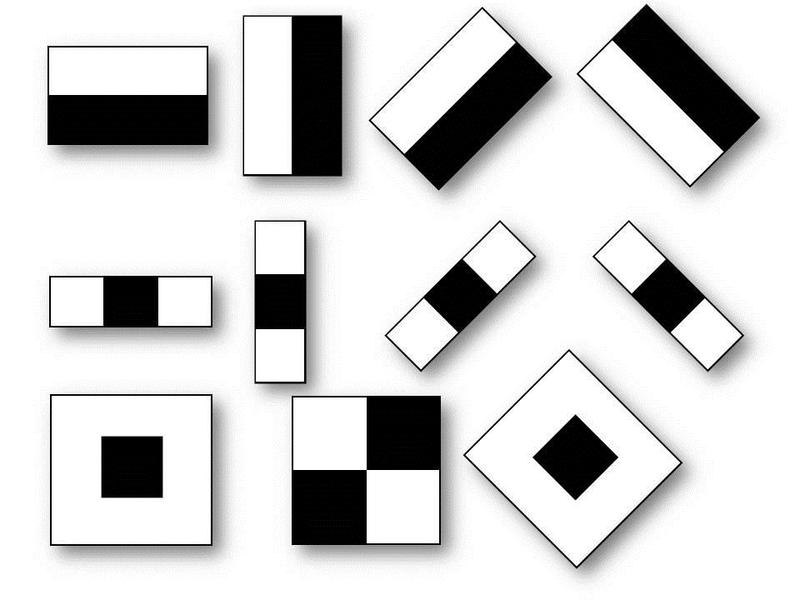


Рисунок 1.3 – Примитивы Хаара

Недостатком подобных алгоритмов является сильная зависимость от качества изображения. Наличие искажений в изображении из-за освещения, шума или окклюзии может привести к неконтролируемым результатам работы.

1. Методы, основанные на знаниях

Эти методы предполагают создание алгоритма, который должен реализовывать набор правил, которым должен отвечать фрагмент изображения, для того, чтобы он мог считаться человеческим лицом. Правила – это попытка формализовать эмпирические знания о том, как выглядит лицо на изображениях, и чем руководствуется человек, когда принимает решение о том, видит ли он перед собой лицо или нет.

Приведем для примера несколько простых правил:

* центральная часть лица имеет однородную яркость и цвет;
* разница в яркости между центральной частью и верхней частью лица значительна;
* лицо содержит в себе два симметрично расположенных глаза, нос и рот, резко отличающиеся по яркости относительно остальной части лица.

В качестве примера можно рассмотреть алгоритм Янга и Хоанга. В своем методе они используют три уровня правил. Правила самого высокого уровня предназначены для нахождения всех потенциальных областей лица методом скользящего окна. Правила на верхнем уровне являются общим описанием того, как выглядит лицо, в то время как правила на более низких уровнях полагаются на признаки черт лица. Иерархия изображений с несколькими разрешениями создается путем усреднения и субдискретизации (рис 1.4).



Рисунок 1.4 – Исходное изображение и соответствующие ему изображение с низким разрешением

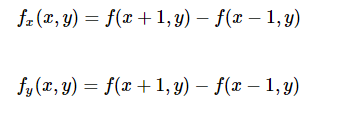
Метод Янга и Хоанга не приводит к высокой частоте обнаружения лица, но во многих более поздних работах на эту тему используется идея изображений с несколькими разрешениями и поиск на основе правил для обнаружения изображений лиц во фронтальной проекции.

1. Методы, основанные на внешнем виде

В данной категории алгоритмов изображению (или его области) ставится в соответствие вычисленный каким-то образом вектор признаков, который в дальнейшем используется для бинарной классификации – лицо или не лицо. Среди этих методов выделяются хорошо себя зарекомендовавшие нейронные сети и более классический подход на основе гистограммы ориентированных градиентов.

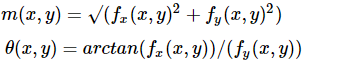
Прежде чем приступить к рассмотрению алгоритма обнаружения лица на основе гистограммы ориентированных градиентов, следует рассмотреть ряд важных понятий и определений. Дескриптор признаков – это представление изображения или части изображения, которое упрощает его, извлекая полезную и отбрасывая постороннюю информацию.Гистограмма направленных (ориентированных) градиентов (Histogram of Oriented Gradients, HOG) — дескрипторы особых точек, которые используются в компьютерном зрении и обработке изображений с целью распознавания объектов [12].

Как правило, дескриптор преобразует входное изображение в вектор или массив длины n. В HOG дескрипторе в качестве признаков используется распределения (гистограммы) направленных (ориентированных) градиентов. Польза градиента изображения в том, что величина градиента велика вокруг ребер и углов (область резких изменений интенсивности), а также известно, что ребра и края содержат гораздо больше информации о форме объекта, чем плоские области. Для каждого пикселя вычисляют горизонтальный и вертикальный градиент (1) (2), на основе этих данных происходит расчёт величины и направления градиента (3) (4).



(1)

(2)

(3)

(4)

Для цветных изображений вычисление градиента происходит для каждого канала по отдельности. Величина градиента в таком случае равна максимальной величине градиента трех каналов, а угол соответствует углу максимального градиента. Обычно используется не значение градиента в точке, а значение градиента в пределах блока. Изображение делится на блоки 8x8 пикселей, и гистограмма рассчитывается для каждого блока по отдельности. Благодаря данной методике, представление изображения становится более компактным, а вычисление гистограммы по блоку делает это представление устойчивым к шуму и изменениям освещения. Гистограмму ориентированных градиентов можно визуализировать (рис. 1.7).

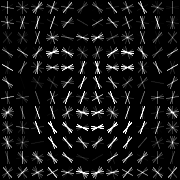


Рисунок 1.7 – Гистограмма ориентированных градиентов лица

Полученные описанным образом дескрипторы можно использовать для обучения классификатора на основе метода опорных векторов, который гарантирует оптимальное разделение дескрипторов на 2 класса – лица и не лица.

Сильными сторонами данного подхода является малое число ложных срабатываний, по сравнению с каскадными классификаторами, при этом сохраняя скорость работы на уровне каскадных классификаторов. Слабыми сторонами – довольно высокие требования к вычислительным мощностям для обеспечения работы в режиме реального времени и невозможность работы данного алгоритма при углах поворота выше 30 градусов.

Искусственная нейронная сеть (ИНС) – математическая модель, а также ее программная или аппаратная реализация, построенная по принципу организации и функционирования биологических нейронных сетей. ИНС – система соединённых и взаимодействующих между собой искусственных нейронов.

Искусственный нейрон – узел искусственной нейронной сети, являющийся упрощенной моделью естественного нейрона. С математической точки зрения, искусственный нейрон обычно представляют как некоторую нелинейную функцию от единственного аргумента – линейной всех входных сигналов. Данную функцию называют [функцией активации](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_%D0%B0%D0%BA%D1%82%D0%B8%D0%B2%D0%B0%D1%86%D0%B8%D0%B8) или функцией срабатывания, передаточной функцией.

Обучение ИНС относится, как правило, к обучение с учителем. Обучение с учителем – это из методов машинного обучения, в ходе которого система принудительно обучается с помощью примеров «стимул-реакция». В данном случае обязательно наличие обучающего набора, который содержит примеры с истинными значениями: тегами, классами, показателями. ИНС активно применяются для решения задач компьютерного зрения, включая задачу распознавания объектов на изображении. В процессе обучения ИНС автоматически подбирает весовые коэффициенты для достижения максимальной точности.

Нейронные сети позволяют решать огромный спектр задач от классификации изображений до обработки естественного языка. При решении задач каждого типа используются свои архитектуры нейронных сетей. Для обнаружения объектов чаще всего используются 3 архитектуры: R-CNN [42], YOLO [43], SSD [41]. Эти архитектуры направлены на решение задачи обнаружения объектов в общем и целом, они не разрабатывались специально для обнаружения лиц. Самой быстро из данных архитектур является SSD.

Термин SSD (Single Shot Detector) используется для описания архитектур, в которых используется одна сверточная нейронная сеть для непосредственного предсказания расположения областей и их классов, без применения второго этапа классификации. В этом методе на выходе нейронной сети формируются несколько тысяч прогнозов для возможных регионов расположения объектов разной формы на разных масштабах, затем с помощью метода подавления немаксимумов происходит выбор нескольких наиболее вероятных областей. Такая единая структура, одновременно с учетом различных масштабов изображения обеспечила методу SSD наиболее высокие показатели по скорости обнаружения объектов по сравнению с R-CNN и YOLO.

Специально для решения задачи обнаружения лиц группой ученых была разработана глубокая каскадная многозадачная архитектура, которая использует внутреннюю корреляцию между блоками для повышения производительности каскада. Эта архитектура носит название MTCNN (Multi-task Cascaded Convolutional Networks) [40]. В отличие от многих алгоритмов для обнаружения объектов, MTCNN работает не с фиксированным размером входного изображения, а с динамическим. Для этого входное изображение предобрабатывается, превращаясь в пирамиду изображений. На первом этапе работы MTCNN порождает области-кандидаты, которые могут являться лицами, при помощи метода скользящего окна и неглубокой ИНС. Затем происходит уточнение ограничивающих прямоугольников, отбрасывая большое количество не лиц при помощи более сложной ИНС. На последнем шаге используется самая мощная ИНС, чтобы оставить только по одной ограничивающей области для каждого лица. Общая схема работы MTCNN представлена на рисунке 1.8.

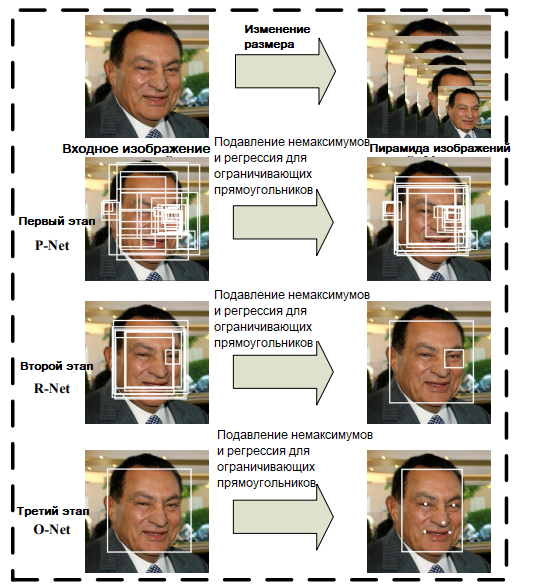


Рисунок 1.8 – Схема работы MTCNN

В результате тестирования методов MTCNN, HOG, SSD и метода Виолы-Джонса была получена таблица 1.2. В качестве критериев оценки были выбраны следующие параметры: возможность в режиме реального времени (более 12 кадров в секунду), точность, наличие ложных срабатывай, возможность работы при углах поворота головы более 30 градусов.

Таблица 1.2 – Сравнительная таблица алгоритмов детектирования лиц

| Параметр  Метод | Работа в режиме реального времени (более 12 кадров в секунду) | Точность | Ложные срабатывания | Определение лица при углах поворота головы более 30 градусов |
| --- | --- | --- | --- | --- |
| SSD | Нет | Высокая | Нет | Да |
| HOG | Да | Средняя | Нет | Не во всех случаях |
| MTCNN | Да | Высокая | Нет | Да |
| Виолы-Джонса | Да | Средняя | В большом количестве | Нет |

Как можно видеть из таблицы, MTCNN - это один из высокоточных детекторов лиц, работающий в реальном времени, обладающий высокой точностью и возможностью работать при углах поворота более 30 градусов.

### Анализ существующих алгоритмов определения области глаз

Определение области глаз на фотографии – одно из основных направлений работ в приложениях по обработке изображений. Например, распознавание лиц, верификация пользователей, оценка положения зрачка и человеко-машинный интерфейс. Для обнаружения глаз были предложение различные алгоритмы: на основе освещения, цвета, геометрической информации и на основе детектора ребер. Существует три основных подхода определения области глаз: методы, основанные на форме, методы, основанные на внешнем виде, и гибридные методы. Методы, основанные на форме, используют локальные точечные признаки глаз и лица, либо используют их контуры. Соответствующими признаками могут быть края, углы глаз или точки, выбранные на основе конкретных откликов фильтра. Методы, основанные на внешнем виде, используют цветовое распределение или отклик фильтров области глаз. Эти методы требуют большого количества обучающих данных, представляющих глаза при различных условиях освещения и положениях лица. Наконец, гибридные методы объединяют в себе два или более подходов для использования их преимуществ.

1. методы, основанные на форме;

Открытый глаз хорошо описывается своей формой, которая включает в себя контуры радужки и зрачка, а также внешнюю форму глаза (веки). Классификация подходов, основанных на форме, зависит от того, является ли предварительная модель только эллиптической или имеет более сложную структуру и форму. Модели формы обычно состоят из двух компонентов: геометрической модели глаза и меры подобия.

Многие приложения нуждаются только в обнаружении и отслеживании радужной оболочки или зрачка. Под определенными углами обзора радужная оболочка и зрачок выглядят эллиптически и, следовательно, могут быть смоделированы с помощью пяти параметров формы. Простые эллиптические модели состоят из методов, основанных на голосовании, и методов подбора моделей. Методы голосования выбирают объекты, которые поддерживают заданную гипотезу, через процесс голосования или накопления. Методы подбора моделей служат для приспособления выбранных объектов к модели.

Ким и Рамакришна в своей работе используют пороговые значения интенсивности изображения для оценки центра зрачка [15]. Методы обнаружения ребер используются для извлечения лимба или границ зрачка. Некоторые области изображения могут иметь профиль интенсивности схожий с областями радужной оболочки и зрачка. Использование пороговых значений допустимо только в ограниченных условиях.

Котари и Митчелл предлагают схему голосования, которая использует пространственную и временную информацию для определения местоположения глаз [14]. Они применяют градиентное поле, используя знание о том, что градиент вдоль границы радужки направлен наружу от центра радужки.

Важным свойством этих методов является их общая способность обрабатывать изменения формы, масштаба и угла вращения.

1. методы, основанные на внешнем виде;

Методы, основанные на внешнем виде, также известны как шаблоны изображений или целостные методы. Данный класс алгоритмов обнаруживает и отслеживает глаза, основываясь на фотометрическом виде, характеризующемся цветовым распределением или откликом фильтров, глаза или его окрестности. Эти методы не зависят от объекта моделирования, то есть позволяют моделировать объекты отличные от глаз. Алгоритмы, основанные на внешнем виде, могут использовать шаблоны изображений, где сохраняется пространственная информация и интенсивность каждого пикселя, или основываться на целостном подходе, где распределение интенсивности характеризуется пренебрежением пространственной информации.

Методы с использованием шаблона имеют ряд характерных недостатков, связанных с масштабированием и изменением угла, которые особенно сильно проявляются в одношаблонных моделях. Изменение положения головы или движение глаз у одного и того же человека могут негативно сказываться на качестве работы алгоритма.

Целостный подход использует статистические методы для анализа распределения интенсивности внешнего вида объекта и получения эффективного представления, определенного в скрытом пространстве. Сходство между сохраненными прототипами и представлением входного изображения осуществляется в скрытом пространстве.

В качестве примера можно рассмотреть алгоритм Хуана и Векслера [16]. В их методе используется представление изображения глаз с использованием вейвлетов и классификатор радиальных базисных функций. В данном алгоритме обнаружение глаза рассматривают как биномиальную классификацию. Эксперименты показывают улучшенную производительность и точность вейвлет-классификатора по сравнению с классификаторами на основе интенсивности изображений. После обнаружения области глаза алгоритм предоставляет точную информацию о центре и радиусе глазных яблок, совмещая в себе информацию о контуре и области.

Основным недостатком методов, основанных на внешнем виде, требование к большому количеству и сильной вариативности тренировочных данных.

1. гибридные методы.

Эти методы направлены на объединение различных моделей глаз в единую систему с целью получения максимального преимущества и преодоления существующих недостатков.

Самым известным на данный момент гибридным алгоритмом является алгоритм Вахида Каземи и Жозефины Салливан, описанный в труде под названием «One Millisecond Face Alignment with an Ensemble of Regression Trees» [17]. В своей работе они показывают, как ансамбль деревьев регрессии может быть использован для оценки антропометрических точек лица непосредственно из разряженного подмножества интенсивностей пикселей. Данный алгоритм обладает высокой производительностью и качеством предсказаний, что позволяет использовать его в режиме реального времени. Обучение регрессионных функций производится по методу градиентного бустинга со среднеквадратичной функцией потерь.

Деревья решений – одни из наиболее эффективных инструментов интеллектуального анализа данных и предсказательной аналитики, которые позволяют решать задачи [классификации](https://wiki.loginom.ru/articles/classification.html) и [регрессии](https://wiki.loginom.ru/articles/regression.html). Дерево принятия решений – это дерево, в листьях которого стоят значения целевой функции, а в остальных узлах – условия перехода, определяющие по какому из ребер идти. Если для данного наблюдения условие – истина, то осуществляется переход по левому ребру, если же ложь – по правому.

Градиентый бустинг — метод анализа данных, представленный Джером Фридманом в 1999 году, и представляющий собой решение задачи регрессии методом построения комитета (ансамбля) “слабых” предсказывающих деревьев принятия решений [18].

Алгоритм Каземи и Салливан позволяет обучить детектор, который может предсказывать положение антропометрических точек лица. На основе этого детектора может быть получено множество, описывающее область глаз. Данный алгоритм был реализован в библиотеке с открытым исходным кодом dlib. Предварительно обученный детектор от dlib используется для оценки местоположения 68 точек (рисунок 1.9).

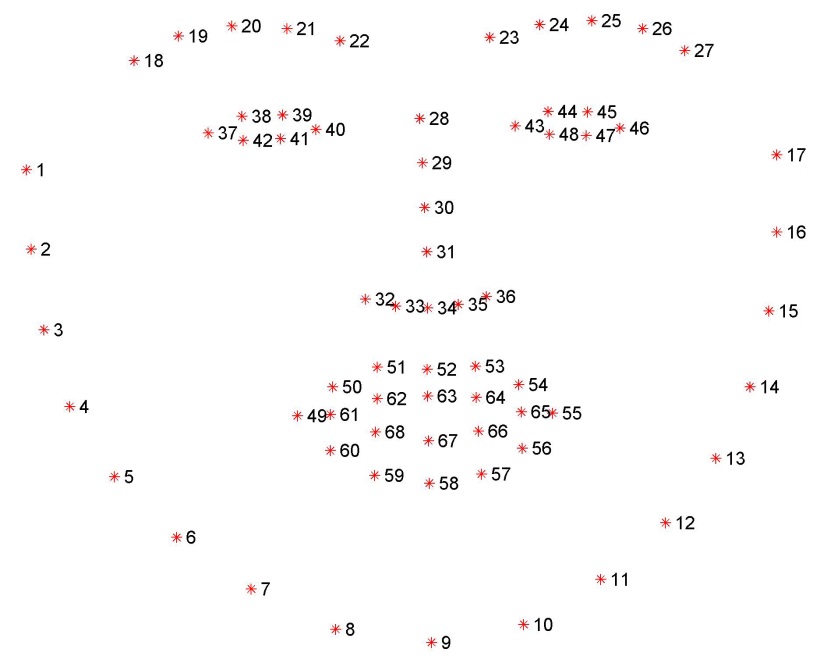


Рисунок 1.9 – Визуализация 68 антропометрических точек лица из набора данных iBUG 300-W

Эти аннотации являются частью 68-точечного [набора данных iBUG 300-W [19], на](https://ibug.doc.ic.ac.uk/resources/facial-point-annotations/) котором обучался предиктор dlib. Область правого глаза может быть описана множеством точек с 37 по 40, а область левого глаза – с 43 по 46 включительно.

Предиктор антропометрических точек от dlib обладает высокой скоростью и точностью работы, что позволяет использовать его для определения области глаз даже на одноплатных компьютерах.

Проведен сравнительный анализ алгоритмов определения области глаз, который представлен в таблице 1.3.

Таблица 1.3 – Сравнение алгоритмов определения области глаз

| Параметр  Алгоритм | Высокая точность определения области глаз | Требуется большое количество данных для обучения | Высокая скорость работы алгоритма |
| --- | --- | --- | --- |
| На основе формы  (алгоритм Ким и Рамакришна) | –/+ | – | + |
| На основе формы  (алгоритм Котари и Митчел) | –/+ | – | + |
| Вейвлеты и классификатор радиальных базисных функций  (алгоритм Хуана и Векслера) | +/– | + | +/– |
| Детектор  антропометрических точек  (алгорим Вахида Каземи и Жозефины Салливан) | + | – | + |

Из таблицы видно, что лучше всего для определения области глаз подходит алгоритм на основе антропометрических точек. Данный алгоритм обеспечивает работу в режиме реального времени. Кроме того, предиктор легко переобучается для определения только области глаз, что позволяет уменьшить размер модели и ещё сильнее увеличить производительность программного модуля. Алгоритм Каземи и Салливан реализован в библиотеке с открытым исходным кодом Dlib.

* 1. Анализ существующих алгоритмов определения состояния глаз

Системы по контролю за вниманием водителя используют обычно в качестве метрики усталости изменение продолжительности и частоты морганий. Моргание, согласно толковому словарю Ожегова, - это непроизвольное опускание и поднимание век, в среднем моргание длится от 30 до 150 миллисекунд. Моргание – это процесс, когда глаз из открытого состояния переходит в закрытое состояние и обратно. Определение состояние глаз – это бинарная классификация. Классификатор на основе каких-либо прямых или косвенных признаков формирует ответ на вопрос, в каком состоянии находятся глаза в данный момент – открытом или закрытом. Задача усложняется из-за условий освещения, качества съемки, наличия очков на глазах.

Основными методами для определения состояния глаз являются три алгоритма:

* алгоритм на основе вейвлетов Габора [23];
* алгоритм на основе антропометрических точек [25];
* алгоритм на основе извлечения контурных признаков [39].

Алгоритм на основе вейвлетов Габора. Согласно системе кодирования лицевых движений [24], состояния глаз можно поделить на 9 категорий: AU5, AU6, AU7, AU41, AU42, AU43, AU44, AU45 и AU46. Алгоритм на основе вейвлетов используется для классификации трех состояний: глаза открыты (AU41), глаза прищурены (AU42), глаза закрыты (AU43).



Рисунок 1.10 – Схема работы алгоритма на основе вейвлетов Габора

На первом этапе работы (рисунок 1.10) происходит определение положения головы и начальных позиций уголков глаз. Изображение, поданное на вход, должно содержать почти фронтальное лицо. На основе начального положения происходит трекинг глаз на последующих кадрах. Для каждого глаза вычисляется набор многомасштабных и многоориентационных коэффициентов Габора. Наконец, нормализованные коэффициенты Габора подаются на вход в нейросетевой детектор для классификации трех состояний глаза.

В таблице 1.4 приведены результаты работы алгоритма на тестовом наборе данных. Средняя точность распознавания составляет 83%. Авторы алгоритма считают, что в системах помощи водителя избыточно наличие трёх категорий состояния глаз. Прищуренный глаз и закрытый глаз могут быть объединены в один класс. Точность распознавания в таком случае возрастет до 93%.

Таблица 1.4 – Результаты тестирования алгоритма

|  |  |  |  |
| --- | --- | --- | --- |
| Состояние глаз | Открыты | Прищурены | Закрыты |
| Открыты | 52 | 4 | 0 |
| Прищурены | 4 | 28 | 8 |
| Закрыты | 0 | 3 | 13 |

Слабыми сторонами данного алгоритма является работа с почти фронтальными лицами, и невысокая точность работы классификатора состояний глаз.

Алгоритм на основе антропометрических точек. Данный алгоритм является простым, но эффективным методом определения морганий. На основе антропометрических точек выводится скалярная величина, отражающая степень открытости глаз. Используя последовательность оценок открытости глаз, производится определения морганий на основе SVM-классификатора.

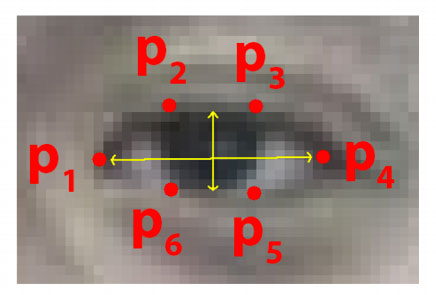
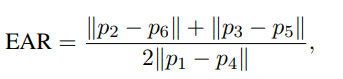


Рисунок 1.11 – Результат работы предиктора антропометрических точек

На каждом видеоизображении для определения лица используют детектор лиц Виолы-Джонса. На следующем шаге применяют предиктор антропометрических точек (рис. 1.11). Для каждого изображения, на котором было найдено лицо, происходит вычисление соотношения сторон глаза (eye aspect ratio, EAR) по формуле (3):



(3)

где p1, ..., p6 – это 2D антропометрические точки, изображенные на рисунке 1.11.

Значение коэффициента соотношения сторон глаза почти всегда является постоянной величиной, когда глаз открыт, и быстро стремится к нулю при закрытии глаза. Считается, что коэффициент соотношения сторон глаз имеет небольшую дисперсию между разными людьми, но полностью инвариантен к равномерному масштабированию изображение и вращению лица в плоскости. Низкое значение коэффициента может возникать не только во время морганий, а ещё в случаях, когда человек зевает, изменяет свое выражение лица или из-за коротких случайных флуктуаций координат антропометрических точек. Для устранения случаев ошибочной классификации предлагается использовать не одно значение коэффициента, а какую-то последовательность подряд идущих значений. Эмпирическим путем было установлено, что лучшая точность достигается при использовании 13-мерного признака.

Плюса данного алгоритма можно назвать скорость работы и интерпретируемость. Минусами же можно считать большую зависимость от качества работы предикторы антропометрических точек.

Алгоритм на основе извлечения контурных признаков. Общая схема алгоритма представлена на рисунке 1.12.



Рисунок 1.12 – Блок-схема алгоритма на основе извлечения контурных признаков

На первом шаге работы происходит получение входного изображение из какого-либо источника. Поскольку блики и тени оказывают большое влияние на цвет кожи, дополнительно используют алгоритм для восстановления цвета. Этот этап позволяет улучшить качество работы алгоритмы определения лиц. Для определения лиц используется алгоритм Виолы-Джонса, основанный на интегральном представлении изображения, каскадном классификаторе и алгоритме Adaboost. Область поиска глаз может быть уменьшена на основе свойств человеческого лица. По этой причине для дальнейшего анализа используется верхняя половина лица, полученного на предыдущем шаге. Для определения области глаз используется метод Оцу, преобразующий изображений в градациях серого в бинарное изображение. Пороговое значение подбирается с учетом того, что на изображении должны остаться две связанные области. При необходимости, пороговое значение уменьшают и повторяют анализ. Полученные области глаз заключают в прямоугольные рамки (рисунок 1.13).



Рисунок 1.13 – Результат работы алгоритма определения области глаз

Область склеры сегментируется методом кластеризации K-средних на основе разницы между красной и синей компонентой области склеры и кожи. На полученную область склеры накладывают две квадратичные кривые (рисунок 1.14), пересечение которых является крайними точками склеры.

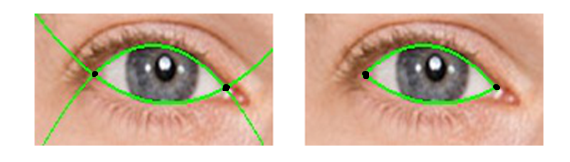


Рисунок 1.14 – Определение крайних точек при помощи квадратичных кривых

На основе области склеры и крайних точек строят минимальный ограничивающий прямоугольник. Полагая, что длина прямоугольника - это L, ширина (или высота) - это H, значение степени закрытости века можно определить по формуле (4).

(4)

Если значение M меньше или равно 0.15, то глаза находятся в закрытом состоянии, иначе – в открытом. Такое пороговое значение было получено экспериментальным образом, оно позволяет достигнуть точности классификации 98,67%.

Сильной стороной алгоритма является высокая точность классификации, но работа с цветными изображениями делает его абсолютно неприменимым при использовании камеры с инфракрасной подсветкой.

На основе проведенного исследования была получена таблица 1.5.

Таблица 1.5 – Результаты тестирования алгоритма

| Параметр  Алгоритм | Верная классификация при повернутом лице | Работа с изображениями в градациях серого | Высокая точность классификации |
| --- | --- | --- | --- |
| На основе вейвлетов Габора | –/+ | +/– | +/– |
| На основе антропометрических точек | –/+ | + | –/+ |
| На основе извлечения контурных признаков | + | – | +/– |

Согласно проведенному анализу, ни один из алгоритмов не подходит под выдвигаемые требования. Алгоритм на основе извлеченных контуров используют информацию о цвете для сегментации, что делает его абсолютно неприменимым при использовании изображений в градациях серого. Алгоритм на основе вейвлетов Габора хорошо работает для фронтальных изображений, но при сильном повороте лица точность классификации резко падает. Алгоритм на основе антропометрических точек инвариантен к цветовому пространству изображения. Но при уменьшении размеров лица качество работы предиктора антропометрических точек снижается, что влечет за собой большую погрешность при вычислении коэффициента соотношения сторон глаза.

Данные рассуждения подводят к разработке алгоритма, способного работать при разных углах поворота головы, инвариантного к цветовому пространству изображению и обеспечивающего высокую точность классификации.

## Цель и задачи работы

Целью работы является повышение безопасности дорожного движения путем создания программного модуля с применением технологий машинного обучения и компьютерного зрения.

Для достижения поставленный цели необходимо решить следующие задачи:

* исследование предметной области;
* сравнительный анализ существующих аналогов;
* выбор языка и среды программирования;
* разработка схемы данных;
* разработка схем алгоритмов;
* программная реализация;
* отладка и тестирование;
* разработка руководства программиста.

## Постановка задачи работы

Разрабатываемый программный модуль должен удовлетворять указанным в ТЗ функциональным требованиям, а именно:

* получение изображения из входного видеопотока (USB/RTSP);
* определение области глаз;
* трекинг лица с 95 % вероятностью;
* классификация состояний глаз с долей правильной классификации на уровне 95%;
* логирование в файл-журнал;
* алгоритм определения лица должен обеспечивать скорость обработки не менее 12 кадров в секунду;
* оповещение и фиксация событий;
* предоставление возможности управления настройками ПМ.

## Выводы по разделу

В исследовательском разделе были исследованы предметная область и потребности потенциальных пользователей, проанализированы существующие алгоритмы обнаружения лиц, определения области глаз и определения состояния глаз, проведен обзор существующих решений, определены цели и задачи разработки.

# КОНСТРУКТОРСКИЙ РАЗДЕЛ

## Структура входных и выходных данных

На начальном этапе программный модуль осуществляет чтение настроек из JSON-файла. На основе этих данных модуль инициализации нейросетевых моделей загружает файлы, содержащих данные моделей и их веса, для модели детектирования лиц и модели, определяющую состояние глаз. На основе данных о входном потоке происходит захват видеоизображения и получение видеокадров из видеопотока. Инициализированная модель и кадр из видеопотока поступают на вход программного модуля детектирования лице. Информация о местоположении лица и изображение подаются на вход модуля определения области глаз. Два изображения, содержащие правый и левый глаз соответственно, поступают на вход программного модуля определения состояния глаз. Информация о текущем состоянии двух глаз передается программному модулю обработки событий. Программный модуль обработки событий принимает решение о наступлении засыпания и, в случае необходимости, подает звуковой сигнал. Все события заносятся в файл-журнал, а также производится запись соответствующих видеопоследовательностей. Таким образом, входные и выходные данные представляют собой файл-настроек, бинарные модели для алгоритмов и файл-журнал, видеофайлы событий соответственно, что отражается на рисунке 2.1.



Рисунок 2.1 – Схема данных программного модуля

Запуск программного модуля может происходить с различными параметрами файла настроек, такими как:

* выбор источника данных. Источником данных может быть USB или RTSP камера. Также указывается ссылка для входного источника данных для RTSP камеры или id для USB;
* путь к файлу модели для определения состояний глаз;
* пороговые значение для трех уровней детектора лиц. Если детектор не может определить положение лица в кадре, можно снизить пороговое значение. Если детектор начинает определять ложные объекты как лицо человека, порог можно повысить. Значение порога устанавливается в пределах от 0 до 1;
* путь к файлу модели для детектора антропометрических точек;
* время события засыпание и процент отношения закрыт глаз к открытым;
* массив из 4 чисел для алгоритма трекинга.

Выходными данными программного модуля является файл-журнал, который сохраняет историю работы ПМ. В котором хранится:

* информация о результатах работы каждого алгоритма;
* статистку за время сессии;
* информация об ошибках, возникших в ходе работы модуля;
* состояние водителя: штатная работа, спит.

## Выбор платформы и инструментальных средств

### Выбор языка программирования

Выбор проводился среди языков, используемых для машинного обучения, компьютерного зрения и нейронных сетей. Таковыми являются: С++, С#, Python, R, Java и JavaScript.

C++ [26] – компилируемый, статически типизированный язык программирования общего назначения. Поддерживает такие парадигмы программирования, как процедурное, объектно-ориентированное, обобщённое программирование. Язык имеет богатую стандартную библиотеку, которая включает в себя распространённые контейнеры и алгоритмы, ввод-вывод, регулярные выражения, поддержку многопоточности и другие возможности. C++ сочетает свойства как высокоуровневых, так и низкоуровневых языков.

C# [28] – современный объектно-ориентированный и типобезопасный язык программирования. C# является объектно-ориентированным языком, но поддерживает также и компонентно-ориентированное программирование. Язык автоматически освобождает память, занятую уничтоженными и неиспользуемыми объектами, строгая типизация языка не позволяет обращаться к неинициализированным переменным, выходить за пределы индексируемых массивов или выполнять неконтролируемое приведение типов. C# является платформо-независимым языком. Он транслируется в промежуточный код, который может быть запущен на соответствующей виртуальной машине.

Python [27] – высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Python поддерживает несколько парадигм программирования, в том числе структурное, объектно-ориентированное, функциональное, императивное и аспектно-ориентированное программирование. Основные архитектурные черты – динамическая типизация, сборка мусора, механизм обработки исключений, поддержка многопоточных вычислений и высокоуровневые структуры данных.

R [30] – интерпретируемый язык программирования, основным способом работы с которым является командный интерпретатор. Язык объектный: любой программный объект в нём имеет набор атрибутов — именованный список значений, определяющих его. Пакеты для R могут разрабатываться на других языках программирования, в том числе на Си, что позволяет, с одной стороны, скомпенсировать ограниченность изобразительных средств самого языка R, а с другой — при необходимости достигнуть высоких показателей вычислительной производительности.

Java [31] – строго типизированный объектно-ориентированный язык программирования. Приложения Java обычно транслируются в специальный байт-код, поэтому они могут работать на любой компьютерной архитектуре, для которой существует реализация виртуальной Java-машины. Поддерживает императивную, объектно-ориентированную, процедурную и скалярные парадигмы программирования.

JavaScript [29] – мультипарадигменный язык программирования. Поддерживает объектно-ориентированный, императивный и функциональный стили. Обычно используется как встраиваемый язык для программного доступа к объектам приложений. Наиболее широкое применение находит в браузерах как язык сценариев для придания интерактивности веб-страницам. Основные архитектурные черты: динамическая типизация, слабая типизация, автоматическое управление памятью, функции как объекты первого класса.

Проведен сравнительный анализ этих языков программирования, который представлен в таблице 2.1.

Таблица 2.1 – Сравнительная таблица языков программирования

| Язык  Критерий | R | Java | C++ | Python | C# | JavaScript |
| --- | --- | --- | --- | --- | --- | --- |
| Кроссплатформенность | + | + | + | + | +/– | + |
| Знание языка | –/+ | +/– | +/– | + | +/– | +/– |
| Поддержка фреймворков для машинного обучения | + | –/+ | + | + | +/– | +/– |
| Динамическая типизация | + | – | – | + | + | + |
| Высокая скорость разработки | + | +/– | – | + | +/– | +/– |
| Использование языка для других проектов компании | – | + | + | + | – | – |
| Возможность написания юнит-тестов без установки дополнительных модулей | – | + | – | + | + | + |
| Поддержка ООП | + | + | + | + | + | + |

Проведенный анализ существующих языков программирования показал (таблица 2.1), что язык Python является наилучшим инструментом для поставленных целей и задачи. Он кроссплатформенный, обладает динамической типизация, поддерживает все известные фреймворки для машинного обучения, а простота и гибкость языка позволяет увеличить скорость разработки программного обеспечения.



### Выбор среды разработки

Интегрированная среда разработки (IDE) – комплекс программных средств, используемый программистами для разработки программного обеспечения (ПО). IDE позволяет значительно ускорить и упростить этап разработки, поэтому среда разработки должна предоставлять удобный инструментарий для программирования, отладки и тестирования.

PyCharm [32] – полнофункциональная среда разработка для Python от компании JetBrains. Предоставляет средства для интегрированного модульного тестирования, интегрированного контроля версий, анализа и рефакторинга кода, в состав также входит графический отладчик и автодополнение кода. PyCharm работает под операционными системами Windows, Mac OS X и Linux.

Sublime Text [33] – редактор кода, поддерживающий большую часть языков программирования, в том числе и Python. Обладает такими особенностями, как быстрая навигация, командная палитра, API плагинов на Python, одновременное редактирование и высокая степень настраиваемости. Поддерживает все современные платформы. Возможности редактора могут быть расширены при помощи дополнительных пакетов.

Atom [36] – текстовый редактор, написанный с использованием с использованием Electron – фреймворка для создания кроссплатформенных приложений для десктопа средствами JavaScript, HTML и CSS. Поддержка Python подключается с помощью расширений. Благодаря Electron поддерживает все операционные системы.

Visual Studio [38] – полнофункциональная IDE от Microsoft, которая во многом сопоставима с Eclipse. Доступная на Windows и Mac OS, Visual Studio представлена как в бесплатном (Community), так и в платном (Professional и Enterprise) вариантах. Visual Studio позволяет разрабатывать приложения для разных платформ и предоставляет свой собственный набор расширений. Python Tools for Visual Studio (PTVS) позволяет писать на Python в Visual Studio и включает в себя Intellisense для Python, отладку и другие инструменты.

Spyder [37] – IDE с открытым исходным кодом для Python, специально оптимизированная для data science. Spyder поставляется вместе с пакетом Anaconda. Обладает функциональностью, которую можно ожидать от стандартной среды разработки, наподобие редактора кода с подсветкой синтаксиса, автодополнения кода и встроенного обозревателя документации. Поддерживает Windows, macOS и Linux.

Visual Studio Code (VS Code) [34] – редактор кода для кроссплатформенной разработки веб- и облачных приложений. Включает в себя отладчик, инструменты для работы с Git, подсветку синтаксиса, автодополнения и эффективные средства для рефакторинга. Обладает обширными возможностями для настройки: пользовательские темы, сочетания клавиш и файлы конфигурации.

Jupyter Notebook [35] – это командная оболочка для интерактивных вычислений для языка программирования Python, которая предоставляет расширенную интроспекцию, блочный запуск, подсветку кода и автоматическое дополнение.

Таблица 2.2 – Сравнительная таблица сред разработки

| Среда  Критерий | PyCharm | Sublime Text | Atom | Visual Studio | Spyder | Visual Studio Code | Jupyter Notebook |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Блочный запуск кода | – | – | – | – | + | – | + |
| Опыт использования | + | – | – | + | – | – | + |
| Автодополнение | + | плагин | плагин | + | + | + | + |
| Кроссплатформенность | + | + | + | + | + | + | + |
| Визуализация результатов выполнения кода | – | – | – | – | + | – | + |
| Нативная поддержка Python | + | – | – | + | + | – | + |
| Установленное ПО | – | – | – | + | – | – | + |

Проведенный анализ существующих сред разработки показал (таблица 2.2), что Jupyter Notebook лучше всего подходит для решения поставленных целей и задач.

### Выбор фреймворка для машинного обучения

Фреймворк – это набор инструментов и стандартных реализаций для обеспечения более быстрой разработки какого-либо программного продукта. Выбор фреймворка – это важный этап разработки, так как важно понимать плюсы и минусы различных решений, знать и оценивать пределы их возможностей, а также понимать лучшие варианты использования каждого решения. В качестве фреймворка для машинного обучения был выбран Tensorflow [44].

Tensorflow – один из лучших и популярнейших инструментов для МО, созданный компанией Google. API для библиотеки реализован для Python, но также поддерживаются R, C#, C++, Java и некоторые другие языки. Следствием популярности фреймворка стала подробная документация, включающая не только версию на официальном сайте, но и сторонние источники. Документация описывает как структуру фреймворка, так и дополнительные инструменты, позволяющие отслеживать процесс обучения нейронной сети, например, визуализация с TensorBoard.

Плюсы:

* фреймворк основан на Python;
* предлагает мощные средства мониторинга процесса обучения моделей и визуализации;
* поддерживает распределенное обучение.

Минусы:

* проигрывает по скорости работы в эталонных тестах, в сравнении с CNTK и MXNet;
* имеет более высокий входной порог вхождения для начинающих;
* достаточно низкоуровневая и требует много шаблонного кода.

Несмотря на существующие минусы, TensorFlow полностью подходит для выполнения поставленных задач.

## Программная архитектура и алгоритм работы

### Общий алгоритм работы



Рисунок 2.2 – Общий алгоритм работы программного модуля

Как показано на рис. 2.2, программный модуль считывает настройки из JSON-файла. На основе данных в этом файле происходит инициализация нейросетевых моделей детектирования лица и определения состояния глаз. Происходит захват видео из видеопотока. В бесконечном цикле читаются и обрабатываются видеокадры. Обработка состоит из применения детектора лица, который определяет наличие и положение головы на каждом кадре. Если лицо присутствует в кадре, то применяются алгоритмы определения области глаз и определения состояния глаз. Результат работы алгоритма определения состояния глаз передается модулю обработки событий, который, в случае необходимости, подаст авариный звуковой сигнал и произведет запись соответствующего видеофайла. Этот модуль также записывает результаты работы в лог-файл.

### Детектирование лиц

В разделе 1.4.2 была выбрана архитектура MTCNN (рис. 2.3) для детектирования лиц.

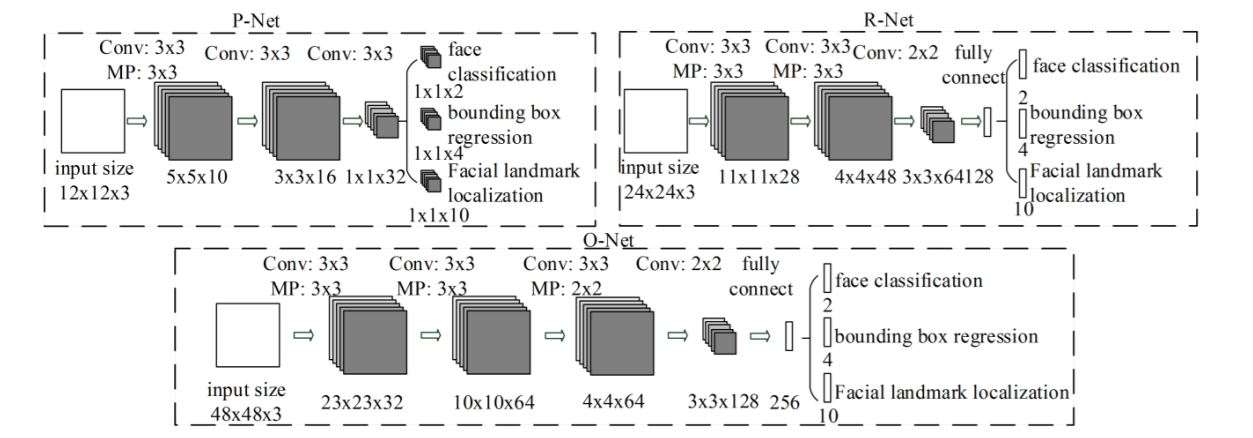


Рисунок 2.3 – Архитектура MTCNN

На вход MTCNN принимает изображение любого размера. Но прежде, необходимо преобразовать его в пирамиду изображений. Это подразумевает под собой создание нескольких копий одного и того же изображения разных размеров (рисунок 2.4).

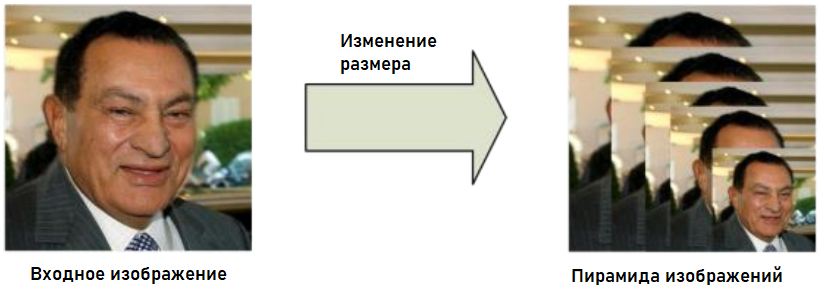


Рисунок 2.4 – Преобразование входного изображение в пирамиду изображений

Общую схему работы MTCNN можно представить на основе трех этапов.

Первый этап. Используется полностью свёрточная нейронная сеть Proposal Network (P-Net). Для каждой уменьшенной копии изображения используется ядро 12х12, которое сканирует изображение в поисках лица. Сканирование начинается в верхнем левому углу. Эта часть изображения передается на вход P-Net, которая возвращает координаты ограничивающей рамки, если она считает, что данная область является лицом. Затем процесс повторяется, сдвигая ядро на два пикселя вправо или на один пиксель вниз. Шаг в 2 пикселя позволяет существенно снизить сложность вычислений без существенного снижения точности. После описанной процедуры следует собрать всех выходные данные P-Net воедино (рисунок 2.5).

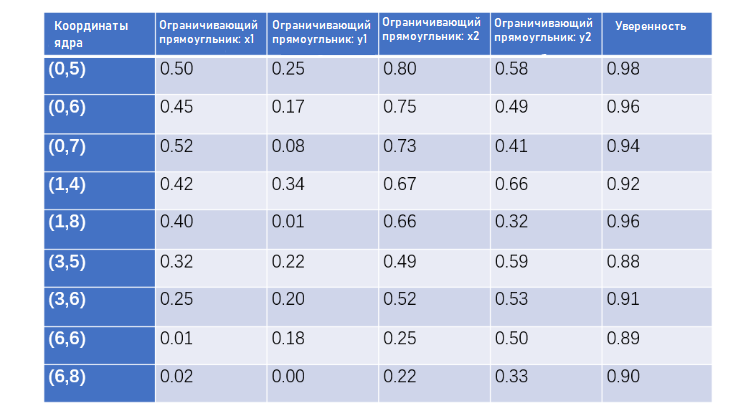


Рисунок 2.5 – Пример вывода P-Net

После этого используется метод подавления немаксимумов (non-maximum suppression), чтобы избавиться от сильно перекрывающих друг друга ограничивающих прямоугольников. Этот метод применяют один раз для каждого масштабированного изображения, а затем ещё один раз со всеми оставшимися кандидатами.

Второй этап. Выход ИНС P-Net после применения метода подавления немаксимумов становится входом ИНС Refine Network (R-Net). Для каждого ограничивающего прямоугольника мы создаем массив и копируем туда значения пикселей из ограничивающей области. В случае, если ограничивающий прямоугольник выходит за границы изображения, происходит копирование лишь части изображения, а остальные значения массива полагают равными нулю. После этого размеры массивов изменяют до 24х24 пикселей и нормализуют. Теперь у нас есть множество массивов 24х24 пикселя (столько же, сколько ограничивающих рамок пережили первый этап), которые подаются на вход R-Net. Выход R-Net аналогичен выходу P-Net: он включает в себя координаты новых, более точных ограничивающих рамок, а также уровень достоверности каждой из них. Так же, как и в P-Net, мы применяем метод подавления немаксимумов.

Третий этап. Выход R-Net преобразуется во вход O-Net таким же образом, как и выход P-Net преобразовывался во вход R-Net, только теперь вместо массивов 24х24 пикселя будут массивы 48х48 пикселей. Выход O-Net немного отличается от выходов P-Net и R-Net. O-Net предоставляет 3 выхода: координаты ограничивающего прямоугольника, координаты 5 лицевых ориентиров и уровень достоверности каждого прямоугольника. Наконец, мы последний раз применяем методов подавления немаксисмумов. На этом этапе должен остаться только один ограничивающий прямоугольник для каждого лица.

### Алгоритм трекинга

Скорость работы алгоритма определения лица можно значительно увеличить, если уменьшить область поиска. Проанализировав более полумиллиона видеокадров людей, управляющих транспортными средствами, я пришёл к выводу, что статистические методы анализы могут выделить область, в которой, с определенной уверенностью, будет находиться лицо (рисунок 2.6).

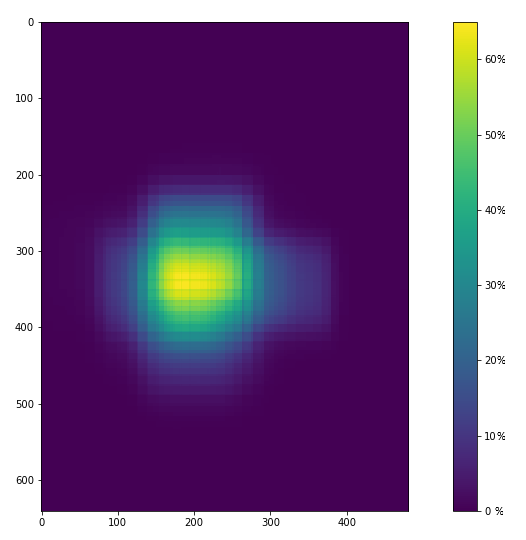


Рисунок 2.6 – Тепловая карта расположения лица

Идеализируем нашу модель, используя некоторые упрощения:

* будем считать, что камера всегда находится в статическом положении;
* размер изображения фиксированного размера;
* в каждый момент времени в видеокадре находится не более одного человека;
* для анализа будут использованы только такие кадры, что если в видеокадре с номером i найдено лицо, а следующий видеокадр, в котором было найдено лицо, имеет номер j, то модуль разности между i и j равен 1.

Пусть х1, y1, x2, y2 – координаты прямоугольника, описывающего область лица, на текущем видеокадре, а х1\*, y1\*, x2\*, y2\* – координаты прямоугольника, описывающего область лица, на следующем видеокадре. Размеры входного изображения – 640х480 пикселей, то есть:

* 0 ≤ х1, x2 , х1\*,x2\*≤ 640;
* 0 ≤ y1, y2 , y1\*,y2\*≤ 480.

В качестве математической модели будем рассматривать функцию (5), которая получает на вход вектор из 4 элементов, который содержит информацию о координатах прямоугольника, где до этого было найдено лицо, а возвращает новый вектор из 4 элементов, где следует искать это лицо в следующий момент времени.

(5)

Очевидно, что задача сводится к тому, чтобы изменить поданную на вход функции область так, чтобы область лица на следующем кадре, целиком находилась в области, являющейся значением функции. При этом необходима, чтобы эта область была как можно меньше.

При дальнейшем анализе использовались 488 тысячи предобработанных видеокадров. Для решения поставленной задачи предлагается рассмотреть распределение разности координат для х1,y1,x2,y2 (рисунок 2.7).

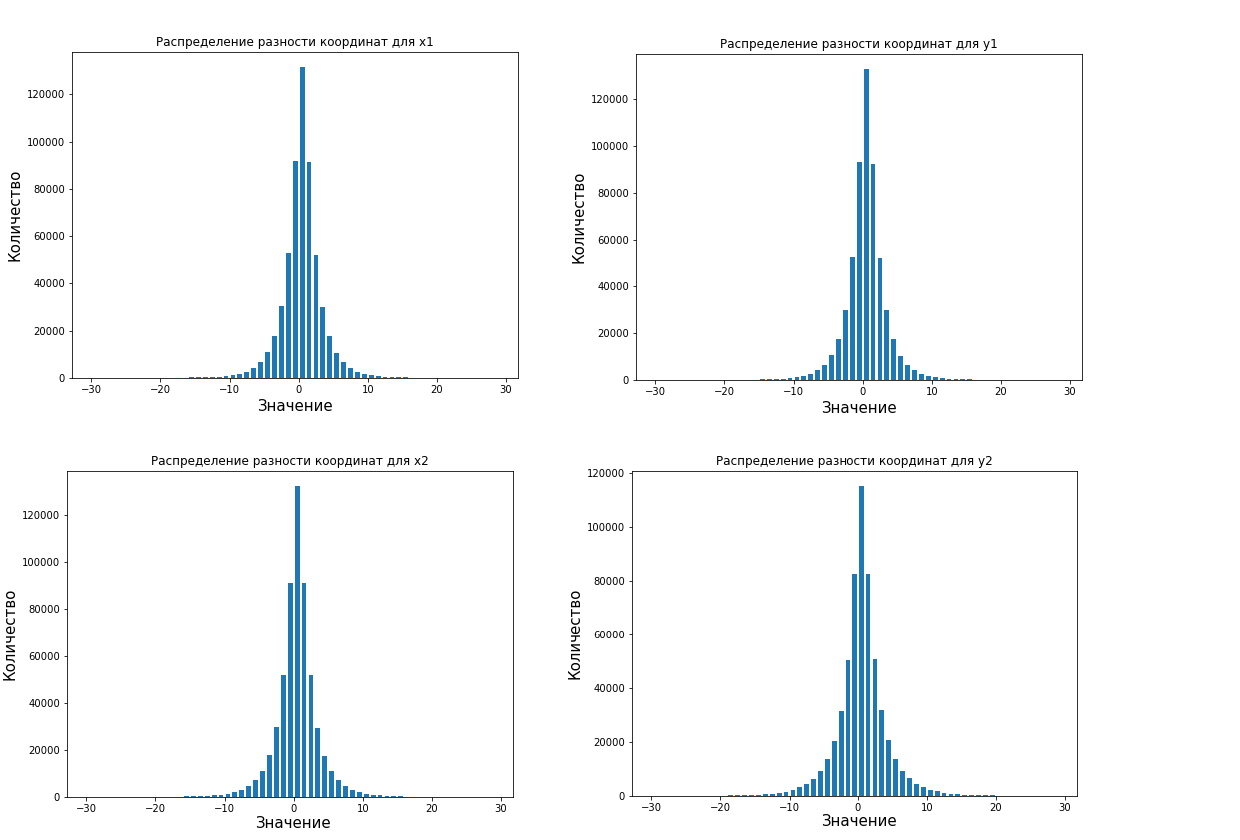
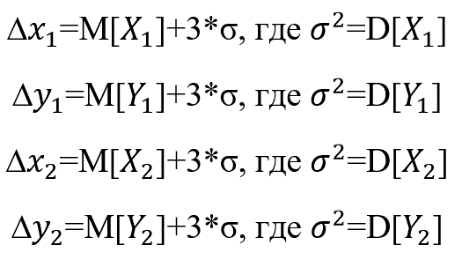


Рисунок 2.7 – Распределения разности координат для х1,y1,x2,y2

Распределения являются нормальными, одномодальными. Воспользуемся неравенством Чебышева. Ожидается, что разность двух координат не превышает своего математического ожидания, более чем на 3 стандартных отклонения, в 95.06 % всех случаев. Исходя из этих рассуждений, запишем формулами величины, на которые будет изменять координаты.

(6)

(7)

(8)

(9)

Значения математического ожидание и дисперсии внесём в таблицу (таблица 2.3).

Таблица 2.3 – Значения дисперсии и математического ожидания для разности координат.

| X | M[X] | D[X] |
| --- | --- | --- |
| x1 | 0.015 | 9.864 |
| y1 | 0.003 | 9.51 |
| x2 | -0.016 | 10.19 |
| y2 | 0.013 | 14.89 |

Окончательный вид функции:

(10)

Согласно теоретическим расчётам, данная функция должна с вероятность 95% правильно преобразовывать координаты для следующего видеокадра, что позволит значительным образом уменьшить вычислительные затраты при использовании алгоритма детектирования лица.

### Алгоритм определения области глаз

В разделе 1.4.2 было показано, что с задачей определения области глаз лучше всего справляется предиктор на основе антропометрических точек. Предиктор от dlib предсказывает 68 антропометрических точек лица, при этом для описания области глаз достаточно 12 точек.

Библиотека dlib предоставляет разработчику возможность создания своего собственного предиктора на основе каскада деревьев регрессии и градиентного бустинга. Для обучения предиктора антропометрических точек обычно используют набор данных iBUG 300-W. Сам набор состоит из 7764 изображений, которые были вручную размечены 68 точками. Использование собственного предиктора позволяет получить преимущество в скорости и размере модели. Так как изначально для каждого изображения существует аннотация из 68 точек, то следует изменить ее, оставив только точки с 36 по 47 включительно.

На качество, скорость, время обучения и размер модели влияют 7 основных параметров.

Глубина каждого дерева регрессии. Небольшие значения глубины приведут к более мелким деревьям, которые быстрее, но потенциально менее точны. Большие значения параметра создадут более глубокие деревья, которые будут медленнее, но более точными.

Параметр регуляризации. Параметр регуляризации – это значение с плавающей точкой в диапазоне [0;1]. Значения, близкие к 1, сделают нашу модель более подходящей для данных обучения, но потенциально могут привести к переобучению. Значения, близкие к 0, помогут нашей модели лучше обобщать. Одно есть один нюанс: чем ближе значение параметра регуляризации к 0, тем больше данных для обучения понадобится.

Количество каскадов, используемых для обучения предиктора. Данный параметр окажет значительное влияние как на точность, так и на размер модели. Увеличение количества каскадов обычно положительно влияет на точность модели, но при этом увеличивает ее размер, а уменьшение количества каскадов – наоборот. Как правило, используют значение в диапазоне [6;18] Влияние данного параметра на качество предсказаний может увидеть на рисунке 2.8.

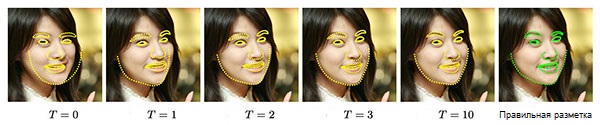


Рисунок 2.8 – Влияние количества каскадов на точность предиктора

Количество пикселей, используемых для генерации объектов для случайных деревьев. Чем больше пикселей мы используем, тем медленнее будет работать наша модель, что может потенциально привести к более высокой точности. Для встраиваемых систем рекомендуется использовать небольшие значения для более быстрой скорости прогнозирования.

Параметр, отвечающий за выбор функции на каждом каскаде при обучении. Оказывает существенное влияние на то, сколько времени будет занимать время обучения модели. Чем больше значение параметра, тем выше вероятность того, что мы получим более точный предиктор. Неправильный выбор данного параметра может привести к тому, что время обучение будет несоизмеримо велико.

Параметр, контролирующий объем дополнительных данных. Значение данного параметра определяет количество случайных деформаций, применяемых к обучающим данным. Выполняет регуляризацию и увеличивает обобщающую способность модели.

Количество потоков/ядер процессора. По умолчанию, этот параметр равен количеству ядер, доступных в системе. Чем больше потоков используются для обучения, тем меньше время обучения предиктора.

Экспериментальным образом было установлено, что лучше всего себя показывает модель со следующими параметрами:

* глубина каждого дерева = 2;
* параметр регуляризации = 0.25;
* количество каскадов = 12;
* количество пикселей = 500;
* параметр, отвечающий за выбор функции на каждом каскаде при обучении = 100;
* параметр, контролирующий объем дополнительных данных = 20.

Для использования нейросетевого классификатора состояния глаз необходимо, чтобы область глаза была целиком помещена в прямоугольник или квадрат. Эмпирическим путем были найдены соответствующие параметры фигуры – квадрат со стороной 40 пикселей. Для улучшения качества классификации область глаза центрируется.

### Алгоритм определения состояния глаз

В разделе 1.4.3 были выявлены недостатки существующих алгоритмов. Для преодоления данных недостатков алгоритм определения состояний глаз был отделен от алгоритма определения области глаз. Так как алгоритм определения области глаз позволяет эффективным способом заключить весь глаз целиком в квадрат со стороной 40 пикселей, это позволяет использовать неглубокую ИНС для решения задачи классификации. Инвариантность к цветовому пространству можно добиться путем предобработки изображения перед тем, как оно попадёт на вход ИНС. Предобработка состоит из фильтра, преобразующего любое изображение в одноканальное изображение в градациях серого.

Архитектура ИНС (рис. 2.9): состоит из 4 слоев: первые три слоя являются сверточными и извлекают признаки из изображения, последний слой полносвязный и отвечает за классификацию.

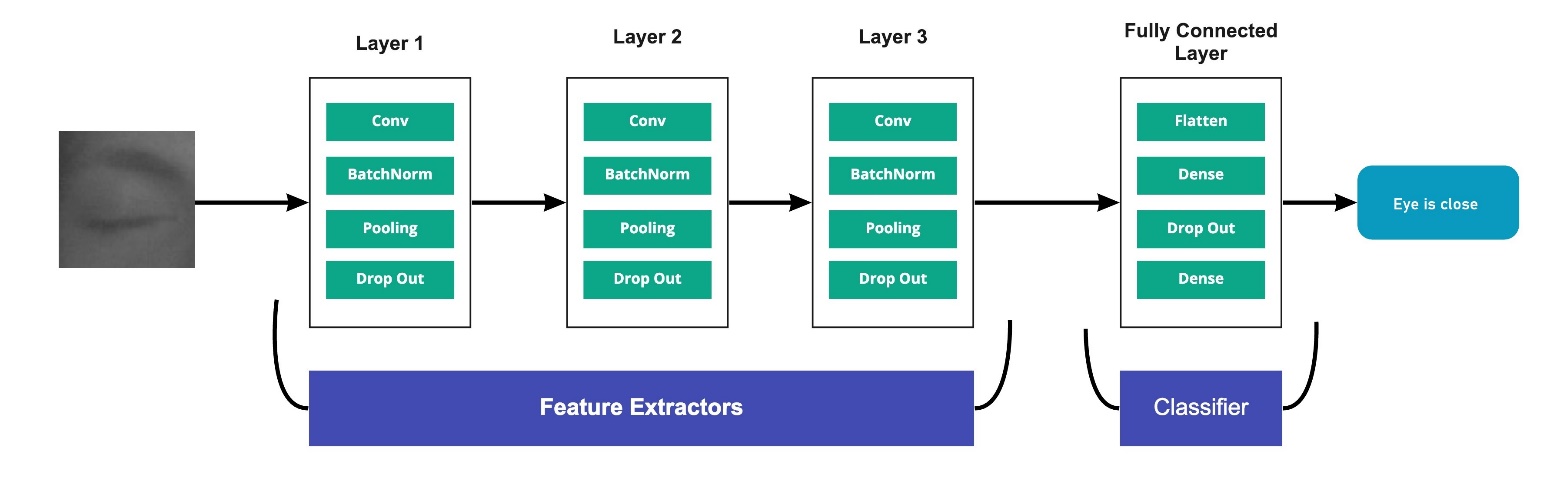


Рисунок 2.9 – Архитектура ИНС определения состояния глаз

### Метрики в задачах классификации

Пусть дана выборка X = {x1,…,xn}. Для каждого объекта выборки мы знаем бинарный ответ yi = y(xi) ∈ {open, close}, i = 1,…,n. Также дан алгоритм a: X → Y.

На основе ответов алгоритма и истинных меток класса на этом объекте происходит построение таблицы сопряженности, которая составляется для каждого класса в отдельности.

Таблица 2.2 – Таблица сопряженности для класса i

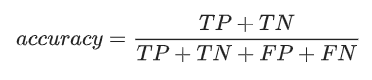
|  |  |  |  |
| --- | --- | --- | --- |
| Категория i | | Истинный класс объекта | |
| Положительная | Отрицательная |
| Ответ алгоритма | Положительная | TP | FP |
| Отрицательная | FN | TN |

В таблице 2.2 содержится информация сколько раз система приняла верное и сколько раз неверное решение по экземплярам заданного класса. А именно:

* TP – истинно-положительное решение;
* TN – истинно-отрицательное решение;
* FP – ложноположительное решение;
* FN – ложноотрицательное решение.

Таким образом, ошибки классификации бывают двух видов: ложноположительное решение (FP) и ложноотрицательное решение (FN).  В статистике первый вид ошибок называют ошибкой I-го рода, а второй – ошибкой II-го рода.

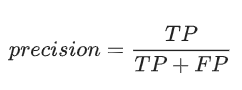
Интуитивно понятной, очевидной и почти неиспользуемой метрикой является доля правильных ответов алгоритма (accuracy) (11).



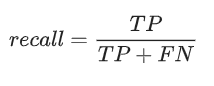
(11)

Эта метрика почти никогда не используется в задачах с неравными классами. По этой причине от общей для всех классов метрики переходят к отдельным показателям качества классов. Для оценки качества работы алгоритма на каждом из классов по отдельности вводят метрики точность (precision) и полнота (recall).

Точность (precision) можно толковать как долю объектов, названных классификатором положительными и при этом действительно являющимися положительными, а полнота (recall), в свою очередь, показывает, какую долю объектов положительного класса из всех объектов положительного класса нашел алгоритм.

Таким образом, точность (12) и полнота (13) могут быть вычислена по следующим формулам:

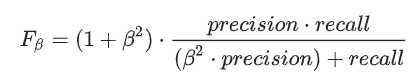
(12)



(13)

Именно введение точности не позволяет нам записывать все объекты в один класс, так как в этом случае мы получаем рост уровня ложноположительных ответов. Полнота демонстрирует способность алгоритма обнаруживать данный класс вообще, а точность – способность отличать этот класс от других классов.

Существует несколько различных способов объединить точность и полноту в единый критерий качества. F-мера (в общем случае  Fβ) — среднее гармоническое точности и полноты:



(14)

β в данном случае определяет вес точности в метрике, и при β=1 это среднее гармоническое с множителем 2. F-мера достигает максимума при полноте и точности, равными единице, и близка к нулю, если один из аргументов близок к нулю. F-мера является хорошим кандидатом на формальную метрику оценки качества классификатора.

## Выводы по разделу

В процессе написания данного раздела проведен комплекс работ, в результате которых:

* определен язык программирования для разработки ПМ КСВ;
* выбрана среда разработки;
* выбраны фреймворк для работы с нейронными сетями;
* составлен общий алгоритм работы ПМ КСВ;

# ИСПЫТАТЕЛЬНЫЙ РАЗДЕЛ

## Технологии программирования

### Достоинства и недостатки языка программирования Python

Python - это интерпретируемый язык программирования, изначально реализующий объектно-ориентированную парадигму. Обладает низким порогом вхождения, но при этом сочетает в себе гибкость и выразительность. Это язык более высокого уровня нежели Pascal, C++ что достигается, в основном, за счет встроенных высокоуровневых структур данных (списки, словари, кортежи).

Достоинства языка:

* интерпретатор языка реализован на всех платформах и операционных системах. Первым таким языком был C, однако его типы данных на разных машинах могли занимать разное количество памяти и это служило некоторым препятствием при написании действительно переносимой программы. Python же таким недостатком не обладает;
* расширяемость языка. Это означает, что у любого желающего есть возможность совершенствовать язык. Интерпретатор написан на С и исходный код доступен для любых манипуляций. В случае необходимости, можно вставить его в свою программу и использовать как встроенную оболочку. Или же, написав на C свои дополнения к Python и скомпилировав программу, получить "расширенный" интерпретатор с новыми возможностями;
* наличие большого числа подключаемых к программе модулей, обеспечивающих различные дополнительные возможности. Примеры модулей: Numerical Python, Tkinter, OpenGL.

Недостатки языка:

* низкое быстродействие. Невысокая скорость выполнения программ обсуловлена интерпретируемостью языка;
* невозможность модификации встроенных типов данных. В Python отсутствует возможность модифицировать встроенные классы, такие, как float, str, bool и другие, что, однако, позволяет Python потреблять меньше оперативной памяти и быстрее работать;
* глобальная блокировка интерпретатора, накладывающая ограничения на потоки. Это приводит к тому, что нельзя использовать нескольких процессоров одновременно.

Обзор особенностей:

* Python, в отличие от многих языков (Pascal, C++, Java, и т.д.), не требует описания переменных. Они создаются в месте их инициализации, т.е. при первом присваивании переменной какого-либо значения. Значит, тип переменной определяется типом присваиваемого значения;
* тип переменной не является неизменным. Любое присваивание для нее корректно и приводит лишь к тому, что типом переменной становится тип нового присваиваемого значения;
* Python в отличие от Pascal, C, C++ не поддерживает работу с указателями, динамической памятью и адресную арифметику. Как известно, указатели служат источником трудноуловимых ошибок, и работа с ними относится больше к программированию на низком уровне. Для обеспечения большей надежности и простоты они не были включены в Python.

Скрипты Python используются для широкого спектра задач - от аналитики до создания программ и даже XML-роботов. Вот их неполный список:

* анализ данных;
* разработка интернет-сайтов;
* создание десктопных и веб-приложений;
* управление системами и базами данных;
* автоматизация процессов;
* создание прототипов;
* тестирование программ;
* проведение расчётов (с использованием расширений, например, NumPy);
* машинное обучение.

### Основные преимущества использования фреймворка Keras

Keras [46] – это минималистичная библиотека, основанная на Python. Она была разработана инженером компании Google, Франсуа Шолле, в целях ускорения экспериментов. Keras совместим с TensoFlow, Theano, CNTK и MXNet. Keras поддерживает широкий спектр слоев нейронных сетей, таких как сверточные слои, рекуррентные и полносвязные. Эта библиотека предоставляет доступный высокопроизводительный интерфейс, включая необходимые абстракции и строительные блоки, для решения задач машинного обучения с акцентом на современное глубокое обучение.

Преимущества:

* быстрое создание прототипов и экспериментирование;
* библиотека готовых моделей машинного обучения;
* наличие простого и интуитивно-понятного интерфейса;
* хорошая документация;
* встроенная поддержка обучения на нескольких GPU;
* быстроразвивающийся фреймворк;
* совместим с TensorFlow, Microsoft Cognitive Toolkit, Theano и MXNet. TensorFlow и Theano являются наиболее часто используемыми численными платформами на Python для разработки алгоритмов глубокого обучения;
* поддерживает GPU от NVIDIA, TPU от Google, GPU с Open-CL, такие как AMD.

Недостатки:

* может оказаться слишком высокоуровневой библиотекой, что может не позволить создать специализированную модель глубокого обучения;
* ограничен количеством бэкендов;
* обладает определенными сложностями при встраивании моделей.

## Используемые сторонние библиотеки

### Библиотека Numpy

NumPy [47] – библиотека с открытым исходным кодом для языка программирования Python. Библиотека NumPy представляет собой набор инструментов для научных расчётов. Библиотека содержит описание специального типа данных ndarray, реализующего возможности многомерных массивов (массивов произвольной размерности). Кроме того, библиотека включает в себя функции для работы с многомерными массивами типа ndarray, а также имеет возможности для решения задач линейной алгебры, использования преобразования Фурье и работы со случайными числами.

Массив ndarray может содержать последовательности данных одного типа, таких, например, как:

* сетка экспериментально измеренных величин (например, показания термометра в разные моменты времени);
* сетка расчётных величин (например, координаты материальной точки в заданные моменты времени, полученные с помощью численного решения уравнения движения);
* пиксели изображения;
* прочие.

Математические алгоритмы, реализованные на интерпретируемых языках (например, Python), часто работают гораздо медленнее тех же алгоритмов, реализованных на компилируемых языках (например, Фортран, Си, Java). Библиотека NumPy предоставляет реализации вычислительных алгоритмов (в виде функций и операторов), оптимизированные для работы с многомерными массивами. В результате любой алгоритм, который может быть выражен в виде последовательности операций над массивами (матрицами) и реализованный с использованием NumPy, работает так же быстро, как эквивалентный код, выполняемый в MATLAB.

### Библиотека OpenCV

OpenCV [48] представляет собой библиотеку программного обеспечения для компьютерного зрения с открытым исходным кодом и машинного обучения.

OpenCV создан для обеспечения общей инфраструктуры приложений основаных на компьютерном зрении для упрошения использования наработок и решений большого количесва разработчиков в готовых продуктах.

Поскольку OpenCV распространяется под BSD лицензией, любой желающий может использована её как в исследовательских целях, так и в коммерческих.

Библиотека имеет более 2500 оптимизированных алгоритмов, которые включают в себя полный набор как классических, так и самых современных алгоритмов компьютерного зрения и машинного обучения.

Эти алгоритмы могут использоваться для обнаружения и распознавания лиц, идентификации объектов, классификации действий человека в видео, отслеживания движения камеры, отслеживания движущихся объектов, извлечения 3D-моделей объектов, создания 3D-облаков точек из стереокамер, сшивания изображений вместе для получения высокого разрешения изображение всей сцены, найти похожие изображения из базы данных изображений, удалять красные глаза с изображений, сделанных с помощью вспышки, следить за движениями глаз, распознавать декорации и устанавливать маркеры, чтобы накладывать их на дополненную реальность и т. д.

Интерфейсы библиотеки доступны на Java, Matlab, C++ и Python с поддержкой Windows, Linux, Android и Mac OS.

OpenCV обладает модулем dnn [49] для работы с нейронными сетями и глубоким обучением (глубинным обучением). Основная возможность dnn заключается, конечно же, в загрузке и запуске нейронных сетей (inference). При этом модель может быть создана в любом из трех фреймворков глубокого обучения – Caffe, TensorFlow или Torch; способ ее загрузки и использования сохраняется независимо от того, где она была создана. Оставляя только возможность выполнения прямого прохода (forward pass) по сети, можно упростить код, ускорить процесс установки и сборки.

Дополнительно, начиная с версии 3.4.10 и 4.3.0 можно компилировать OpenCV с поддержкой Tengine, что позволяет значительно ускорить работу нейронных сетей на встроенных устройствах под архитектурой ARM. Tengine [50] – это эффективная библиотека логики глубокого обучения для ARM. Поддерживаются как 32-разрядные, так и 64-разрядные архитектуры ARM, если доступны инструкции NEON [51]. Сравнение производительности OpenCV и OpenCV с поддержкой Tengine можно наблюдать на рисунке 3.1.

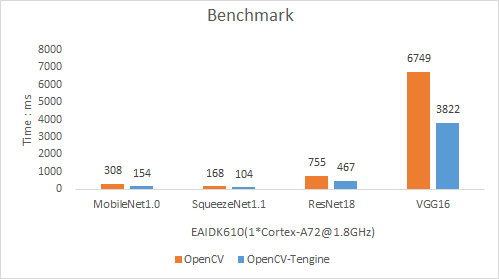


Рисунок 3.1 – Сравнение производительности OpenCV и OpenCV-Tengine

### Библиотека Dlib

Dlib [52] - это современный инструментарий C ++, содержащий алгоритмы машинного обучения и инструменты для создания сложного программного обеспечения на C ++ для решения реальных задач. Он используется как в промышленности, так и в научных кругах в самых разных областях, включая робототехнику, встроенные устройства, мобильные телефоны и большие высокопроизводительные вычислительные среды. Лицензирование Dlib позволяет использовать его в любом приложении бесплатно. По состоянию на 2023 год он содержит программные компоненты для работы с сетями, потоками, графическими интерфейсами пользователей, структурами данных, линейной алгеброй, машинным обучением, обработкой изображений, интеллектуальным анализом данных, XML и текстовым анализом, численной оптимизацией, Байесовскими сетями и многими другими задачами.

## Методы отладки

Отладка – этап разработки компьютерной программы, на котором обнаруживают, локализуют и устраняют ошибки.

Jupyter Notebook обладает своей отличительной способностью запуска кода блоками, что позволяет легко, просто и удобно проводить отладку посредством вывода переменных и их свойств в отдельном блоке (рис. 3.2).

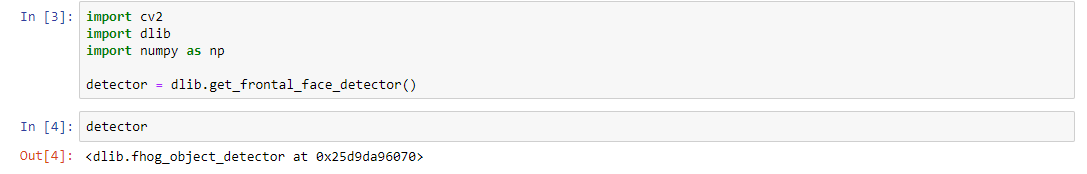


Рисунок 3.2 – Вывод переменной detector

Помимо этого, Python обладает встроенным отладчиком под названием pdb [53] (рис. 3.3). Pdb представляет из себя консольную утилиту, обладающую основным функционалам для отладки кода.

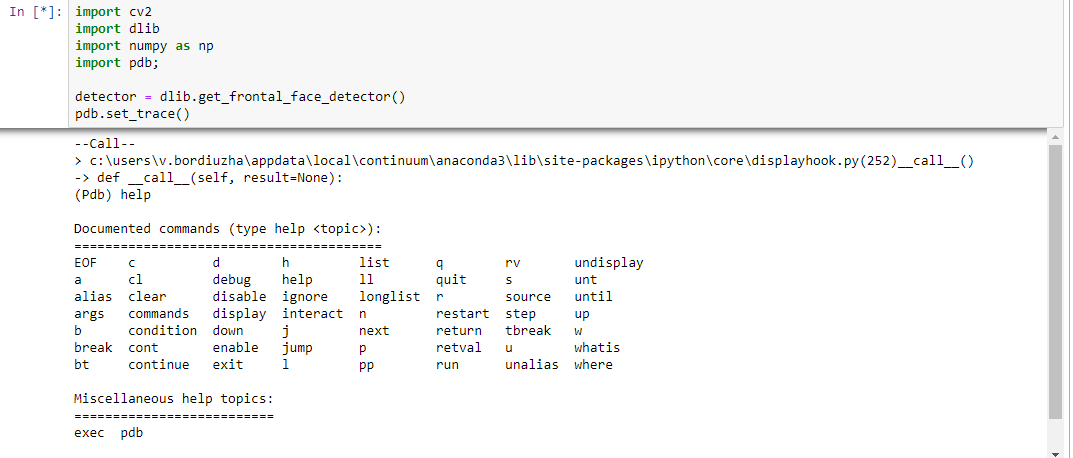


Рисунок 3.3 – Вывод списка команд pdb

Список команд отладчика pdb:

* h (help). Без аргумента выводит список доступных команд. С командой в качестве аргумент выведет полную документацию об этой команде;
* n (next). Выполняет инструкции до следующей строки в текущей функции. Если в текущей строке присутствуют вызовы других функций, они не учитываются;
* s (step). Выполняет одну строку исходного программного кода и останавливает выполнение внутри вызываемых функций;
* c (continue). Выполняет инструкции, пока не будет встречена точка останова. Очень полезна, когда скрипт отрабатывает с исключением, позволяет дойти до строки, где возникло исключение;
* l (list). Выводит листинг исходного программного кода. При использовании без аргументов эта команда выведет текущую строку, пять строк до и пять строк после. При использовании с двумя аргументами – выведет строки из указанного диапазона;
* ll (longlist). Выводит листинг всего метода или функции, в которой мы находимся;
* a (args). Отображает список аргументов текущего метода или функции;
* p. Выводит значение переменной, работает по аналогии print;
* pp. Выводит значение переменной, работает по аналогии pprint;
* until. Продолжает выполнение программы до указанной строки;
* r (return). Выполняет команды до момента выхода из текущего метода или функции;
* u (up). Перемещается на один уровень выше в стеке вызовов;
* d (down). Передвинуться на один уровень ниже в стеке вызовов;
* b (break). Команда для установки breakpoint.

## Методы тестирования нейросетевых алгоритмов

### Подготовка репрезентативной выборки

В ходе работы были построены три различные конфигурации нейросетевого алгоритма определения состояния глаз. Это одна и та же архитектура ИНС, но с разными весами у нейронов. Разные веса нейронов обусловлены тем, что для обучения были использованы различные обучающие выборки.

Первая выборка состоит из двух категорий изображений: глаза открыты и глаза закрыты. При этом, в выборку глаза закрыты были добавлены изображения не только глаз, а случайных объектов.

Вторая выборка состоит только из двух категорий изображений. При этом, из категории глаза закрыты были изъяты изображения не глаз.

Третья выборка содержит больше данных для обучения, чем первая и вторая выборка. Была применена техника аугментации данных на основе простых геометрических преобразований: сдвиги, зеркальное отображение, повороты, масштабирование. Применение небольшого числа преобразований к входному изображению немного изменит его внешний вид, но не изменит метку класса. Данная техника позволяет добиться лучшей обобщающей способности, а также устойчивости к шумам и изменениям данных.

Для каждой модели была измерена скорость обработки изображения, а также вычислены метрики accuracy, precision, recall и F1-мера для закрытых и открытых глаз для каждой модели в отдельности.

Тестирование скорости работы нейросетевого алгоритма производилось на целевом устройстве Firefly-RK3399 [54].

Тестирование остальных параметров было произведено на другом устройстве, конфигурации которого приведены ниже:

* CPU: Intel Core i7-8700 @ 3.20GHz
* GPU: 8192MB NVIDIA GeForce RTX2070 Gigabyte
* память: 32GB DDR4 Kingston HyperX Fury @ 3200MHz
* операционная система: Windows 10 64-bit

Несмотря на то, что один из наиболее важных параметров предсказательных моделей является точность распознавания, необходимо также учесть влияние полноты и F1-меры.

Тестирование нейросетевых моделей происходило на двух датасетах: датасет, подготовленный вручную, и MRL Eye Dataset. Датасет [55], подготовленный вручную, включает в себя 41 000 изображений, сделанных на камеру с инфракрасной подсветкой. В данном наборе данных, как и в обучающих наборах, глаз считается закрытым только в том случае, если веко полностью опущено и не видно зрачка. В ином случае – глаз является открытым.

MRL Eye Dataset – это крупномасштабный набор данных изображений человеческого глаза. Этот набор данных содержит инфракрасные изображения с низким и высоким разрешением, снятые в различных условиях освещения и на разные устройства. В наборе данных аннотированы следующие свойства:

* идентификатор субъекта. В наборе данных собраны данные 37 разных людей (33 мужчины и 4 женщины);
* идентификатор изображения. Набор данных состоит из 84 898 изображений;
* пол [0 - мужчина, 1 - женщина]. Набор данных содержит информацию о поле для каждого изображения (мужчина, женщина);
* очки [0 - нет, 1 - да]. Информация, если изображение глаза содержит очки, также предоставляется для каждого изображения (с очками и без них);
* состояние глаза [0 - закрыт, 1 - открыт]. Это свойство содержит информацию о двух состояниях глаз (открытый, закрытый);
* отражения [0 - нет, 1 - небольшое, 2 - значительное];
* условия освещения [0 - плохие, 1 - хорошие]. Каждое изображение имеет два состояния степени освещенности в зависимости от количества света во время захвата видео;
* идентификатор датчика [01 - RealSense, 02 - IDS, 03 - Aptina]. Набор данных содержит изображения, снятые тремя различными датчиками (датчик Intel RealSense RS 300 с разрешением 640 x 480, датчик изображения IDS с разрешением 1280 x 1024 и датчик Aptina с разрешением 752 x 480).

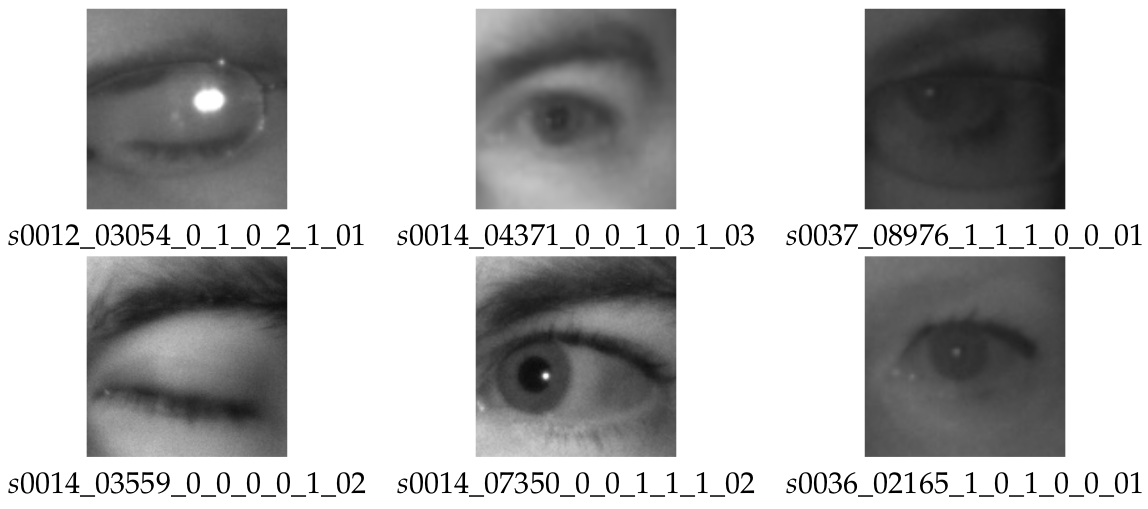


Рисунок 3.4 – Примеры аннотаций изображений

Изображения в MRL Eye Dataset были приведены в соответствие обучающим данным: размер каждого изображения перед тестированием изменяют до 40х40 пикселей. Из набора данных были убраны все изображения в очках, а также были перемещены изображения из категории закрытые глаза в категорию открытые глаза, если веко было опущено не полностью или виден зрачок. Итоговый набор данных состоит из 101 тысячи изображений.



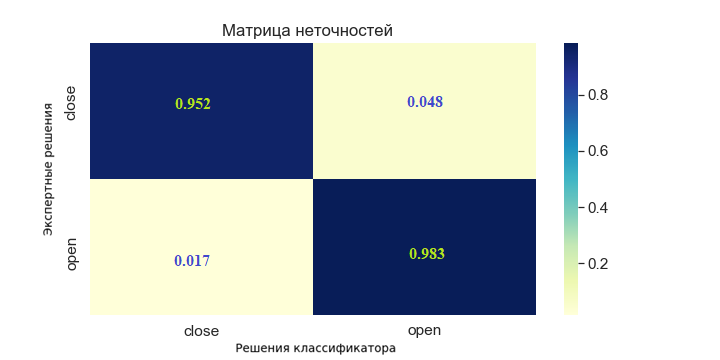
### Результаты тестирования алгоритма определения состояния глаз

Результаты тестирования нейросетевых моделей алгоритма определения состояния глаз представлены в таблице 3.1. Согласно результатам, лучшего всего с задачей справляется cnn\_eyes\_model\_v3, которая обеспечивает долю правильных 97.3%, а также точность распознавания каждого класса на уровне 95%.

Таблица 3.1 – Сравнение моделей определения состояния глаз.

| Модель | Класс | Accuracy, % | Precision, % | Recall, % | F1 |
| --- | --- | --- | --- | --- | --- |
| cnn\_eyes\_model\_v1 | open | 64.3 | 99.7 | 48.5 | 0.652 |
| close | 46.4 | 99.7 | 0.633 |
| cnn\_eyes\_model\_v2 | open | 78.3 | 99.5 | 69 | 0.815 |
| close | 58.9 | 99.2 | 0.739 |
| cnn\_eyes\_model\_v3 | open | 97.3 | 98.3 | 97.7 | 0.981 |
| close | 95.2 | 96.3 | 0.957 |

Для самой точной модели была построена нормализованная матрица неточностей. Матрица неточностей (рис. 3.5) – это матрица размера N на N, где N – это количество классов. Столбцы этой матрицы резервируются за экспертными решениями, а строки за решениями классификатора. Когда классифицируется изображение из тестовой выборки, число, стоящее на пересечении строки класса, который вернул классификатор и столбца класса, к которому действительно относится изображение инкрементируется.

Рисунок 3.5 – Нормализованная матрица неточностей для самой точной модели

Для определения скорости алгоритм запускают на изображении 40х40 пикселей 10 000 раз и измеряют время работы. Затем происходит вычисление среднего времени обработки одного изображения, на основе данного значения рассчитывают FPS.

Таблица 3.2 – Измерение скорости работы алгоритма

|  |  |  |
| --- | --- | --- |
| Время обработки 10 000 изображений, сек | Среднее время обработки изображения, сек | FPS |
| 78 | 0.0078 | 128.2 |

## Тестирование точности алгоритма трекинга

Для оценки качества алгоритма трекинга и проверки адекватности математической модели требуется протестировать работу алгоритма на специально подготовленных данных. Данные представляют собой набор csv-файлов с информацией о местоположении головы на каждом видеокадре соответствующего видеофайла. Пример содержания такого файла представлен на рисунке 3.6.

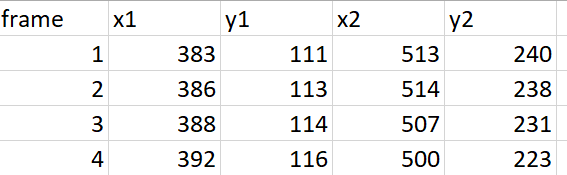


Рисунок 3.6 – Содержание csv-файла

Тестирование представляет из себя обработку i-го видеокадра, то есть расширение области текущего лица, и проверку, что область на i+1 видеокадре, то есть область лица на следующем видеокадре, целиком входит в расширенную область.

Средняя площадь расширенной области равняется 26002 пикселям, что можно представить себя как квадрат со стороной 162 пикселя. Согласно результатам тестирования, в 96.8 % случаев область на i+1 видеокадре целиком поместится в расширенную область i-го кадра.

## Тестирование скорости работы алгоритма детектирования лиц

Тестирование скорости алгоритма производилось тремя способами. В первом случае используется одно изображение и операция детектирования повторяется 1000 раз. Во втором способе уже используется 100 изображений и так же, как и в первом случае, детектирование повторяют 1000 раз. Последний способ состоит из тестирования на видеопоследовательности длинной в одну минуту, записанной на камеру с частотой кадров 20 FPS. Размеры изображений – 640x480 пикселей. Результаты занесены в таблицу 3.3.

Таблица 3.3 – Измерение скорости работы алгоритма детектирования лица

| Номер теста | Общее время, сек | Количество изображений | Количество повторений | Среднее время на обработку одного изображения, сек | Производительность, количество кадров в секунду |
| --- | --- | --- | --- | --- | --- |
| 1 | 63,1 | 1 | 1000 | 0,0631 | 15,85 |
| 2 | 5 800,3‬ | 100 | 1000 | 0,058 | 17,24 |
| 3 | 64,8 | 1200 | 1 | 0,054 | 18,51 |

## Выводы по разделу

В данном разделе были проведены следующие работы:

* Рассмотрены язык программирования Python и фреймворк Keras, использованные при создании программного модуля, оценены их преимущества и недостатки;
* приведено краткое описание сторонних библиотек, использованных в программном модуле;
* проведено тестирование скорости и точности работы алгоритмов определения состояния глаз, алгоритма трекинга и алгоритма детекции лица;

# ОСНОВНЫЕ РЕЗУЛЬТАТЫ И ВЫВОДЫ

В рамках проектной работы решены следующие задачи:

* проведен обзор существующих методов детектирования лица, определения области глаз и классификации состояния глаз;
* разработан алгоритм классификации состояния глаз с точностью классификации выше 95%;
* разработан быстрый алгоритм определения области глаз;
* разработан алгоритм трекинга лица с вероятностью правильного прогнозирования 96.8%;
* разработан программный модуль по контролю за вниманием водителя;
* проведена экспериментальная оценка эффективности разработанного решения.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. **Тендер: Оказание услуг по оперативному мониторингу функционального состояния машинистов** [Электронный ресурс]. – URL: <https://rostender.info/region/moskva-gorod/66178005-tender-okazanie-uslug-po-operativnomu-monitoringu-funkcionalnogo-sostoyaniya-voditelej-gup-mosgortrans> (дата обращения 11.03.2023).
2. **Тендер: Оказание услуг по оперативному мониторингу функционального состояния водителей** [Электронный ресурс]. – URL: <https://rostender.info/region/moskva-gorod/64207466-tender-okazanie-uslug-po-operativnomu-monitoringu-funkcionalnogo-sostoyaniya-mashinistov> (дата обращения 11.03.2023).
3. Система контроля состояния водителя [Электронный ресурс]. – URL: http://avtoritet.su/articles/topics/car-devices/27058/ (дата обращения 4.03.2023).
4. Как мы разработали устройство для контроля внимания водителей: Опыт Яндекс.Такси[Электронный ресурс]. – URL: https://habr.com/ru/company/yandex/blog/461137/ (дата обращения 4.03.2023).
5. DUNOBIL INSOMNIA [Электронный ресурс]. – URL: <http://dunobil.ru/insomnia> (дата обращения 18.02.2023)
6. MDSM-7 [Электронный ресурс]. – URL: <https://www.euromobile.ru/produkciya/sistemy-adas/mdsm-7/> (дата обращения 21.02.2023).
7. MDSM-7 [Электронный ресурс]. – URL: <http://www.movon.co.kr/mdsm7> (дата обращения 21.02.2023).
8. Оптический контроль оператора [Электронный ресурс]. – URL: <https://oko-systems.ru/> (дата обращения 21.02.2023).
9. Yang Ming-Hsuan, Kriegman D., Ahuja N. Detecting Faces in Images: A Survey . – 2002. – URL: <https://www.researchgate.net/publication/3193340_Detecting_Faces_in_Images_A_Survey> (дата обращения 09.04.2023).
10. Татаренков, Д. А. Анализ методов обнаружения лиц на изображении. – 2015. – URL: https://moluch.ru/archive/84/15524/ (дата обращения: 15.04.2023).
11. G. Yang, T. S. Huang. Human Face Detection in Complex Background Pattern Recognition. – 1994. – URL: https://www.sciencedirect.com/science/article/abs/pii/0031320394900175?via%3Dihub (дата обращения 23.04.2023).
12. Гистограмма направленных градиентов [Электронный ресурс]. – URL: https://ru.wikipedia.org/wiki/Гистограмма\_направленных\_градиентов (дата обращения 10.04.2023).
13. Alireza R., Farhad K. Real time eye detection using edge detection and euclidean distance. – 2016. – URL: <https://www.researchgate.net/publication/306018638_Real_time_eye_detection_using_edge_detection_and_euclidean_distance> (дата обращения 21.04.2023).
14. R. Kothari, J.L. Mitchell. Detection of Eye Locations in Unconstrained Visual Images. –1996. – URL: https://ieeexplore.ieee.org/document/560546 (дата обращения 22.04.2023).
15. K.-N. Kim, R.S. Ramakrishna. Vision-Based Eye-Gaze Tracking for Human Computer Interface. – 1999. – URL: <https://www.researchgate.net/publication/3829197_Vision-Based_Eye-Gaze_Tracking_For_Human_Computer_Interface> (дата обращения 22.04.2023).
16. J. Huang, H. Wechsler. Eye Detection Using Optimal Wavelet Packets and Radial Basis Functions (RBFs). – 1999. – URL: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.39.418&rep=rep1&type=pdf> (дата обращения 24.04.2023).
17. Kazemi V., Sullivan J. One Millisecond Face Alignment with an Ensemble of Regression Trees. – 2014. – URL: <https://www.researchgate.net/publication/264419855_One_Millisecond_Face_Alignment_with_an_Ensemble_of_Regression_Trees> (дата обращения 25.04.2023).
18. Friedman J. Stochastic Gradient Boosting. – 1999. – URL: <https://statweb.stanford.edu/~jhf/ftp/stobst.pdf> (дата обращения 10.04.2023).
19. iBUG 300-W [Электронный ресурс]. – URL: <https://ibug.doc.ic.ac.uk/resources/facial-point-annotations/> (дата обращения 13.04.2023).
20. Real-Time Face Detection and Recognition with SVM and HOG Features [Электронный ресурс]. – URL: <https://www.eeweb.com/profile/rajeevthaware/articles/real-time-face-detection-and-recognition-with-svm-and-hog-features> (дата обращения 5.04.2023).
21. Нейронная сеть [Электронный ресурс]. – URL: <https://ru.wikipedia.org/wiki/Нейронная_сеть> (дата обращения 7.04.2023).
22. Искусственный нейрон [Электронный ресурс]. – URL: <https://ru.wikipedia.org/wiki/Искусственый_нейрон> (дата обращения 7.04.2023).
23. Tian Y., Cohn J. Eye-State Action Unit Detection by Gabor Wavelets. – 2000. – URL: <https://www.researchgate.net/publication/221052178_Eye-State_Action_Unit_Detection_by_Gabor_Wavelets> (дата обращения 15.04.2023).
24. T. Kanade, J. Cohn, and Y. Tian. Comprehensive database for facial expression analysis. – 2000. – URL: <https://ieeexplore.ieee.org/document/840611> (дата обращения 15.04.2023).
25. Soukupova T., Cech J. Real Time Eye Blink Detection using Facial Landmarks. – 2000. – URL: <http://vision.fe.uni-lj.si/cvww2016/proceedings/papers/05.pdf> (дата обращения 15.04.2023).
26. C++ reference [Электронный ресурс]. – URL: <http://en.cppreference.com/w/cpp> (дата обращения 16.04.2023).
27. Python docs [Электронный ресурс]. – URL: <https://www.python.org/doc/> (дата обращения 16.04.2023).
28. C# reference [Электронный ресурс]. – URL: <https://msdn.microsoft.com/en-us/library/618ayhy6.aspx> (дата обращения 16.04.2023).
29. JavaScript [Электронный ресурс]. – URL: <https://www.javascript.com/> (дата обращения 16.04.2023).
30. R [Электронный ресурс]. – URL: <https://www.r-project.org/> (дата обращения 16.04.2023).
31. Java [Электронный ресурс]. – URL: <https://www.java.com/ru/> (дата обращения 16.04.2023).
32. PyCharm [Электронный ресурс]. – URL: <https://www.jetbrains.com/pycharm/> (дата обращения 17.04.2023).
33. Sublime Text [Электронный ресурс]. – URL: <https://www.sublimetext.com/> (дата обращения 17.04.2023).
34. VS Code [Электронный ресурс]. – URL: <https://code.visualstudio.com/> (дата обращения 17.04.2023).
35. Jupiter Notebook [Электронный ресурс]. – URL: <https://jupyter.org/> (дата обращения 17.04.2023).
36. Atom [Электронный ресурс]. – URL: <https://atom.io/> (дата обращения 17.04.2023).
37. Spyder [Электронный ресурс]. – URL: <https://www.spyder-ide.org/> (дата обращения 17.04.2023).
38. Visual Studio [Электронный ресурс]. – URL: <https://visualstudio.microsoft.com/ru/> (дата обращения 17.04.2023).
39. [Yingyu Ji](https://www.spiedigitallibrary.org/profile/notfound?author=Yingyu_Ji), [Shigang Wang](https://www.spiedigitallibrary.org/profile/Shigang.Wang-4199494), [Yang Lu](https://www.spiedigitallibrary.org/profile/Yang.Lu-3774391), [Jian Wei](https://www.spiedigitallibrary.org/profile/Jian.Wei-4118536), [Yan Zhao](https://www.spiedigitallibrary.org/profile/notfound?author=Yan_Zhao).Eye and mouth state detection algorithm based on contour feature extraction. – 2018. – URL: <https://www.spiedigitallibrary.org/journals/Journal-of-Electronic-Imaging/volume-27/issue-5/051205/Eye-and-mouth-state-detection-algorithm-based-on-contour-feature/10.1117/1.JEI.27.5.051205.full> (дата обращения 25.04.2023).
40. [Kaipeng Zhang,](https://ieeexplore.ieee.org/author/37085861445)[Zhanpeng Zhang,](https://ieeexplore.ieee.org/author/37085611668) [Zhifeng Li,](https://ieeexplore.ieee.org/author/37279080100)[Yu Qiao](https://ieeexplore.ieee.org/author/37401299900). Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks. – 2016. – URL: <https://kpzhang93.github.io/MTCNN_face_detection_alignment/paper/spl.pdf> (дата обращения 26.04.2023).
41. Liu W., Anguelov D., Erhan D., Szegedy C., Reed S., Fu C.Y., Berg A.. SSD: Single Shot MultiBox Detector. – 2016. – URL: <https://arxiv.org/abs/1512.02325> (дата обращения 07.05.2023).
42. Ren Sh., He K., Girshick R., Sun J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks – 2015. – URL: <https://arxiv.org/abs/1506.01497> (дата обращения 16.04.2023).
43. Javad M. Sh., Chywl B., Li F., Wong A. Fast YOLO: A Fast You Only Look Once System for Real-time Embedded Object Detection in Video. – 2017. – URL: <https://arxiv.org/abs/1709.05943> (дата обращения 17.04.2023).
44. Tensorflow [Электронный ресурс]. – URL: <https://www.tensorflow.org/> (дата обращения 21.05.2023).
45. Tensorboard [Электронный ресурс]. – URL: <https://www.tensorflow.org/tensorboard> (дата обращения 21.05.2023).
46. Keras [Электронный ресурс]. – URL: <https://keras.io/> (дата обращения 22.05.2023).
47. Numpy [Электронный ресурс]. – URL: <https://www.numpy.org> (дата обращения 22.05.2023).
48. OpenCV [Электронный ресурс]. – URL: <https://opencv.org/> (дата обращения 22.05.2023).
49. Deep Neural Networks module [Электронный ресурс]. – URL: <https://docs.opencv.org/master/d6/d0f/group__dnn.html> (дата обращения 23.05.2023).
50. Tengine [Электронный ресурс]. – URL: <https://github.com/OAID/Tengine> (дата обращения 23.05.2023).
51. Neon [Электронный ресурс]. – URL: <https://developer.arm.com/architectures/instruction-sets/simd-isas/neon> (дата обращения 23.05.2023).
52. Dlib [Электронный ресурс]. – URL: <http://dlib.net/> (дата обращения 24.05.2023).
53. The Python Debugger [Электронный ресурс]. – URL: <https://docs.python.org/3/library/pdb.html> (дата обращения 26.05.2023).
54. Firefly-RK3399 [Электронный ресурс]. – URL: <https://en.t-firefly.com/product/rk3399.html> (дата обращения 28.05.2023).
55. MRL Eye Dataset [Электронный ресурс]. – URL: <http://mrl.cs.vsb.cz/eyedataset> (дата обращения 28.05.2023).

# ПРИЛОЖЕНИЕ 1. Программный код

Исходный код программ

main.py // определения засыпания на основе USB/RTSP видеопотока

from scipy.spatial import distance as dist

import time

import dlib

import cv2

import imutils

from mtcnn import MTCNN

import logging

import datetime

def getEye(landmarks\_eye):

'''Getting centered coordinates of eye

Args:

landmarks\_eye: facial landmarks describing the eye area

Returns:

array: the array contains two int variables(x,y) describing the lower left corner of the eye

'''

ans = []

L = dist.euclidean(landmarks\_eye[0], landmarks\_eye[3])

A = dist.euclidean(landmarks\_eye[1], landmarks\_eye[5])

B = dist.euclidean(landmarks\_eye[2], landmarks\_eye[4])

which = False

if (A > B):

which = True

K = A

else:

K = B

dx = (40.0 - L) / 2.0

dy = (40.0 - K) / 2.0

ans.append(int(landmarks\_eye[0][0] - dx))

if which:

ans.append(int(landmarks\_eye[1][1] - dy))

else:

ans.append(int(landmarks\_eye[2][1] - dy))

return ans

def detectFace(detector, frame):

''' Detects faces on image

Args:

detector: loaded MTCNN model for face detection

frame (array): image to detect faces

Returns:

box (array): vector of x,y,w,h with max confidence

bool: True if process was successfull

time: algorithm running time

'''

success = True

try:

img = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

t = time.time()

boxes = detector.detect\_faces(img)

t = time.time() - t

confidence = 0

best\_box = []

for box in boxes:

if box['confidence'] > confidence:

confidence = box['confidence']

best\_box = box['box']

return best\_box, success, t

except Exception as e:

print("Exception in detectFace!")

logging.WARNING('Exception in detectFace!',exc\_info=True)

success = False

return [],success, t

def eyeStateRecongition(net, image):

''' Classifies of eye state : open or close

Args:

net: loaded model for eye classification

image (array): image of eye

Returns:

int : 1 if eye is open or 0 if eye is close

bool: True if process was successfull

'''

success = True

try:

net.setInput(cv2.dnn.blobFromImage(image, size=(40, 40), swapRB=False, crop=False))

networkOutput = net.forward()

if networkOutput[0][0] > networkOutput[0][1]:

return 0, success

else:

return 1, success

except Exception as e:

print("Exception in eyeStateRecongition!")

logging.WARNING('Exception in eyeStateRecongition!',exc\_info=True)

success = False

return 0, success

def tracking(box, increments):

''' Extends the area of the face

Args:

box (array): 4 face coordinates:x1, y1, x2, y2

increments (array): vector of increment values for each coordinate

Returns:

array: extended face area

'''

dx1, dx2, dy1, dy2 = increments

if len(box) < 1:

return box

else:

new\_box = box

new\_box[0] -= dx1

new\_box[1] -= dy1

new\_box[2] += dx2

new\_box[3] += dy2

for i in range(0, 4):

if (new\_box[i] < 0):

new\_box[i] = 0

if (i % 2 == 0):

if (new\_box[i] >= 640):

new\_box[i] = 640

else:

if (new\_box[i] >= 480):

new\_box[i] = 480

return new\_box

def crop(image, box):

''' Crops an image area

Args:

image (array): input image

box (array): region of interest (x1,y1,x2,y2)

Returns:

image (array): cut image

'''

return image[box[1]:box[3], box[0]:box[2]]

def sum(new\_box, oldbox):

''' Converts coordinates from one system to another

Args:

new\_box (array): face coordinates in the new system

oldbox (array): previous face coordinate value

Returns:

'''

if (new\_box == [] or oldbox == []):

return new\_box

new\_box[0] += oldbox[0]

new\_box[2] += oldbox[0]

new\_box[1] += oldbox[1]

new\_box[3] += oldbox[1]

def setlabel(frame, text, position):

''' Set text on frame

Args:

frame (array): it is the image on which text is to be drawn

text(string): text string to be drawn

position(array): it is the coordinates of the bottom-left corner of the text string in the image.

The coordinates are represented as tuples of two values (X coordinate value, Y coordinate value).

Returns:

image (array): image with draw text

'''

cv2.putText(frame, str(text), position, cv2.FONT\_HERSHEY\_COMPLEX\_SMALL, 1, (255, 255, 255), 1, cv2.LINE\_AA)

return frame

def drawrect(frame, rect):

''' Draw a rectangle on any image

Args:

frame (array): it is the image on which text is to be drawn

rect(array): array contains starting coordinates of rectangle and the ending coordinates of rectangle

Returns:

image (array): image with drawn rectangle

'''

cv2.rectangle(frame, (rect[0], rect[1]), (rect[2], rect[3]), (0, 255, 0), 4)

return frame

def get\_settings():

'''Read settingы from json file

Args:

Returns:

data(dict): dictionary with values from json

bool: True if process was successfull

'''

success = True

try:

import json

json\_file\_path = "settings.json"

with open(json\_file\_path, 'r') as j:

data = json.loads(j.read())

return data,success

except BaseException:

success = False

logging.critical('Settings were not read!')

print("Settings were not read!")

return [],success

def initialize\_model(settings):

''' Initializes the shape predictor, face detector, and eye classifier

Args:

data(dict): dictionary with settings

Returns:

face\_detector (object): MTCNN

shape\_predictor(object): dlib shape predictor

eye\_state\_detector(object): CNN keras model

'''

success = True

try:

face\_detector = MTCNN(steps\_threshold=settings['mtcnn']['steps\_threshold'])

shape\_predictor = dlib.shape\_predictor(settings['shape\_predictor']['path'])

eye\_state\_detector = cv2.dnn.readNetFromTensorflow(settings['cnn\_eyes']['path'])

return face\_detector, shape\_predictor, eye\_state\_detector, success

except Exception as e:

success = False

logging.critical('Models were not initialize!',exc\_info=True)

print('Models were not initialize!')

return success,success,success,success

def draw\_info(frame,mtcnnTime,faceBox,leftEyeBox,rightEyeBox,eyesTime,fps\_time):

''' Adds information about the operation of algorithms

Args:

frame(array): input image

mtcnnTime (float): MTCNN algorithm running time

faceBox(array): rect of face

leftEyeBox(array): rect of left eye

rightEyeBox(array): rect of right eye

eyesTime(float): CNN eye keras algorithm running time

fps\_time(float): processing time of one frame

Returns:

frame(array): image with draw info

'''

frame = drawrect(frame,faceBox)

frame = drawrect(frame,leftEyeBox)

frame = drawrect(frame,rightEyeBox)

frame = setlabel(frame,"FPS: {:.1f}".format(1.0 / fps\_time),(20, 60))

frame = setlabel(frame,"MTCNN:{:.5f}".format(mtcnnTime),(20, 80))

frame = setlabel(frame,"CNN\_EYES:{:.5f}".format(eyesTime),(20, 100))

return frame

def get\_eyes\_landmarks(frame,predictor, facebox):

''' Getting the 6 landmarks for each eye

Args:

frame(array): input image with face

predictor(object): dlib shape predictor

facebox(dlib.rectangle): face rectangle

Returns:

leftEye(array): 6 coordinates of the left eye consisting of two values(x and y)

rightEye(array): 6 coordinates of the right eye consisting of two values(x and y)

'''

shape = predictor(frame, facebox)

shape = face\_utils.shape\_to\_np(shape)

(lStart, lEnd) = face\_utils.FACIAL\_LANDMARKS\_IDXS["left\_eye"]

(rStart, rEnd) = face\_utils.FACIAL\_LANDMARKS\_IDXS["right\_eye"]

leftEye = shape[lStart:lEnd]

rightEye = shape[rStart:rEnd]

return leftEye,rightEye

def getEyesState(frame,leftEyeCoord,rightEyeCoord,eyeStateDetector):

''' Cuts and classifies eyes

Args:

frame(array): input image

leftEyeCoord(object): dlib shape predictor

rightEyeCoord(dlib.rectangle): face rectangle

eyeStateDetector(object): CNN keras model

Returns:

eyesState(int): 0 if eyes are close or 1 otherwise

bool: True if process was successfull

float: algorithm running time

'''

cropLeftEye = frame[leftEyeCoord[1]:leftEyeCoord[1] + 40, leftEyeCoord[0]:leftEyeCoord[0] + 40]

cropRightEye = frame[rightEyeCoord[1]:rightEyeCoord[1] + 40, rightEyeCoord[0]:rightEyeCoord[0] + 40]

eyesState = 0

works\_time = time.time()

leftEyeState,leftSuccess = eyeStateRecongition(eyeStateDetector,cropLeftEye)

rightEyeState,rightSuccess = eyeStateRecongition(eyeStateDetector,cropRightEye)

works\_time = time.time() - works\_time

success = leftSuccess \* rightSuccess

eyesState = leftEyeState + rightEyeState

eyesState = min(eyesState,1)

logging.info('Eyes state is %d',eyesState)

return eyesState, success , works\_time

def event\_happened(frames,settings):

''' Records a sequence of image to a video file

Args:

frames(array of arrays): sequence of image

settings(dict): dictionary containing fps video source value

Returns:

'''

logging.info('Event happened')

print("Event happened")

fourcc = cv2.VideoWriter\_fourcc(\*'XVID') # cv2.VideoWriter\_fourcc(\*'MPEG')

save\_path = "event\_"+datetime.datetime.now().strftime("%d%m%Y\_%H\_%M\_%S")+".avi"

video = cv2.VideoWriter(filename=save\_path, fourcc=fourcc, fps=settings['video\_source']['fps'], frameSize=(frames[0].shape[1],frames[0].shape[0]))

for frame in frames:

video.write(frame)

video.release()

def is\_sleep(eyes\_state,settings):

''' Checks the conditions for a falling asleep event

Args:

eyes\_state(array): sequence of image

settings(dict): dictionary containing value ratio of closed eyes to total

Returns:

bool: True if event was happened

'''

if(not eyes\_state):

return False

s = 0

for i in eyes\_state:

s+=i

#print((len(eyes\_state)-s)/len(eyes\_state))

if (len(eyes\_state)-s)/len(eyes\_state) >= settings['sleep']['percent']:

return True

else:

return False

from imutils.video import VideoStream

from imutils import face\_utils

from imutils.video import WebcamVideoStream

from threading import Thread

import numpy as np

import imutils

import time

import dlib

import logging

import cv2

import sys

def main():

# logging settings

logging.basicConfig(format = u'%(levelname)-8s [%(asctime)s] %(message)s', level = logging.DEBUG, filename = u'application.log')

# read settings from json

settings, success = get\_settings()

#if it was not possible to read the settings, then shut down the module

if(not success):

sys.exit(-1)

#init models

faceDetector, shapePredictor, eyeStateDetector, success = initialize\_model(settings)

#settings for tracking algo

boxIncrements = settings['tracking']['increment']

#if it was not possible to init models, then shut down the module

if(not success):

sys.exit(-1)

#input stream definition

if (settings['video\_source']['usb']):

vs = WebcamVideoStream(src=settings['video\_source']['link']).start()

if (settings['video\_source']['rtsp']):

vs = VideoStream(settings['video\_source']['link']).start()

all\_time = 0

frame\_counter = 0

frames\_buffer = []

eyes\_state\_buffer = []

facebox = []

while True:

fps\_time = time.time()

frame\_counter += 1

# reading image

frame = vs.read()

# adding image into buffer

frames\_buffer.append(frame)

#if there is no information about the previous position of the face

if not facebox:

#face detecting full image

facebox, success, mtcnnTime = detectFace(faceDetector,frame)

#cast box(x1,y1,w,h) to box(x1,y1,x2,y2)

if facebox:

facebox[2]+=facebox[0]

facebox[3]+=facebox[1]

else:

#getting an expanded area of the face

estimatedArea = tracking(facebox,boxIncrements)

#clipping from image area

cropFrame = crop(frame,estimatedArea)

#face detecting crop image

estimatedBox, success, mtcnnTime = detectFace(faceDetector,cropFrame)

#if we detected face

if estimatedBox:

#cast box(x1,y1,w,h) to box(x1,y1,x2,y2)

estimatedBox[2]+=estimatedBox[0]

estimatedBox[3]+=estimatedBox[1]

#cast old coord to new coord

sum(estimatedBox,facebox)

facebox = estimatedBox

#if the detection was successful

if facebox and success:

#cast array to dlib.rectangle

faceRect = dlib.rectangle(facebox[0], facebox[1], facebox[2], facebox[3])

#getting 6 landmarks of each eye

leftEye,rightEye = get\_eyes\_landmarks(frame,shapePredictor,faceRect)

#getting low left corner of each eye

leftEyeCoord = getEye(leftEye)

rightEyeCoord = getEye(rightEye)

#cast array(x,y) to array(x,y,x+40,y+40) for each eye

leftEyeBox = [leftEyeCoord[0],leftEyeCoord[1],leftEyeCoord[0]+40,leftEyeCoord[1]+40]

rightEyeBox = [rightEyeCoord[0],rightEyeCoord[1],rightEyeCoord[0]+40,rightEyeCoord[1]+40]

#eye classification

eyesState, success, eyesTime = getEyesState(frame,leftEyeCoord,rightEyeCoord,eyeStateDetector)

# adding prediction inti buffer

eyes\_state\_buffer.append(eyesState)

fps\_time = time.time() - fps\_time

all\_time +=fps\_time

#adding information on frame

frame = draw\_info(frame,mtcnnTime,facebox,leftEyeBox,rightEyeBox,eyesTime,all\_time/frame\_counter)

#show image

cv2.imshow('image', frame)

# wait Esc to be pushed

if cv2.waitKey(1) == 27:

break

#if there are enough predictions of eye condition in the buffer

if len(eyes\_state\_buffer) == settings['sleep']['time'] \* settings['video\_source']['fps']:

#checking that the driver is sleeping

if is\_sleep(eyes\_state\_buffer,settings):

#if sleeping, then record a video

event\_happened(frames\_buffer,settings)

#buffer flushing

frames\_buffer.clear()

eyes\_state\_buffer.clear()

else:

#show image

cv2.imshow('image', frame)

# wait Esc to be pushed

if cv2.waitKey(1) == 27:

break

if \_\_name\_\_ == "\_\_main\_\_":

main()

cv2.