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

ФАКУЛЬТЕТ «**Информатика и системы управления**»

КАФЕДРА «**Программное обеспечение ЭВМ и информационные технологии**»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

НА ТЕМУ:

«Метод оценки безопасности водителя с использованием глубоких нейронных сетей»

Студент группы **ИУ7-83Б**	_____	**Неумоин Д.Ю**
	(Подпись, дата)	(И.О. Фамилия)
Руководитель ВКР	_____	**Кузнецова О.В**
	(Подпись, дата)	(И.О. Фамилия)
Нормоконтролер	_____	_____
	(Подпись, дата)	(И.О. Фамилия)

***2024** г.*

РЕФЕРАТ

Расчетно-пояснительная записка 94с., 27 рис., 1 табл., 18 ист., 4 прил.

Ключевые слова: глубокие нейронные сети, классификация изображений, оценка безопасности.

В работе представлена разработка метода оценки безопасности водителя с использование глубоких нейронных сетей.

В разделе 1 рассмотрена задача классификации изображений, рассмотрены методы и технологии ее решения с использованием глубоких нейронных сетей. Проведен сравнительный анализ методов.

В разделе 2 спроектирован метод оценки безопасности водителя с использование глубоких нейронных сетей. Представлены схемы алгоритмов, разработана архитектура программного комплекса.

В разделе 3 разработан программный комплекс и описаны средства его реализации, приведены примеры его использования.

В разделе 4 проведено исследование зависимости точности модели от размера входных данных и времени работы метода от объема входных данных..

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

1) ДТП (дорожно-транспортное происшествие)
2) MLP (от англ. Multilayer perceptron) — многослойный персептрон.
3) CNN (от англ. Convolutional neural network) — сверточная нейронная сеть.

ВВЕДЕНИЕ

В последние годы такси стало полноценной частью общественного транспорта, особенно в мегаполисах. В 2021 году количество занятых на постоянной основе в отрасли таксоперевозок составило 575 тысяч человек. Опросы показывают, что для пассажиров в вопросе выбора такси приоритетна не только цена поездки, но и безопасность [1].

Согласно исследованиям[2] за 2017 год произошло 142000 ДТП, из которых 2917 с участием такси, при этом рост ДТП с участием такси составил порядка 17.5%.

Исходя из приведенных выше фактов, можно сделать вывод, что изучение поведения водителей и прогнозирование потенциального риска автомобильных аварий являются критическими аспектами обеспечения безопасности дорожного движения. В связи с этим возникает необходимость разработки эффективных методов оценки безопасности водителя.

Для оценки безопасности водителей можно воспользоваться компьютерными технологиями, такие технологии могут быть полезны как для сервисов такси, так и для сервисов каршеринга.

Цель работы – разработка метода оценки безопасности водителя с использованием глубоких нейронных сетей.

Для достижения поставленной цели требуется решить следующие задачи:

- описать термины предметной области и формализовать задачу оценки безопасности водителя;
- рассмотреть и сравнить методы классификации изображений;
- спроектировать метод оценки безопасности водителя на основе глубоких нейронных сетей;
- разработать программное обеспечение, реализующее данный метод;
- провести исследование зависимости точности модели от размера входных данных и времени работы метода от объема входных данных.

1 Аналитический раздел

1.1 Задача оценки безопасности водителя

Оценка безопасности водителя – это комплексная задача, нацеленная на анализ действий водителя. В последнее время большое внимание уделяется разработке интеллектуальных систем, способных автоматически оценивать уровень внимания водителя. Основным источником данных для таких систем часто выступают фото или видео, получаемые с камер из салона автомобиля направленных на водителя. Таким образом, основной частью задачи оценки безопасности водителя является задача классификации изображений, где основные классы – это различные состояния водителя, например, «смотрит на дорогу», «пьет воду», «смотрит в телефон» и так далее. После шага классификации всех изображений, можно вычислить оценку безопасности водителя проанализировав количество тех или иных классов.

1.2 Задача классификации изображений

Задачи классификации – это задачи, в которых объект должен быть отнесен к одному из n классов на основе индекса сходства его характеристик с каждым классом. Под классами понимается набор подобных объектов. Объекты считаются похожими на основе совпадающих характеристик, таких как цвет, форма, размер и т.д. Классы определяются на основе их уникальных меток. В частности, задача «распознавания действий человека за рулем автомобиля по фото» является задачей классификации изображений.

Основным алгоритмом для решения задачи классификации изображения являются глубокие нейронные сети, а именно сверточные нейронные сети.

1.3 Введение в сверточные нейронные сети

1.3.1 Нейрон

Биологическая нервная система представляет собой сеть, состоящую из множества нейронов. Аналогичным образом, нейроны также являются основным процессорным блоком искусственных нейронных сетей. Принцип работы заключается в том, что множество входных значений подвергаются математическому преобразованию для получения выходного значения (рисунок 1). Соотношение математического преобразования между входным сигналом и выходным значением представлено формулой:

$$f(b + \sum_{i=1}^{n}(x_i \times w_i), \tag{1}$$

где f – функция активации, например ReLU, Sigmoid, Tanh и другие.

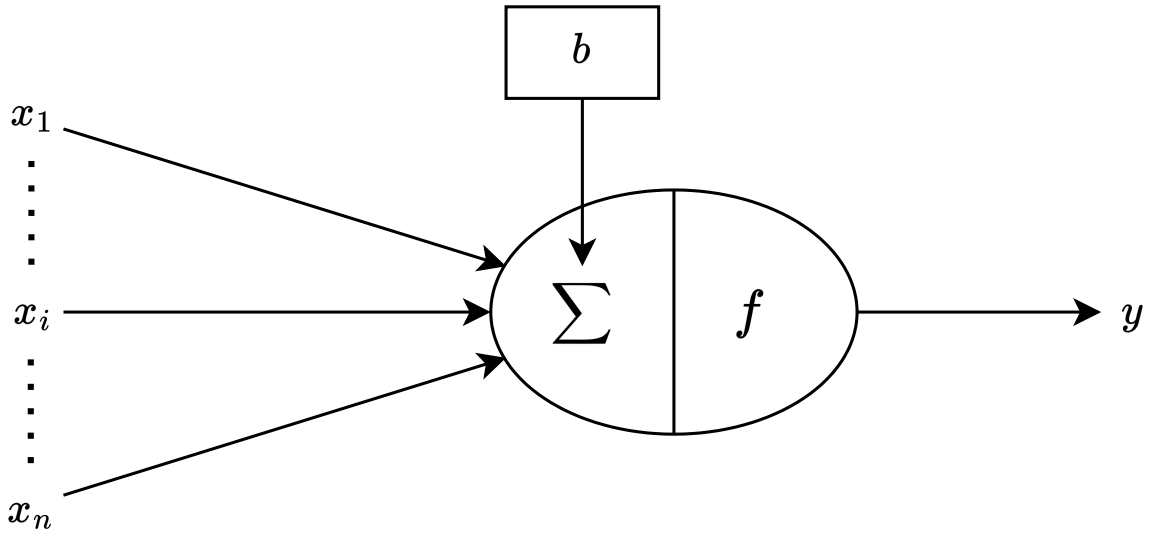

Рисунок 1 – Модель нейрона.

1.3.2 Многослойный персептрон

Многослойный персептрон состоит из входного слоя, скрытого слоя (одного или нескольких) и выходного слоя. Он содержит несколько базовых мо-

дульных нейронов, которые осуществляют передачу сигнала посредством послойной проводимости между нейронами. На рисунке 2 приведен пример структуры MLP. H – вектор выходного значения скрытого модуля $H = F(W_h X + B_h)$, Y – вектор выходного значения выходного модуля $Y = F(W_y H + B_y)$, где X – матрица входных значений, W_h и W_y матрица весов между слоями, B_h B_y – матрицы смещения.

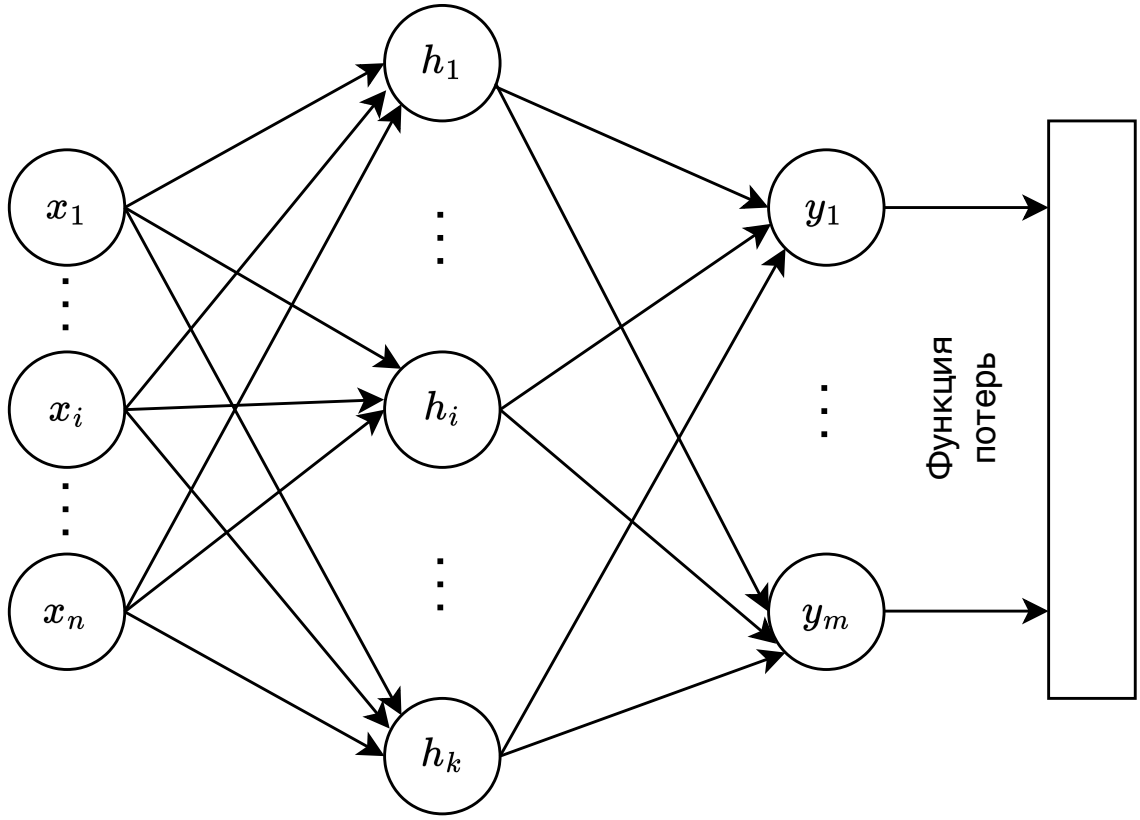

Рисунок 2 – Структура многослойного персептрона.

1.4 Архитектура сверточных нейронных сетей

Основная структура CNN включает сверточный слой, слой пулинга, слой нелинейной активации и полносвязный слой. Изображение сначала предварительно обрабатывается и подается на вход сети, где проходит через чередующиеся слои свертки и пулинга, после чего классифицируется полносвязным слоем.

CNN дополняют структуру MLP за счет введения сверточных и слоев пулинга, что дает преимущества в обработке изображений с большим коли-

чеством пикселей и в больших наборах данных, обеспечивая лучший размер модели и производительность [3]. Сверточный слой обладает свойствами локального рецептивного поля, сохраняя форму входного изображения и эффективно распознавая взаимосвязь между пикселями по длине и ширине за счет использования параметрического разделения и разреженной связи. Слой пулинга уменьшает вычислительную нагрузку и чувствительность к положению, способствуя инвариантности к перемещению, масштабированию и искажениям входных изображений.

1.4.1 Сверточный слой

Для сверточных нейронных сетей определенной глубины операция свертки нескольких сверточных слоев может извлекать различные характеристики входного сигнала. Нижний слой свертки обычно извлекает общие характеристики, такие как текстуры, линии и грани, в то время как более высокие слои извлекают более абстрактные признаки.

Сверточный слой имеет несколько сверточных ядер с обучаемыми параметрами. Это матрица, составленная из обучаемых весов, которые обычно имеют размеры 3×3, 5×5 и 7×7 с равной длиной, шириной и нечетным числом. Обычно сверточный слой принимает карты признаков (feature maps). Матрица весов ядра свертки соответствует локальной области карты объектов-соединений, и ядро свертки последовательно выполняет операции свертки над областью на карте [4].

Как правило, размеры входных карт $H \times W \times C$ (высотой H, шириной W и каналы C), каждое ядро свертки $K \times K \times C$ это число ядра свертки должна быть такой же, как число входных каналов. На рисунке 3 представлена схема процесса свертки входных карт объектов ($5 \times 5 \times 3$) и ядра свертки ($3 \times 3 \times 3$). Поток данных в сверточном слое можно приблизительно выразить как:

$$feature\_surface_{out} = f\left(\sum_{i=3}^{3} M_i * W_i + B\right), \quad (2)$$

где M_i представляет собой поверхность объекта входных карт объектов, W_i – матрица весов ядра свертки, M - матрица смещения, $f(\cdot)$ – нелинейная функция активации, а $feature\_surface_{out}$ выходная карта признаков.

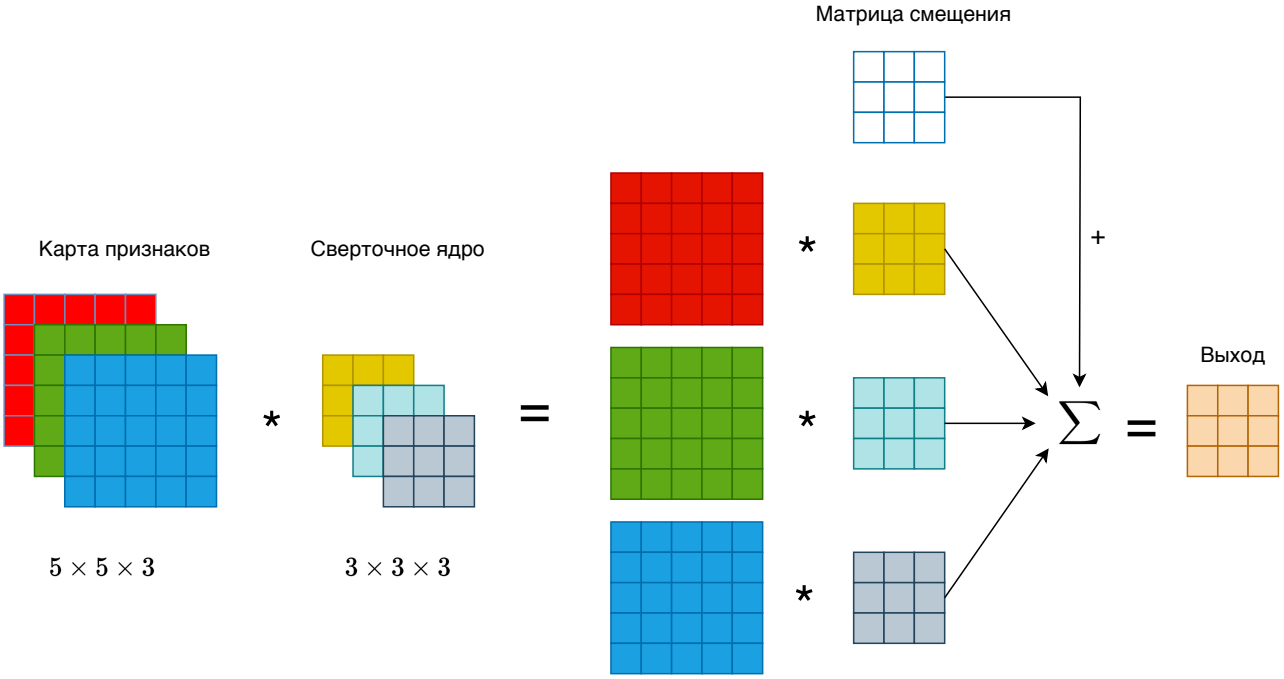

Рисунок 3 – Схематичная диаграмма процесса свертки.

Слой активации Скалярный результат каждой свёртки попадает на функцию активации, которая представляет собой некую нелинейную функцию. Слой активации обычно логически объединяют со слоем свёртки (считают, что функция активации встроена в слой свёртки).

Функция активации решает, следует активировать нейрон или нет. Это означает, что она будет решать, важен или нет вход нейрона в сеть в процессе прогнозирования. На данный момент в качестве функции активации используются ReLU (сокращение от англ. rectified linear unit) функции, такие как ReLU, Leaky ReLU, PReLU, RReLU и ELU [5].

1.4.2 Слой пулинга

Пулинг-слои обычно следуют за сверточными слоями. Основные причины использования пулинг-слоя:

- выполнение процесса субдискретизации и уменьшения размерности входного изображения для уменьшения количества соединений сверточного слоя, что в свою очередь снижает вычислительную нагрузку сети [6];
- обеспечение инвариантности к масштабированию, сдвигу и повороту входного изображения[7];
- обеспечение более устойчивой работы выходной карты признаков к искажениям и ошибкам отдельного нейрона [8].

Наиболее широко используемыми методами пулинга являются средний и максимальный пулинг [9].

На рисунке 4 показан процесс субдискретизации максимального и среднего пулинга.

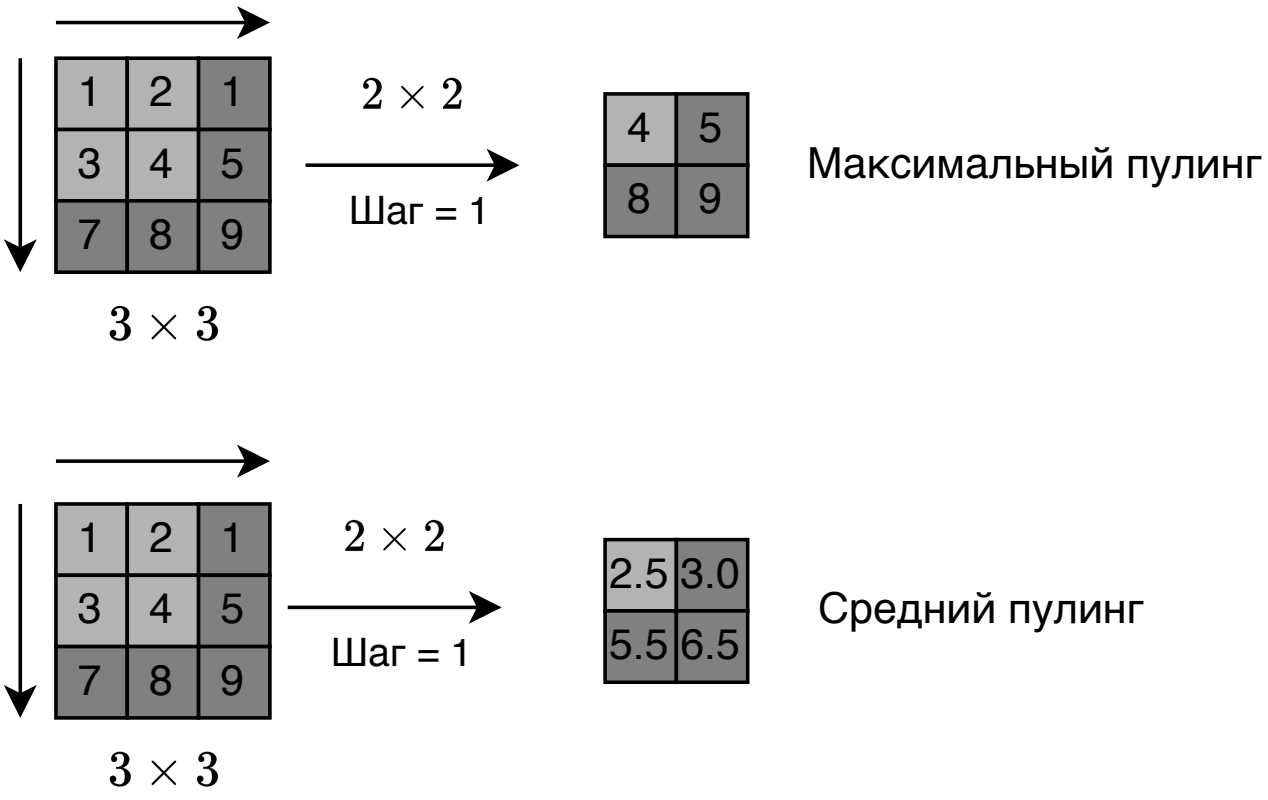

Рисунок 4 – Максимальный и средний пулинг.

1.4.3 Полносвязный слой

Полносвязный слой обычно находится после непрерывного сверточного слоя и слоя пулинга. Он интегрирует и классифицирует локальную информацию, извлеченную после свертки и пулинга и в конечном итоге выводит информацию о категории изображения.

Он содержит несколько скрытых слоев, которые извлекают высокоуровневые признаки из предыдущей сети в более сложной форме. Количество нейронов на выходном конце равно количеству категорий. Затем выходной вектор используется для определения принадлежности изображения к категории. Проще говоря, полносвязный слой действует как классификатор в сверточной нейронной сети.

Во время обучения выход сети обычно подвергается softmax регрессии[6] для нормализации вероятности перед функцией потерь полносвязного слоя. Па-

раметры слоя обновляются с использованием обратного распространения градиента.

1.4.4 Функция потерь

В дополнение к различным типам слоёв архитектуры сверточных нейронных сетей, которые были представлены в выше, окончательная классификация достигается с помощью выходного слоя, который обычно является последним слоем полносвязного слоя. Разные функции потерь оказывают влияние на производительность архитектуры CNN и применяются для различных задач, таких как классификация изображений, распознавание лиц и объектов.

1.5 Технологии классификации изображений

Для решения задачи классификации изображений используются такие модели сверточных сетей, как[10]:

- AlexNet;
- VGGNet;
- GoogleNet;
- ResNet.

1.5.1 AlexNet

AlexNet имеет восьмислойную структуру: первые пять слоев являются сверточными, а последние три полносвязными. Для выполнения обучения сети требуется изображение с разрешением 227×227 в качестве входных данных и информация о классификации, встроенная в изображение в качестве выходных данных.

Принцип работы AlexNet представлен на рисунке 5.

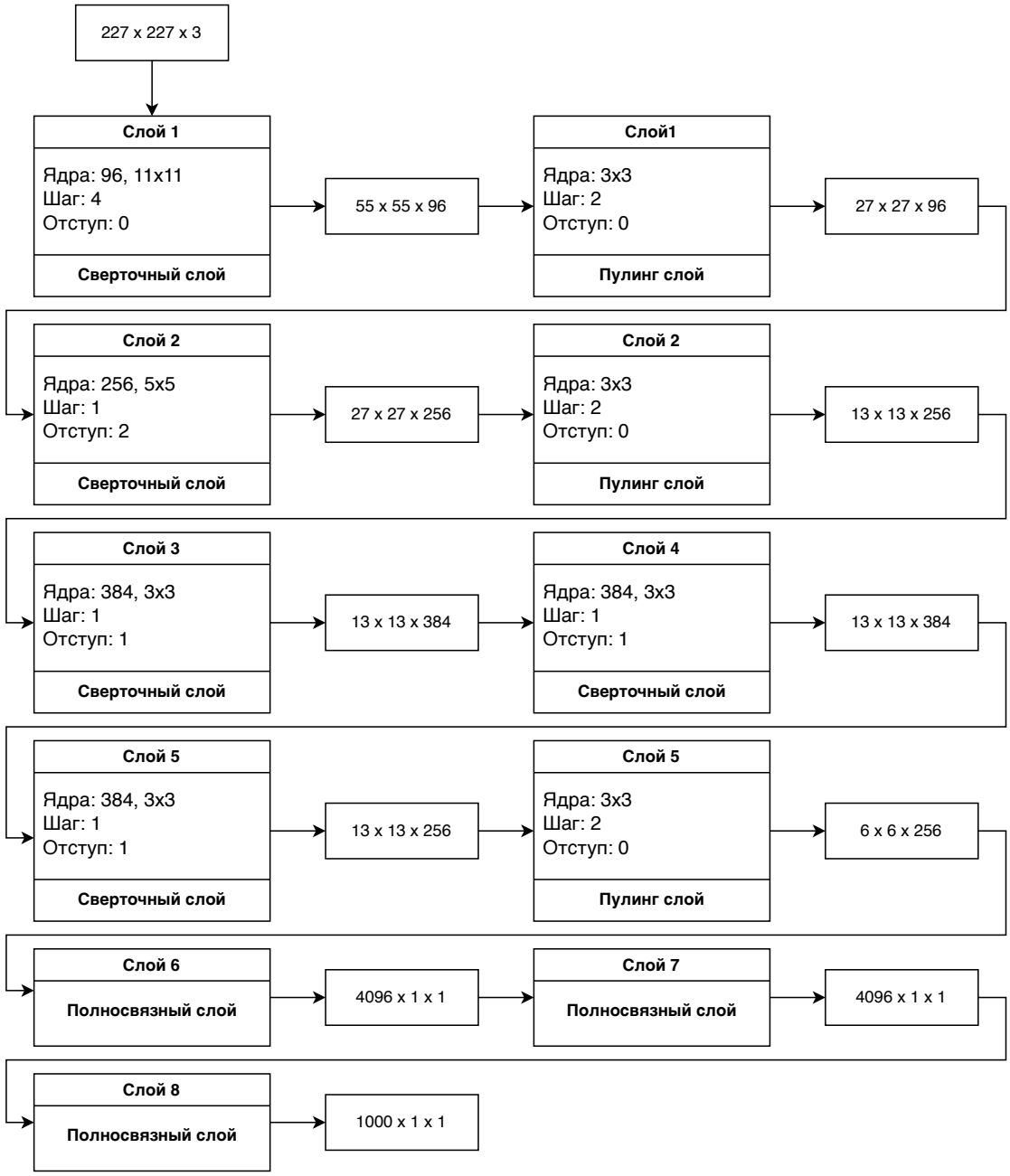

Рисунок 5 – Принцип работы AlexNet

1.5.2 VGGNet

Эта модель аналогична модели AlexNet и также использует структуру области сверточных слоев, за которой следует область полносвязных слоев. Правило компоновки модуля VGG заключается в последовательном использовании нескольких идентичных сверточных слоев, за которыми следует слой максимального пулинга, при этом сверточный слой сохраняет высоту и ширину входных данных неизменными, в то время как пулинг слой уменьшает их вдвое.

Сеть VGG имеет множество различных моделей, одной из которых является VGG-16, принцип работы которой изображен на рисунке 6. Она содержит 16 уровней весов, сеть последовательно соединяет пять блоков, и в конце подключаются два полносвязных слоя с 4096 и выходной слой с 1000 классами.

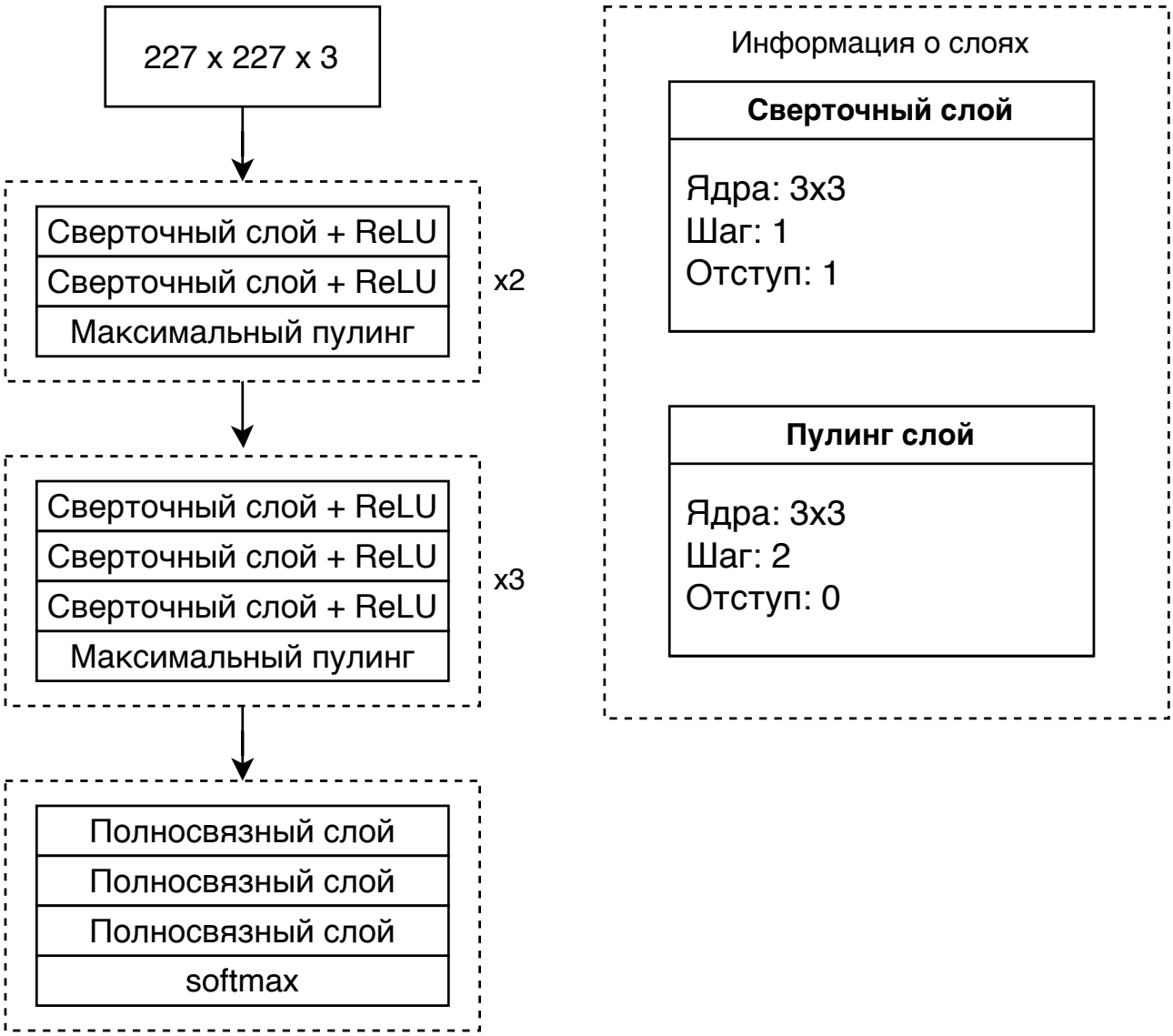

Рисунок 6 – Принцип работы VGG-16

1.5.3 GoogleNet

GoogleNet изменила структуру CNN, введя параллельное внутреннее соединение с помощью структуры $Inception$. Как показано на рисунке 7, когда данные обрабатываются с использованием $inception$, они должны одновременно проходить через четыре пути с различными сверточными ядрами; в итоге данные объединяются в новый слой сети. Структура $inception$ является самым значительным отличием GoogleNet от традиционных сетей CNN. Существует 4 разновидности структуры [11]: $inceptionV1$, $inceptionV2$, $inceptionV3$, $inceptionV4$, в данной работе рассматривается первая вариация.

Использование структуры $inception$ имеет два основных преимущества: во-первых, одновременное свертывание на нескольких масштабах позволяет извлекать характеристики на разных масштабах, что также приводит к большей точности в окончательном классификационном суждении; второе – использование свертки 1×1 для сокращения размерности позволяет уменьшить вычислительную сложность, и объем вычислений значительно сокращается, когда уменьшается количество признаков, после чего выполняется свертывание. Благодаря преимуществам $inception$, хотя у GoogleNet 22 слоя сети, что глубже, чем 8 слоев у AlexNet или 19 слоев у VGGNet, она может достичь гораздо большей точности, чем AlexNet, имея только 5 миллионов параметров (1/12 от параметров AlexNet и 1/25 от параметров VGG-16) [11].

На рисунке 8 представлен принцип работы GoogleNet, на рисунке присутствуют $Stem$ блок и блок вспомогательного классификатора.

$Stem$ блок является фундаментов модели, он основан на идее, что перед тем как данные будут переданы через серию $Inception$-блоков, первоначальное представление входных данных должно быть наиболее эффективно обработано для дальнейшего извлечения признаков.

Блок вспомогательного классификатора вводится в качестве дополнительного средства для улучшения обучающих характеристик глубоких нейронных

сетей.

Структуры $Stem$ блока и вспомогательного классификатора представлена на рисунке 9.

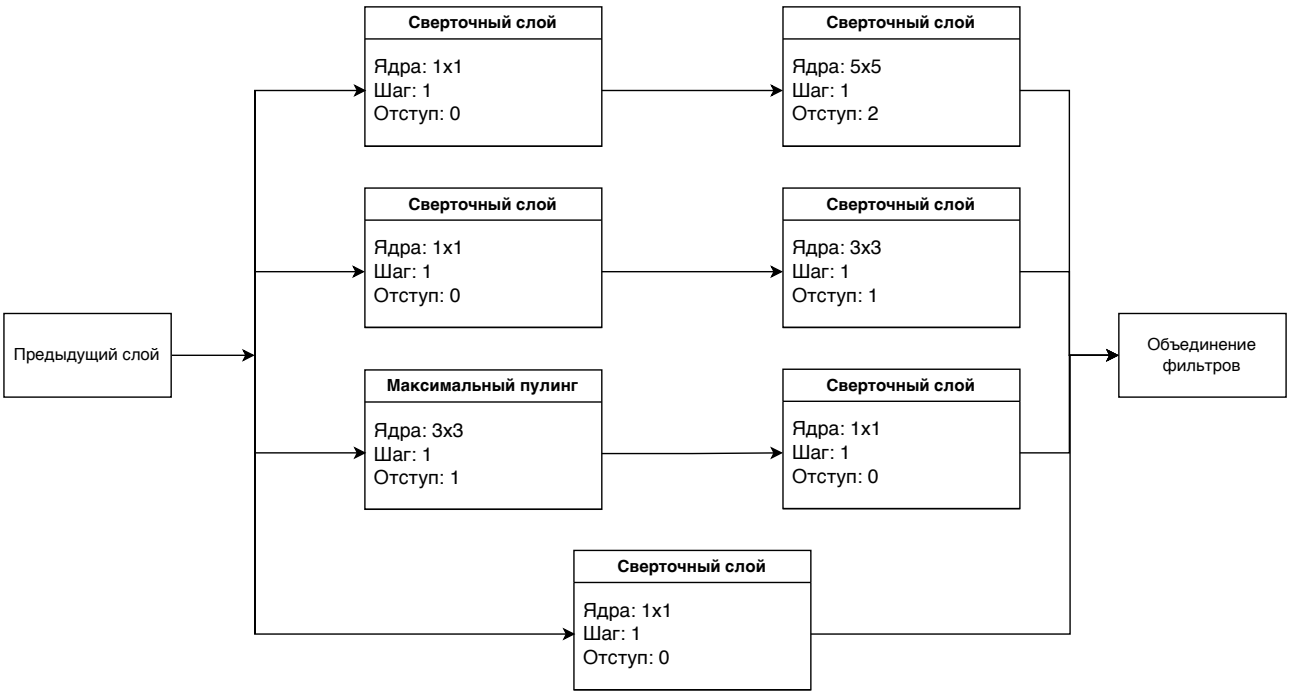

Рисунок 7 – Структура $inceptionV1$

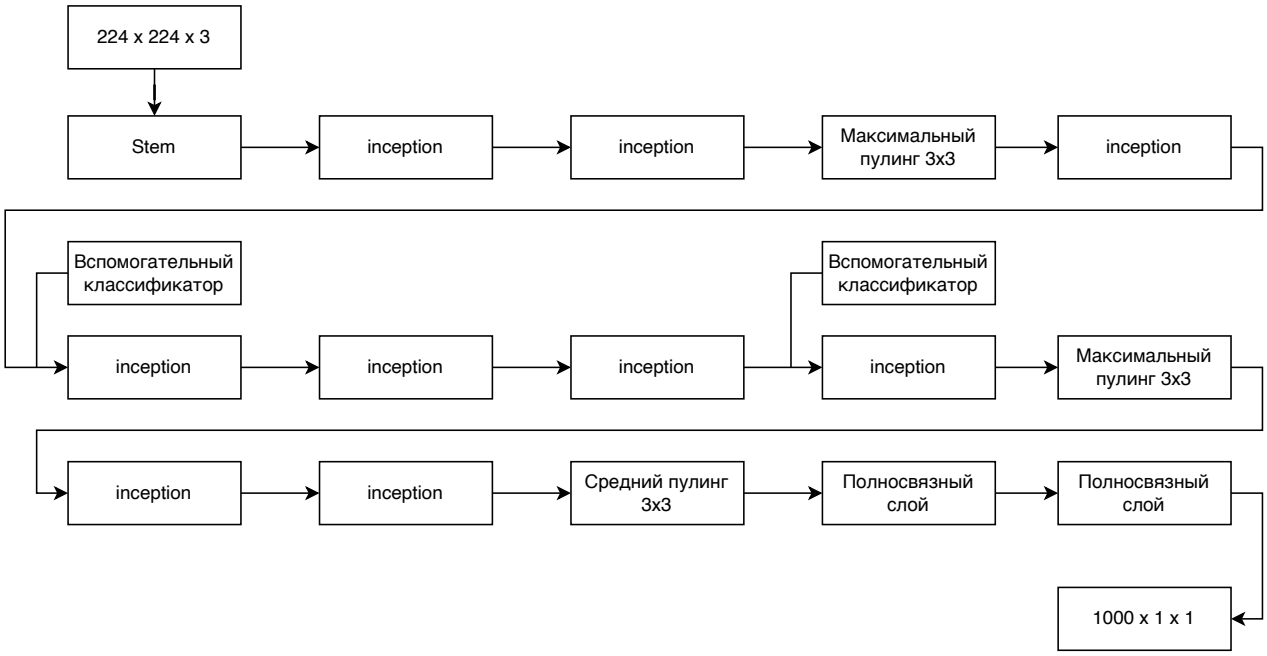

Рисунок 8 – Принцип работы GoogleNet

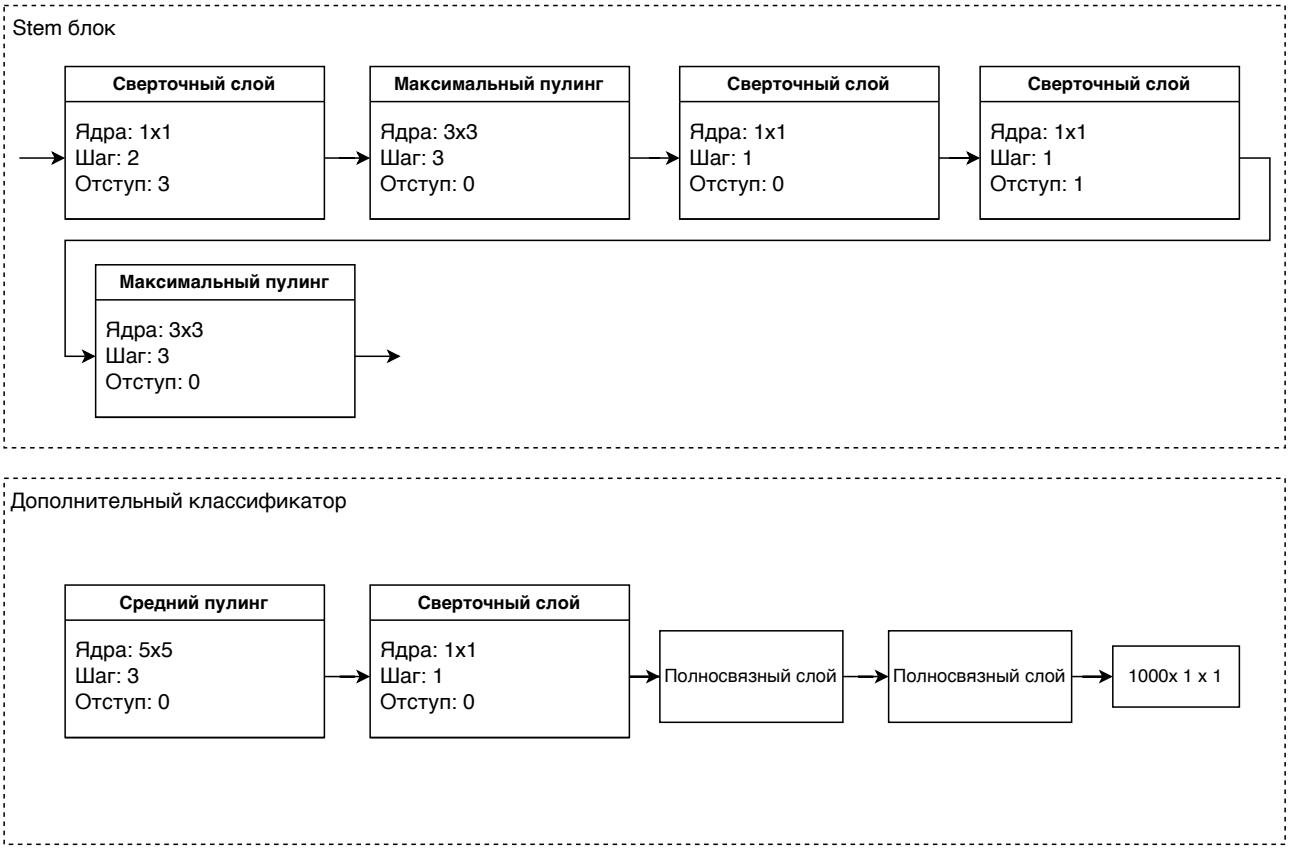

Рисунок 9 – Структура $Stem$ блока и вспомогательного классификатора

1.5.4 ResNet

ResNet (от англ. Residual Network) - это остаточная сеть. Считалось, что чем глубже сеть, тем больше информации можно получить, и тем устойчивее будут характеристики. Однако эксперименты показывают, что по мере углубления сети эффективность оптимизации ухудшается, а точность на тестовых и обучающих данных снижается. Это явление является результатом проблем расширения градиента и исчезновения градиента, вызванных углублением сети [12].

Для решения проблемы дегенерации в глубоких сетях можно искусственно заставить определенные слои нейронной сети пропустить соединение нейронов в следующем слое и соединить их в чередующихся слоях, ослабляя сильную связь между каждым слоем. Такие нейронные сети называются остаточными сетями (ResNet).

Как показано на рисунке 10, входные данные x складываются с оригинальными данными x, и после обработки сверточным слоем получается $F(X)$, в сравнении с традиционной структурой CNN. Впоследствии можно одновременно сохранить действительную информацию двух слоев, и одновременно снизить потерю информации, вызванную слишком большим количеством слоев. Модель CNN, построенная с использованием этой остаточной сети, может содержать 100 и более слоев. Модель, рассматриваемая в данной работе – ResNet101, которая имеет 101 слой и 44 миллиона параметров.

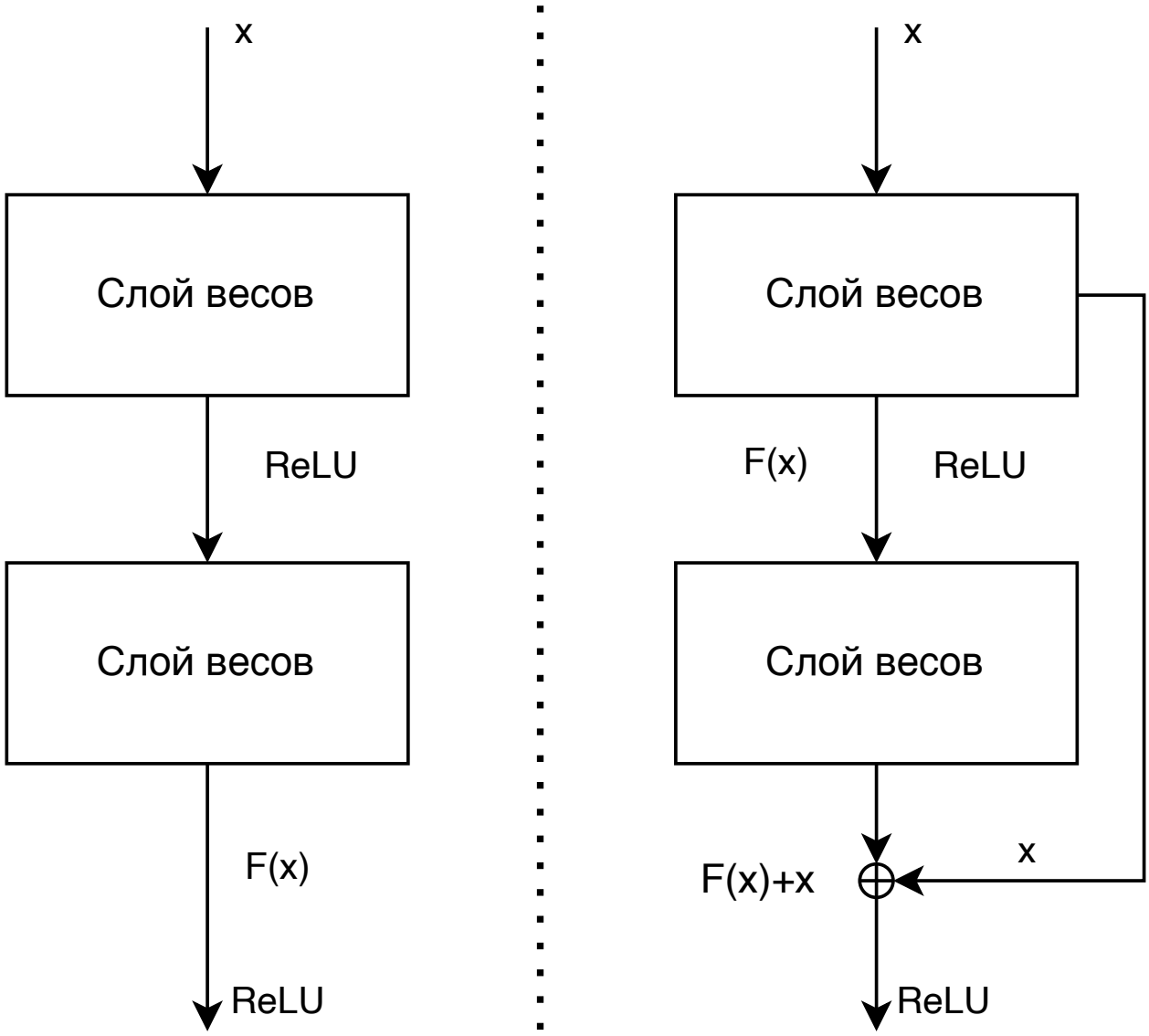

Рисунок 10 – Сравнение обычного обучения CNN (слева) и остаточного обучения (справа)

1.5.5 Сравнение рассмотренных методов

Сравнительная характеристика рассмотренных методов представлена в таблице 1.

Таблица 1 – Сравнение методов распознавания образов

Критерий	**AlexNet**	**VGG-16**	**GoogleNet**	**ResNet101**
Время вычисления (сек)	56	212	77	197
Точность на тестовых данных (%)	86.67	85.19	94.81	90.37
FLOPs	727	16000	2000	7600

FLOPs — количество операций с плавающей точкой (от англ. floating point operations), отражающее сложность вычислений.

Значения критериев для каждого из методов взяты из исследования «A Comparative Study of Different CNN Models and Transfer Learning Effect for Underwater Object Classification in Side-Scan Sonar Images» [10].

Из представленной таблицы можно сделать вывод: GoogleNet является наиболее подходящей моделью, так как является наиболее точной, к тому же по времени вычисления и FLOPs она незначительно уступает AlexNet, в сравнении с VGG и ResNet имеет лучшие показатели по всем критериям.

1.6 Формализация постановки задачи

На основе рассуждений, представленных в данном разделе, можно сделать вывод, что для оценки безопасности водителя требуется:

- видео или множество фотографий из салона автомобиля;
- обученная модель нейронной сети;
- значения весов для каждого действия водителя;
- алгоритм вычисления коэффициента безопасности водителя на основе количества полученных действий и весов.

Формально данная задача может быть описана с помощью IDEF0-диаграммы нулевого уровня, представленной на рисунке 11.

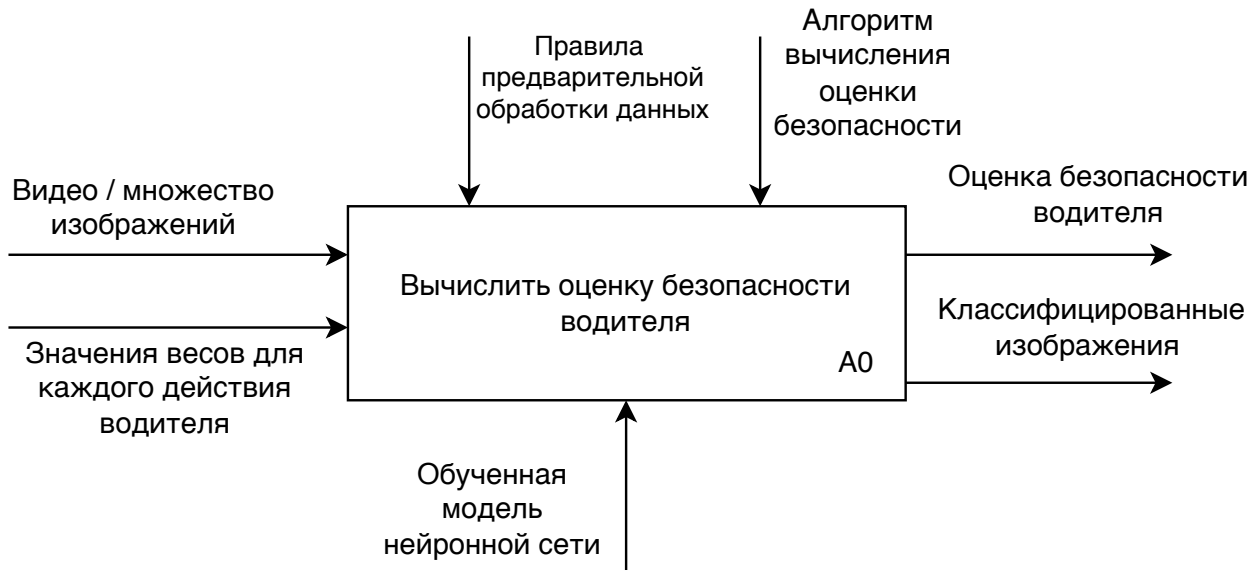

Рисунок 11 – Постановка задачи в виде IDEF0-диаграммы нулевого уровня

Вывод

В данном разделе была рассмотрена задача оценки безопасности водителя, которая может быть сведена к задаче классификации изображений. Также были описаны сверточные нейронные сети и их архитектура, рассмотрены технологии классификации изображений на основе сверточных нейронных сетей.

В качестве базовой технологии для разрабатываемого метода была выбрана модель сверточной нейронной сети GoogleNet, так как имеет наилучшую точность, в сравнении с другими рассмотренными методами, а также сравнительно небольшую сложность и время вычисления.

Также была формализована задача с помощью IDEF0-диаграммы нулевого уровня.

2 Конструкторский раздел

2.1 Требования к разрабатываемому методу

Метод оценки безопасности водителя (далее — метод оценки) должен:

- принимать на вход набор изображений в формате PNG, JPG, JPEG или видео в формате mp4 водителя из салона автомобиля, а также участвующие в оценке веса для каждого действия водителя, где вес – число с плавающей точкой в диапазоне от 0 до 1;
- в случае загрузки видео, должен разбивать его на изображения с конфигурируемым интервалом (по умолчанию в 1 секунду);
- выполнять предварительную обработку изображений, которые будут подаваться на вход нейронной сети;
- вычислять оценку безопасности водителя на основе данных, полученных из обученной модели нейронной сети.

2.2 Требования к разрабатываемому программному комплексу

Программный комплекс, реализующий интерфейс для разработанного метода, должен предоставлять:

- возможность загрузить видео или набор изображений через графический интерфейс;
- возможность задать веса для каждого действия водителя через графический интерфейс;
- запуск вычисления оценки и ее отображение в графическом интерфейсе;
- возможность отобразить изображения разделенные по действиям водителя в отдельном окне.

2.3 Особенности метода оценки

2.3.1 Модель нейронной сети

В качестве основы метода оценки будет использоваться модель нейронной сети GoogleNet, которая будет классифицировать изображения.

Для использования модели требуется ее предварительное обучение на размеченном наборе данных вида изображение – класс (метка).

После обучения модель принимает изображения размером 224x224 в формате RGB и возвращает массив вероятностей для каждого класса. Класс с наибольшей вероятностью будет считаться достоверным.

2.3.2 Общее описание метода оценки

Основные этапы метода оценки приведены на функциональной декомпозиции метода на рисунке 12.

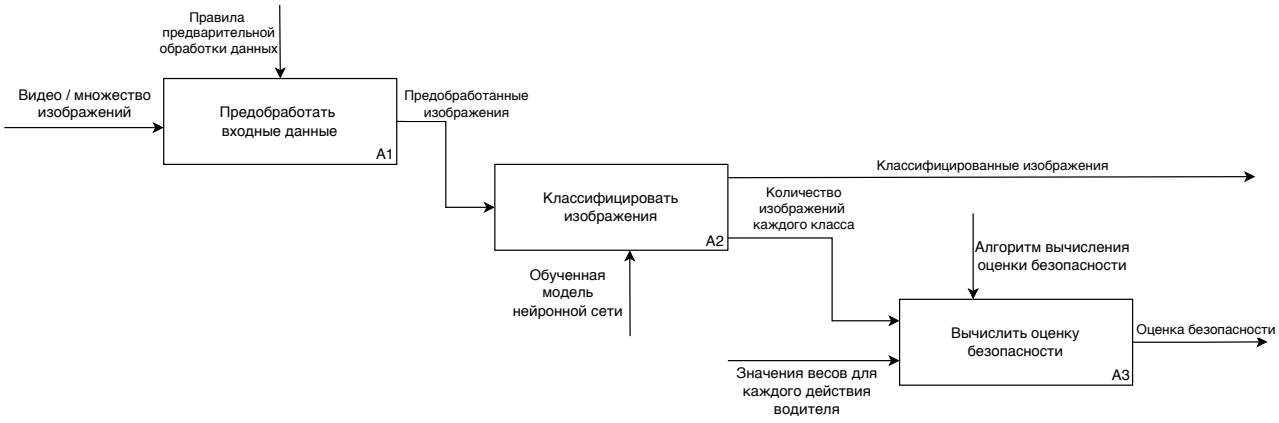

Рисунок 12 – Постановка задачи в виде IDEF0-диаграммы первого уровня

На вход методу подается видео или множество изображений, которые проходят предварительную обработку по определенным правилам, описанным далее. Затем идет классификация изображений, где каждому изображению присваивается класс (метка), а также ведется подсчет количества изображений каждого класса. На основе количества изображений каждого класса и заданных зна-

чений весов для каждого класса вычисляется оценка безопасности по алгоритму описанному далее.

2.3.3 Предварительная обработка данных

Разрабатываемый метод оценки предполагает предварительную обработку входных данных, схема алгоритма которой представлена на рисунке 13

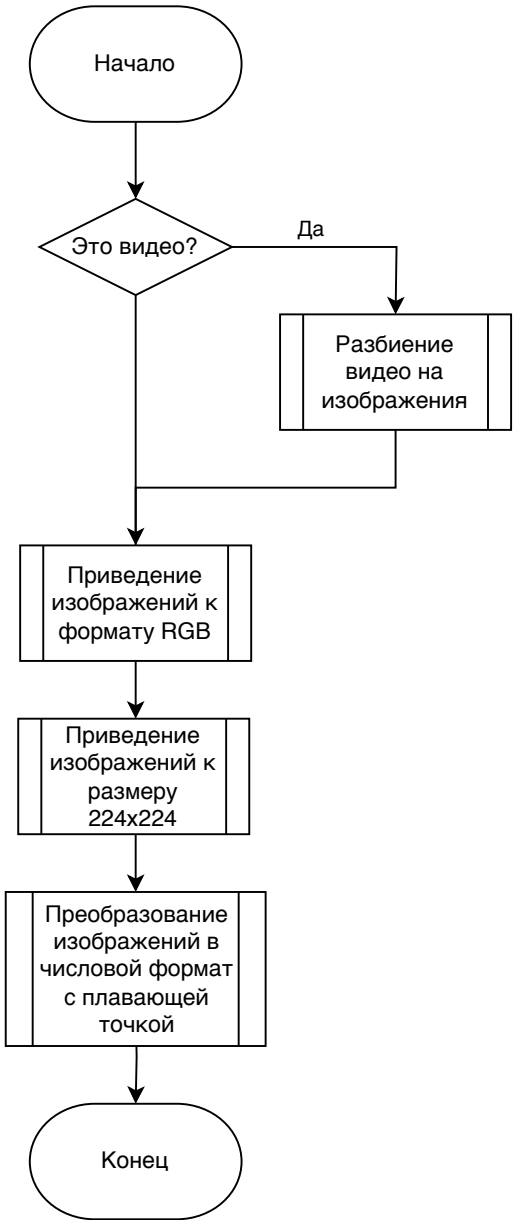

Рисунок 13 – Схема алгоритма предварительной обработки данных

Все перечисленные в схеме действия требуются для того, чтобы привести изображения к нужному для модели нейронной сети формату.

2.3.4 Алгоритм вычисления оценки безопасности

Схема алгоритма вычисления оценки безопасности на основе количества изображений каждого класса представлена на рисунке 14.

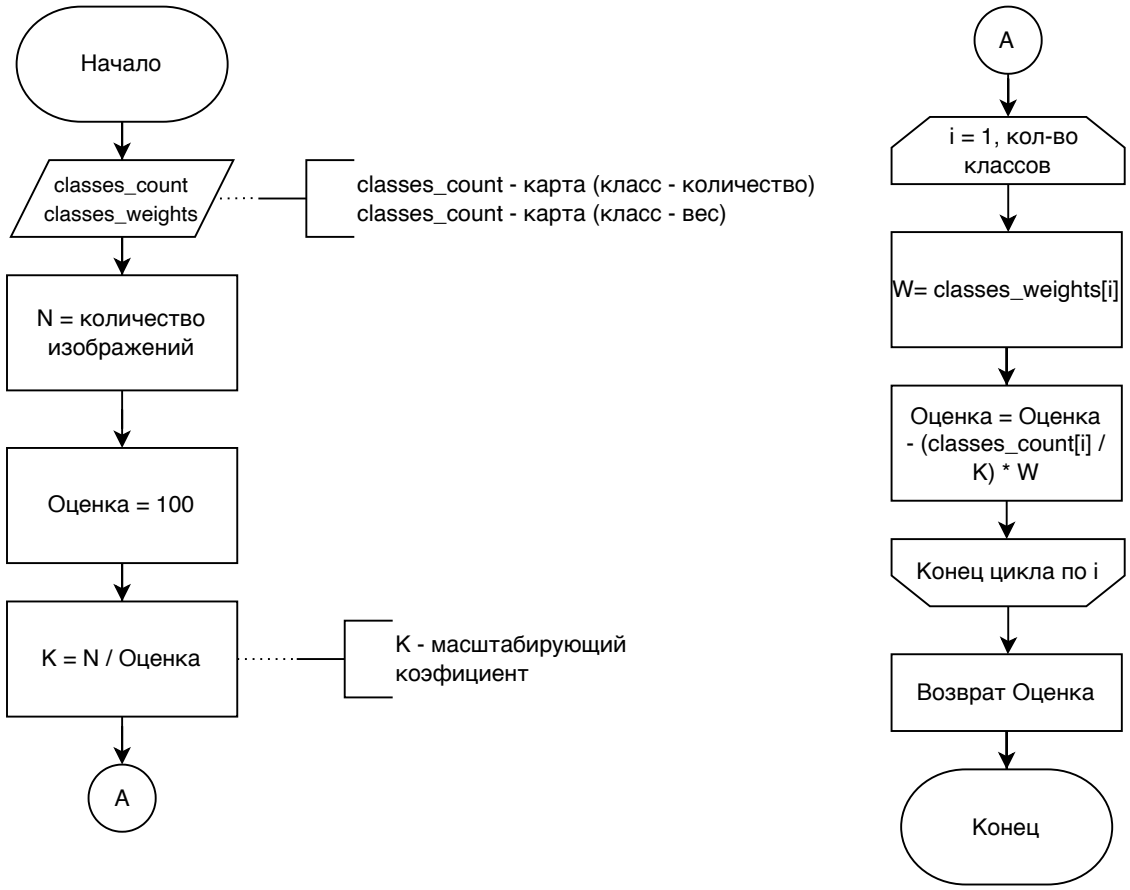

Рисунок 14 – Схема алгоритма вычисления оценки безопасности

2.4 Структура разрабатываемого программного обеспечения

Программное обеспечение состоит из четырех модулей;

- модуль GoogleNet модели;
- модуль пользовательского интерфейса;
- модуль метода оценки безопасности водителя;
- модуль загрузки и предобработки данных.

2.4.1 Описание модулей программного комплекса

Модуль GoogleNet модели

Модуль GoogleNet модели реализует модель GoogleNet для классификации изображений и предоставляет интерфейс для ее обучения. На рисунке 15 представлена схема работы модуля GoogleNet модели.

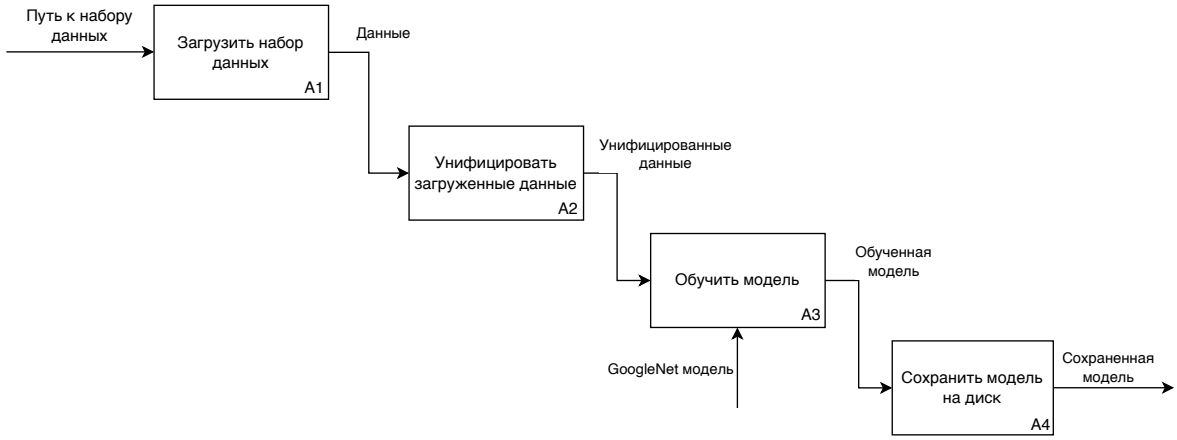

Рисунок 15 – Схема работы модуля GoogleNet модели

В данном модуле происходит обучение модели, на тренировочном наборе данных, описанном далее. Модуль используется единожды, чтобы получить обученную модель или когда требуется внести изменения в модель. Результатом работы модуля является файл с сохраненной обученной моделью, который в дальнейшем можно загрузить и использовать как модель. Также модуль предоставляет возможность дообучать модель на каком-либо другом наборе данных.

Модуль пользовательского интерфейса

Модуль пользовательского интерфейса предоставляет пользователю графический интерфейс для взаимодействия с методом оценки. Данный модуль должен давать пользователю такие возможности как:

- выбрать режим работы (видео или набор изображений);
- задать путь к данным;
- задать значения весов;
- запустить вычисление оценки;
- возможность просмотреть классифицированные изображения.

Модуль метода оценки безопасности водителя

Модуль метода оценки безопасности предоставляет программный интерфейс к методу оценки. Данный модуль загружает модель с диска, загружает данные с помощью модуля загрузки данных, предобрабатывает данные и вычисляет оценку. Результатом работы модуля являются оценка безопасности водителя и классифицированные изображения.

Модуль загрузки данных

Модуль загрузки данных предоставляет программный интерфейс позволяющий: загрузить изображения из директории, разделить видео на изображения.

2.4.2 Архитектура программного обеспечения

Схема архитектуры программного обеспечения, реализующего метод оценки безопасности водителя на основе глубоких нейронных сетей, представлена на рисунке 16.

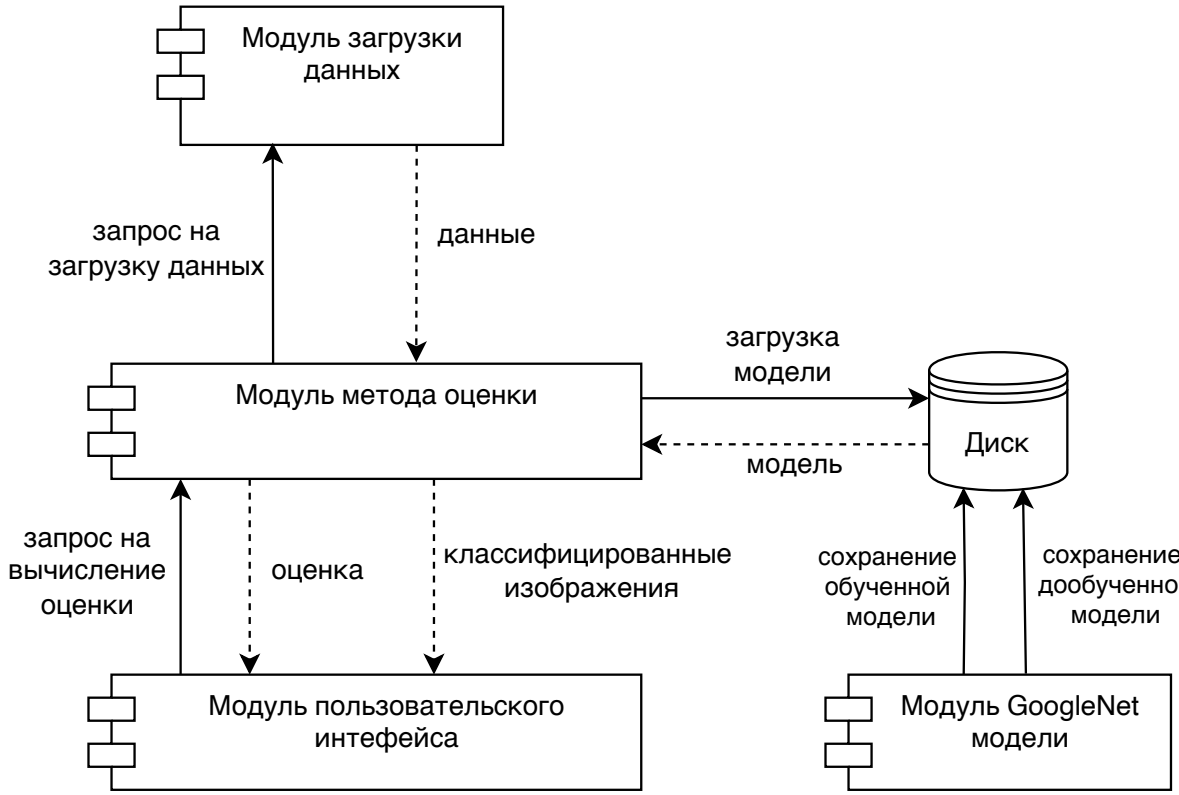

Рисунок 16 – Архитектура программного комплекса

2.4.3 Данные для обучения модели

В качестве данных для обучения модели был выбран набор данных[13], состоящий из 22424 фотографий 26 водителей из салона автомобиля, разделенных на 10 классов:

- сосредоточен;
- печатает (правая рука);
- разговаривает по телефону (правая рука);
- печатает (левая рука);
- разговаривает по телефону (правая рука);
- управляет радио;
- пьет;
- тянется назад;
- делает прическу/макияж;
- разговаривает с пассажиром.

Примеры изображений для классов «сосредоточен» и «печатает (правая рука)» представлены на рисунке 17.

Рисунок 17 – Примеры изображений из набора данных

Вывод

Были представлены требования к разрабатываемому методу оценки и программному обеспечению, реализующему интерфейс взаимодействия с методом.

Обозначены особенности разрабатываемого метода, показано применение глубоких нейронных сетей при классификации действий водителя за рулем.

Описаны модули разрабатываемого программного обеспечения.

Представлен выбор набора данных для обучения модели и примеры из него.

3 Технологический раздел

3.1 Средства реализации программного комплекса

Выбор языка программирования

Для написания программного комплекса был выбран язык программирования Python [14], так как он обладает следующими критериями:

- большое количество библиотек для работы с нейронными сетями;
- возможность создавать графические интерфейсы;
- возможность тренировать модель нейронной сети на графическом процессоре.

Выбор библиотеки для работы с нейронной сетью

Для реализации и обучения модели нейронной сети была выбрана библиотека tensorflow [15], так как является наиболее часто используемой и требует меньшего времени для обучения модели по сравнению с второй по популярности библиотекой PyTorch [16].

3.2 Реализация программного комплекса

Модули метода оценки и загрузки данных представлены в листингах В.1–В.5 и Г.1 соответственно. Далее будут подробно рассмотрены модули пользовательского интерфейса и GoogleNet модели.

3.2.1 Модуль пользовательского интерфейса

Интерфейс пользовательского приложения при запуске представлен на рисунке 18.

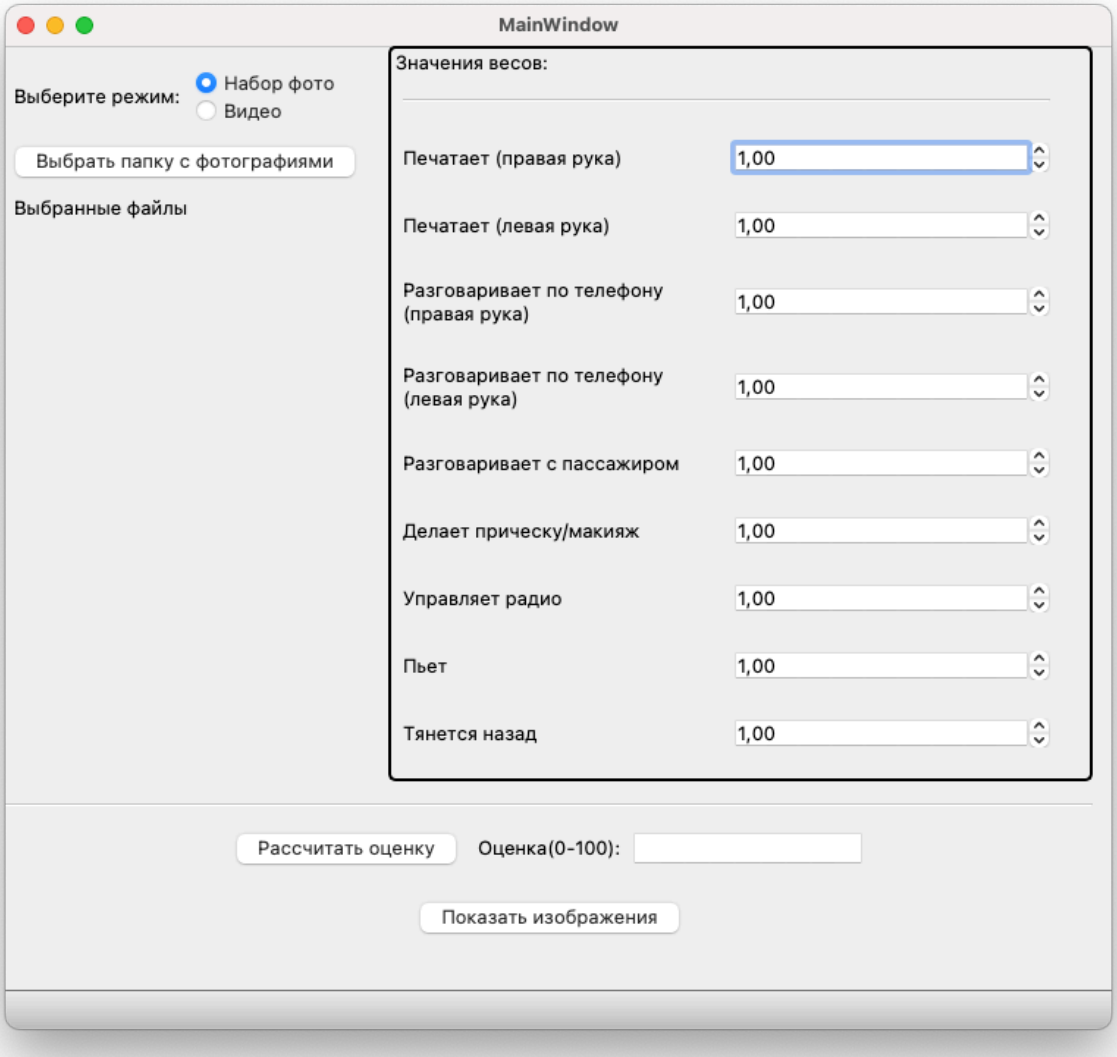

Рисунок 18 – Пользовательское приложение

3.2.2 Модуль GoogleNet модели

Реализация модели построена на классической GoogleNet модели с 9 *inception* блоками и 2 вспомогательными классификаторами. Вспомогательные классификаторы обеспечивают более эффективную и стабильную сходимость обучения.

В листинге А.1–А.6 приведена реализация модели GoogleNet, отвечающая за классификацию изображений.

У модели 3 выхода, 1 основной и 2 вспомогательных (засчет дополнитель-

ных классификаторов). Результат работы модели берется из основного выхода. Каждый выход представляет собой массив из 10 (количество классов) чисел (вероятностей). В данной работе класс с наибольшей вероятностью считается достоверным.

3.2.3 Обучение модели

Перед обучением модели необходимо произвести конвертацию исходных изображений к размеру 224 × 224 пикселей и формату RGB как этого требует модель.

В процессе обучения тренировочные данные подвергаются преобразованиям, таким как отражение, переворот, масштабирование. Это позволяет расширить набор тренировочных данных и повысить точность модели.

Обучение производилось на облачном NVIDIA DGX [17] сервере, с графическим процессором NVIDIA Tesla V100 имеющий 32 гигабайта видеопамяти.

В листинге А.7–А.10 приведена реализация процесса обучения реализованной модели GoogleNet.

Пример классификации представлен на рисунках 19 — 20.

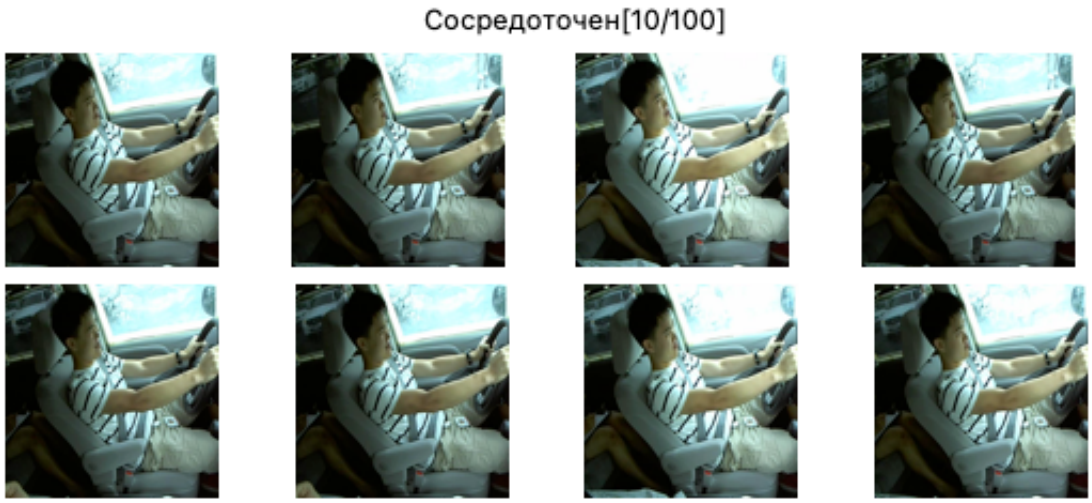

Рисунок 19 – Пример классификации, класс «Сосредоточен»

Рисунок 20 – Пример классификации, класс «Управляет радио»

Обучение производилось в течение 200 итераций (эпох), каждая эпоха состояла из 180 шагов. Набор данных для обучения был разделен на обучающий и валидационный в соотношении 4 к 1 соответственно. Валидационная часть набора нужна для того, чтобы можно оценить точность модели в конце каждой эпохи на данных, неучавствующих в обучении.

3.2.4 Результаты обучения модели

На рисунках 21 и 22 представлены метрики точности на обучающих и валидационнах данных соответственно. Стоит отметить, что точность модели начала расти только после 30 эпох на обоих наборах данных и достигла 100 процентов на обучающих данных и 98 процентов на валидационных.

Рисунок 21 – Точность модели на обучающих данных

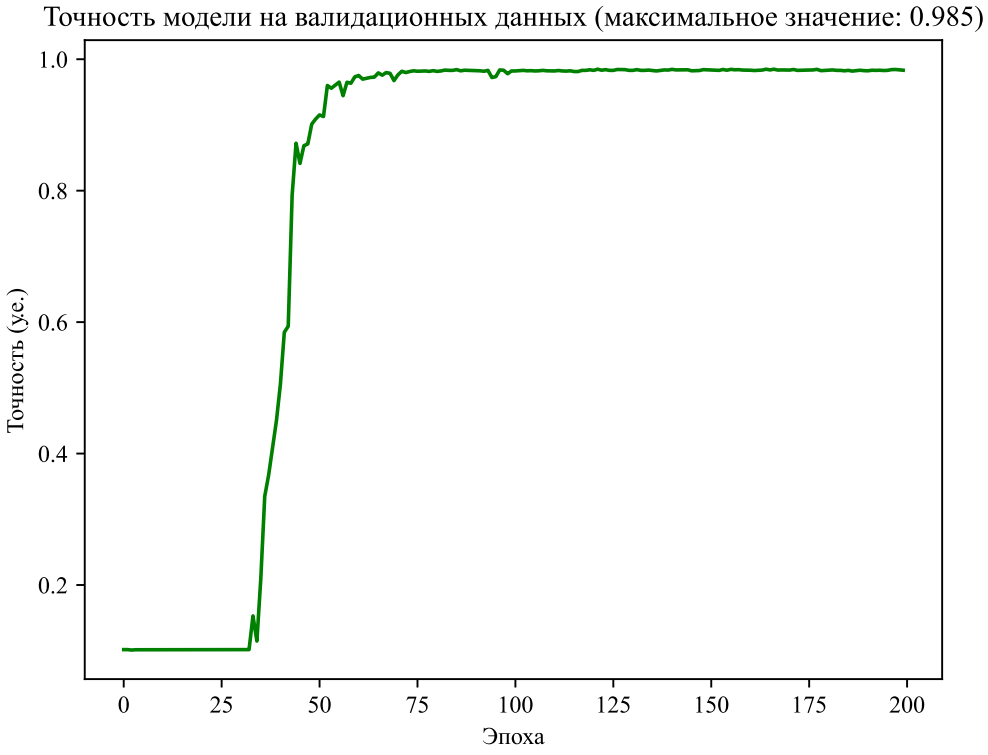

Рисунок 22 – Точность модели на валидационных данных

3.3 Примеры использования разработанного программного комплекса

Модуль модели GoogleNet представляет собой скрипт на языке программирования python, выполняющий обучение модели. Запуск данного скрипта осуществляется через консоль. Пример запуска модуля для обучения модели представлен в листингах 1–2. В качестве параметров для скрипта можно указать путь к набору данных, остальные параметры можно менять непосредственно в программном коде.

Листинг 1 – Запуск обучения модели

```
python main.py -d ./data/train

2024-04-27 01:39:16.372411: I
↪ tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:116] None
↪ of the MLIR optimization passes are enabled (registered 2)
2024-04-27 01:39:16.375235: I
↪ tensorflow/core/platform/profile_utils/cpu_utils.cc:112] CPU
↪ Frequency: 2693670000 Hz
Epoch 1/200

Epoch 00001: LearningRateScheduler reducing learning rate to 0.01.
2024-04-27 01:39:19.291402: I
↪ tensorflow/stream_executor/platform/default/dso_loader.cc:49]
↪ Successfully opened dynamic library libcublas.so.11
2024-04-27 01:39:19.771270: I
↪ tensorflow/stream_executor/platform/default/dso_loader.cc:49]
↪ Successfully opened dynamic library libcublasLt.so.11
2024-04-27 01:39:22.171898: I
↪ tensorflow/stream_executor/platform/default/dso_loader.cc:49]
↪ Successfully opened dynamic library libcudnn.so.8
```

Листинг 2 – Запуск обучения модели. Продолжение

```
2024-04-27 01:39:22.171898: I
↪ tensorflow/stream_executor/platform/default/dso_loader.cc:49]
↪ Successfully opened dynamic library libcudnn.so.8

180/180 [==============================] - 59s 269ms/step - loss: 3.7943
↪ - output_loss: 2.3949 - auxilliary_output_1_loss: 2.3347 -
↪ auxilliary_output_2_loss: 2.3298 - output_accuracy: 0.1004 -
↪ auxilliary_output_1_accuracy: 0.1069 - auxilliary_output_2_accuracy:
↪ 0.1022 - val_loss: 3.6899 - val_output_loss: 2.3087 -
↪ val_auxilliary_output_1_loss: 2.3020 - val_auxilliary_output_2_loss:
↪ 2.3021 - val_output_accuracy: 0.1017 -
↪ val_auxilliary_output_1_accuracy: 0.1015 -
↪ val_auxilliary_output_2_accuracy: 0.1015
Epoch 2/200

...

Epoch 00200: LearningRateScheduler reducing learning rate to
↪ 0.0036039671685801802.
180/180 [==============================] - 21s 118ms/step - loss:
↪ 3.0684e-04 - output_loss: 1.6573e-06 - auxilliary_output_1_loss:
↪ 9.4422e-04 - auxilliary_output_2_loss: 7.3059e-05 - output_accuracy:
↪ 1.0000 - auxilliary_output_1_accuracy: 0.9998 -
↪ auxilliary_output_2_accuracy: 1.0000 - val_loss: 0.3523 -
↪ val_output_loss: 0.2355 - val_auxilliary_output_1_loss: 0.1905 -
↪ val_auxilliary_output_2_loss: 0.1988 - val_output_accuracy: 0.9833 -
↪ val_auxilliary_output_1_accuracy: 0.9802 -
↪ val_auxilliary_output_2_accuracy: 0.9857
```

Модуль пользовательского интерфейса представляет собой графический интерфейс, запуск которого также производится через терминал.

После выбора режима нужно выбрать либо папку с фотографиями либо видеофайл на диске, затем в приложении в интерфейсе приложения будет продублирован выбранный путь как на рисунке 23.

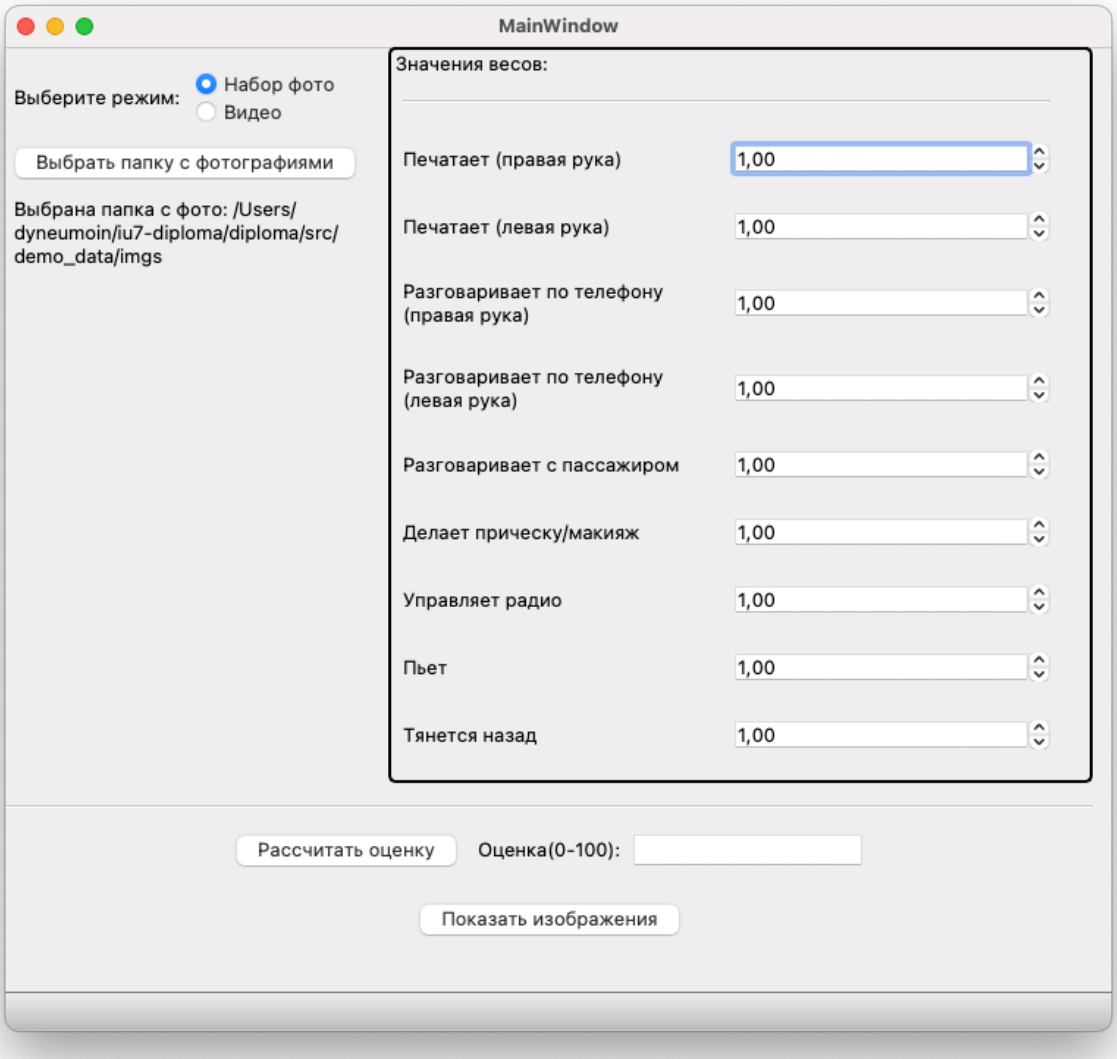

Рисунок 23 – Пользовательское приложение (выбран путь к изображениям)

После выбора пути к изображениям или видеофайлу можно рассчитать оценку, нажав на соответствующую кнопку «Рассчитать оценку», значение оценки отобразится в поле с заголовком «Оценка(0-100)» как на рисунке 24.

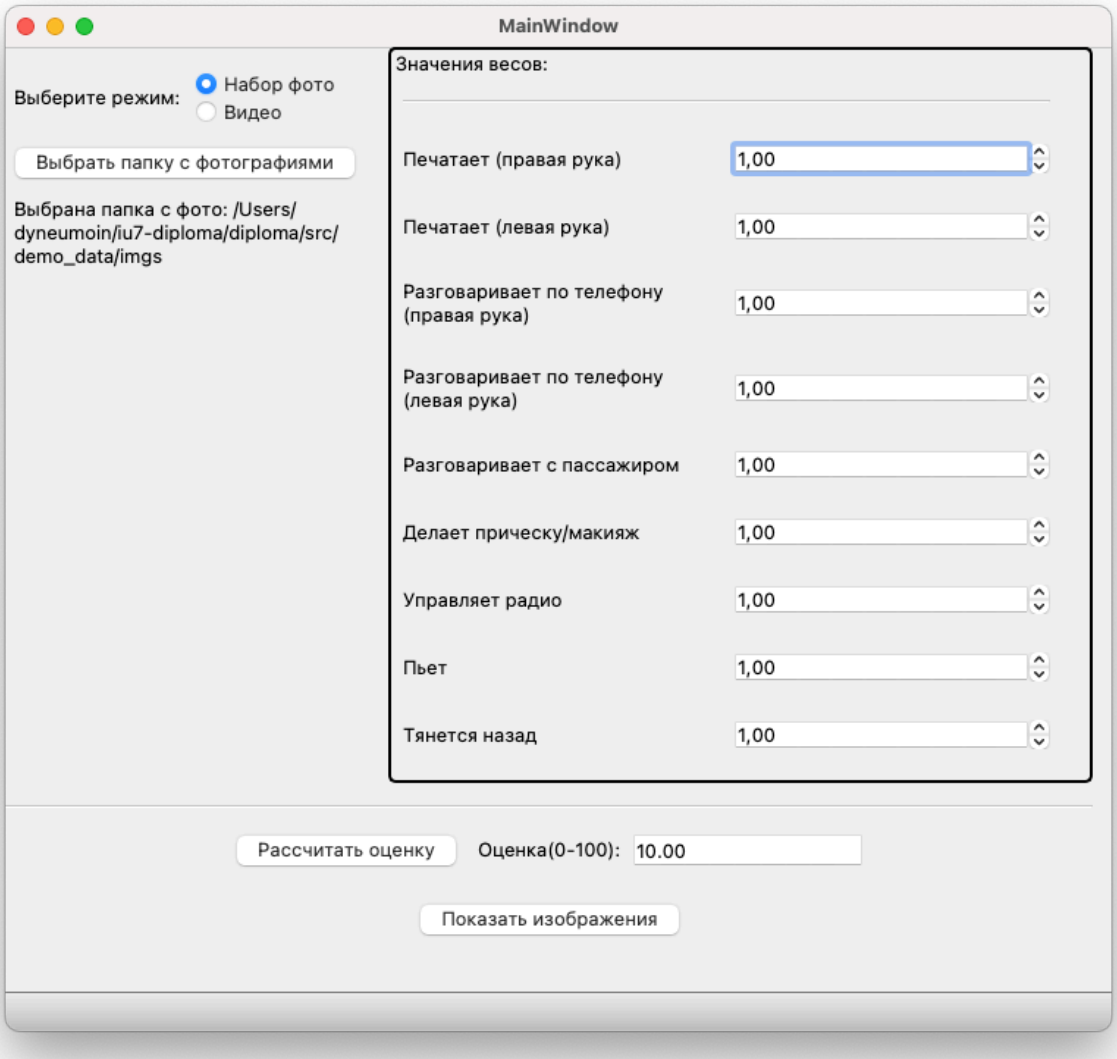

Рисунок 24 – Пользовательское приложение (рассчитана оценка)

Если требуется просмотреть изображения, разделенные по классам, то это можно сделать по нажатию кнопки «Показать изображения», откроется отдельное диалоговое окно с группами классов, в начале группы будет название класса и количество в формате [количество изображений класса / общее количество изображений] как на рисунке 25.

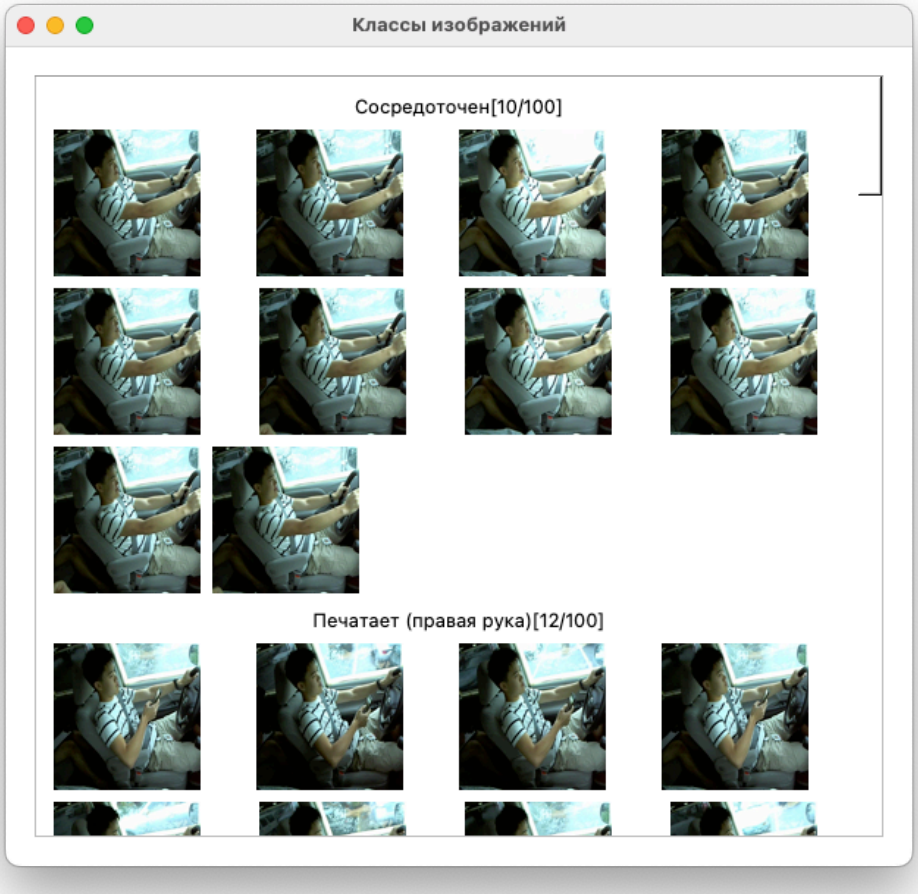

Рисунок 25 – Пользовательское приложение (диалоговое окно с изображениями)

Вывод

Были описаны средства реализации программного комплекса. Приведены листинги реализации каждого компонента комплекса, примеры работы компонентов, их входные и выходные данные. Описаны технологии и методы, использовавшиеся при реализации. Представлены примеры взаимодействия с модулями.

4 Исследовательский раздел

4.1 Предмет исследования

В данной работе будут рассмотрены следующие зависимости:

- зависимость времени работы метода от количества изображений;
- зависимость точности модели нейронной сети от размера изображений.

Время работы метода напрямую зависит от количества изображений, подаваемых на вход, так как основной операцией является классификация изображения. Точность модели зависит в основном от размера (качества) изображений и ракурса, так как в наборе данных нет изображений под другим ракурсом, то будет рассмотрена зависимость от качества изображений.

Технические характеристики

Технические характеристики устройства, на котором выполнялись сравнения, следующие.

- Операционная система: macOS 12.5.1.
- Память: 32 ГБ.
- Процессор: Apple M1 Pro CPU @ 3.22ГГц [18].

Исследование проводилось на ноутбуке, включенном в сеть электропитания. Во время экспериментов ноутбук был нагружен только встроенными приложениями окружения, а также непосредственно системой исследования.

4.2 Сравнение времени работы реализованного метода на разных объемах входных данных

На рисунке 26 приведен график зависимости времени работы метода от количества изображений, все изображения имеют исходный размер 640x480.

Из приведенного графика можно сделать вывод, что зависимость времени работы метода от количества изображений практически линейна и что на вычисление оценки на 500 изображениям потребуется примерно 25 секунд. Так как при обработке видео оно делится на изображения с интервалом в 1 секунду, то 500 изображений соответствуют примерно 8 минутам видео, следовательно на обработку 12 часового видео уйдет около 36 минут.

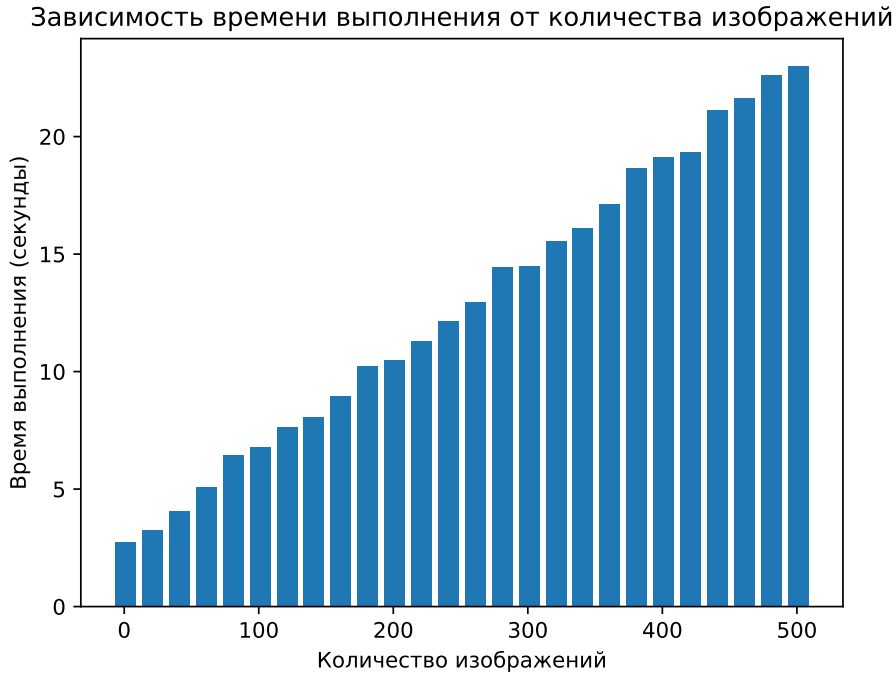

Рисунок 26 – Зависимость времени работы метода от количества изображений

4.3 Сравнение точности модели нейронной сети на изображениях разного размера

Так как модель нейронной сети требует, чтобы на вход подавались изображения размером 224x224, то есть все изображения большего размера сжимаются до этого размера. Следовательно имеет смысл рассмотреть изображения меньшего размера и сравнить точность.

Для сравнения были взяты 100 изображений из валидационного набора данных предварительно приведенных к нужным размерам.

На рисунке 27 приведен график зависимости точности модели от размера изображений.

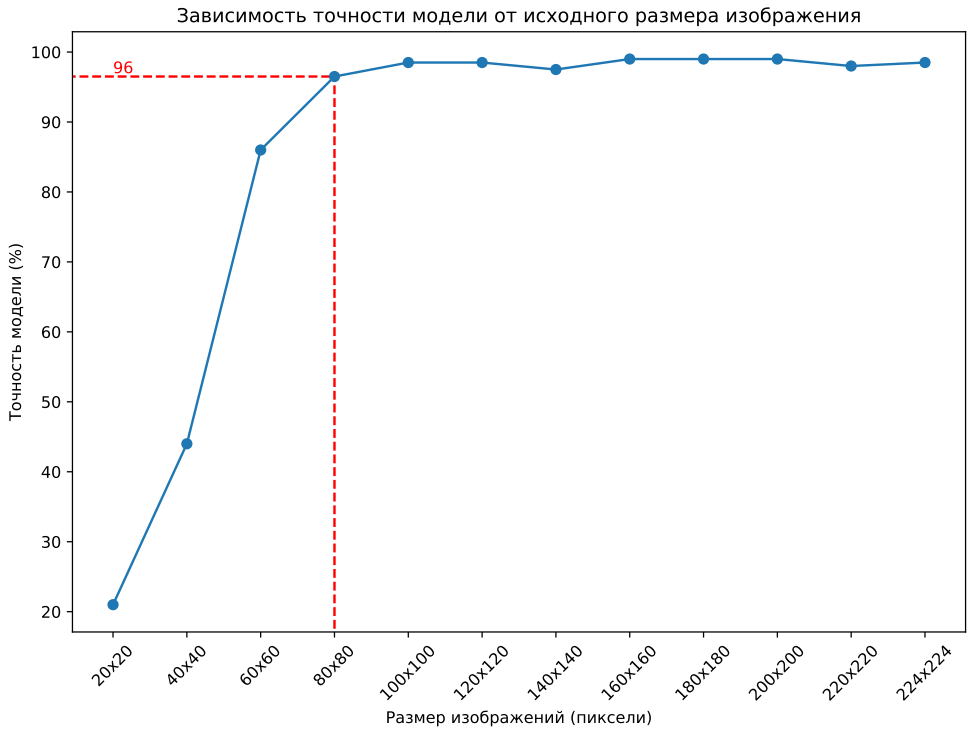

Рисунок 27 – Зависимость точности модели от размера изображений

Из приведенного графика можно сделать вывод, что на изображениях, размер которых меньше 80x80, точность значительно ниже (85% для размера 60x60, 45% для размера 40x40, 20% для размера 20x20), нежели на изображениях большего размера. Из этого следует, что для высокой точности модели и, соответственно, корректной оценки, оптимальным размером изображений является 80x80 и больше.

4.4 Результаты исследования

На основе сравнений можно сделать следующие выводы:

- зависимость времени работы метода от количества изображений линейно;
- на обработку, например, 12 часового видео потребуется примерно 36 минут, такое время работы может являться проблемой, если требуется оценивать большое количество водителей;
- наиболее оптимальным размером изображений является 80x80 и выше, при размере 80x80 точность модели остается высокой (более 95%), что позволят хранить меньший объем данных на диске без потери точности.

ЗАКЛЮЧЕНИЕ

В результате выполнения работы была достигнута цель – разработан метод оценки безопасности водителя на основе глубоких нейронных сетей.

В ходе выполнения поставленной цели были выполнены следующие задачи:

- описаны термины предметной области и формализована задача оценки безопасности водителя;
- проведен анализ методов классификации изображений;
- спроектирован метод метод оценки безопасности водителя на основе глубоких нейронных сетей;
- разработано программное обеспечение, реализующее данный метод;
- проведено исследование зависимости точности модели от размера входных данных и времени работы метода от объема входных данных.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ВЦИОМ: Такси в России [Электронный ресурс]. — Режим доступа, URL: https://wciom.ru/analytical-reviews/analiticheskii-obzor/taksi-v-rossii-mnenie-polzovatelei (дата обращения 15.10.2023).

2. Аналитический центр при правительстве РФ [Электронный ресурс]. — Режим доступа, URL: https://ac.gov.ru/archive/files/content/19540/taksi-avariinost-final-2311-pdf.pdf (дата обращения 15.10.2023).

3. Learning and transferring mid-level image representations using convolutional neural networks / Maxime Oquab, Leon Bottou, Ivan Laptev [и др.] // Proceedings of the IEEE conference on computer vision and pattern recognition. 2014. С. 1717–1724.

4. LeCun Yann, Bengio Yoshua, Hinton Geoffrey. Deep learning // nature. 2015. Т. 521, № 7553. С. 436–444.

5. Bai Yuhan. RELU-function and derived function review // SHS Web of Conferences / EDP Sciences. Т. 144. 2022. С. 02006.

6. Recent advances in convolutional neural networks / Jiuxiang Gu, Zhenhua Wang, Jason Kuen [и др.] // Pattern recognition. 2018. Т. 77. С. 354–377.

7. Review of image classification algorithms based on convolutional neural networks / Leiyu Chen, Shaobo Li, Qiang Bai [и др.] // Remote Sensing. 2021. Т. 13, № 22. С. 4712.

8. A survey of deep neural network architectures and their applications / Weibo Liu, Zidong Wang, Xiaohui Liu [и др.] // Neurocomputing. 2017. Т. 234. С. 11–26.

9. Boureau Y-Lan, Ponce Jean, LeCun Yann. A theoretical analysis of feature

pooling in visual recognition // Proceedings of the 27th international conference on machine learning (ICML-10). 2010. С. 111–118.

10. A comparative study of different CNN models and transfer learning effect for underwater object classification in side-scan sonar images / Xing Du, Yongfu Sun, Yupeng Song [и др.] // Remote Sensing. 2023. Т. 15, № 3. С. 593.

11. Going deeper with convolutions / Christian Szegedy, Wei Liu, Yangqing Jia [и др.] // Proceedings of the IEEE conference on computer vision and pattern recognition. 2015. С. 1–9.

12. Philipp George, Song Dawn, Carbonell Jaime G. The exploding gradient problem demystified-definition, prevalence, impact, origin, tradeoffs, and solutions. 2017.

13. State Farm Distracted Driver Detection [Электронный ресурс]. — Режим доступа, URL: https://kaggle.com/competitions/state-farm-distracted-driver-detection (дата обращения 01.04.2024).

14. Python [Электронный ресурс]. — Режим доступа, URL: https://www.python.org/ (дата обращения 01.04.2024).

15. Tensorflow [Электронный ресурс]. — Режим доступа, URL: https://www.tensorflow.org/ (дата обращения 01.04.2024).

16. Performance analysis of deep learning libraries: TensorFlow and PyTorch / Felipe Florencio, Thiago Valen, Edward David Moreno [и др.] // Journal of Computer Science. 2019. Т. 15, № 6. С. 785–799.

17. Tensorflow [Электронный ресурс]. — Режим доступа, URL: https://www.nvidia.com/ru-ru/data-center/dgx-systems/ (дата обращения 01.04.2024).

18. Apple M1 Pro [Электронный ресурс]. — Режим доступа, URL: https://apple.fandom.com/wiki/Apple_M1_Pro (дата обращения 01.04.2024).

ПРИЛОЖЕНИЕ А

Модуль модели GoogleNet

Листинг А.1 – Модель GoogleNet. Часть 1

```
import keras
from keras.models import Model
from keras.layers import Conv2D, MaxPool2D,Dropout, Dense, Input,
↪  concatenate,GlobalAveragePooling2D, AveragePooling2D,Flatten

kernel_init = keras.initializers.glorot_uniform()
bias_init = keras.initializers.Constant(value=0.2)

def inception_module(x,
                     filters_1x1,
                     filters_3x3_reduce,
                     filters_3x3,
                     filters_5x5_reduce,
                     filters_5x5,
                     filters_pool_proj,
                     name=None):

    conv_1x1 = Conv2D(filters_1x1, (1, 1), padding='same',
    ↪  activation='relu', kernel_initializer=kernel_init,
    ↪  bias_initializer=bias_init)(x)

    conv_3x3 = Conv2D(filters_3x3_reduce, (1, 1), padding='same',
    ↪  activation='relu', kernel_initializer=kernel_init,
    ↪  bias_initializer=bias_init)(x)
    conv_3x3 = Conv2D(filters_3x3, (3, 3), padding='same',
    ↪  activation='relu', kernel_initializer=kernel_init,
    ↪  bias_initializer=bias_init)(conv_3x3)

```

Листинг А.2 – Модель GoogleNet. Часть 2

```
22        conv_5x5 = Conv2D(filters_5x5_reduce, (1, 1), padding='same',
          ↪  activation='relu', kernel_initializer=kernel_init,
          ↪  bias_initializer=bias_init)(x)
23        conv_5x5 = Conv2D(filters_5x5, (5, 5), padding='same',
          ↪  activation='relu', kernel_initializer=kernel_init,
          ↪  bias_initializer=bias_init)(conv_5x5)
24
25        pool_proj = MaxPool2D((3, 3), strides=(1, 1), padding='same')(x)
26        pool_proj = Conv2D(filters_pool_proj, (1, 1), padding='same',
          ↪  activation='relu', kernel_initializer=kernel_init,
          ↪  bias_initializer=bias_init)(pool_proj)
27
28        output = concatenate([conv_1x1, conv_3x3, conv_5x5, pool_proj],
          ↪  axis=3, name=name)
29
30        return output
31
32    def GoogleNet():
33        input_layer = Input(shape=(224, 224, 3))
34
35        x = Conv2D(64, (7, 7), padding='same', strides=(2, 2),
          ↪  activation='relu', name='conv_1_7x7_2',
          ↪  kernel_initializer=kernel_init,
          ↪  bias_initializer=bias_init)(input_layer)
36        x = MaxPool2D((3, 3), padding='same', strides=(2, 2),
          ↪  name='max_pool_1_3x3_2')(x)
37        x = Conv2D(64, (1, 1), padding='same', strides=(1, 1),
          ↪  activation='relu', name='conv_2a_3x3_1')(x)
38        x = Conv2D(192, (3, 3), padding='same', strides=(1, 1),
          ↪  activation='relu', name='conv_2b_3x3_1')(x)
39        x = MaxPool2D((3, 3), padding='same', strides=(2, 2),
          ↪  name='max_pool_2_3x3_2')(x)
40
```

Листинг А.3 – Модель GoogleNet. Часть 3

```
41      x = inception_module(x,
42                           filters_1x1=64,
43                           filters_3x3_reduce=96,
44                           filters_3x3=128,
45                           filters_5x5_reduce=16,
46                           filters_5x5=32,
47                           filters_pool_proj=32,
48                           name='inception_3a')
49
50      x = inception_module(x,
51                           filters_1x1=128,
52                           filters_3x3_reduce=128,
53                           filters_3x3=192,
54                           filters_5x5_reduce=32,
55                           filters_5x5=96,
56                           filters_pool_proj=64,
57                           name='inception_3b')
58
59      x = MaxPool2D((3, 3), padding='same', strides=(2, 2),
        ↪  name='max_pool_3_3x3_2')(x)
60
61      x = inception_module(x,
62                           filters_1x1=192,
63                           filters_3x3_reduce=96,
64                           filters_3x3=208,
65                           filters_5x5_reduce=16,
66                           filters_5x5=48,
67                           filters_pool_proj=64,
68                           name='inception_4a')
69
70
71      x1 = AveragePooling2D((5, 5), strides=3)(x)
72      x1 = Conv2D(128, (1, 1), padding='same', activation='relu')(x1)
73      x1 = Flatten()(x1)
74      x1 = Dense(1024, activation='relu')(x1)
```

Листинг А.4 – Модель GoogleNet. Часть 4

```
        x1 = Dropout(0.7)(x1)
        x1 = Dense(10, activation='softmax', name='auxilliary_output_1')(x1)

        x = inception_module(x,
                             filters_1x1=160,
                             filters_3x3_reduce=112,
                             filters_3x3=224,
                             filters_5x5_reduce=24,
                             filters_5x5=64,
                             filters_pool_proj=64,
                             name='inception_4b')

        x = inception_module(x,
                             filters_1x1=128,
                             filters_3x3_reduce=128,
                             filters_3x3=256,
                             filters_5x5_reduce=24,
                             filters_5x5=64,
                             filters_pool_proj=64,
                             name='inception_4c')

        x = inception_module(x,
                             filters_1x1=112,
                             filters_3x3_reduce=144,
                             filters_3x3=288,
                             filters_5x5_reduce=32,
                             filters_5x5=64,
                             filters_pool_proj=64,
                             name='inception_4d')


        x2 = AveragePooling2D((5, 5), strides=3)(x)
        x2 = Conv2D(128, (1, 1), padding='same', activation='relu')(x2)
        x2 = Flatten()(x2)
        x2 = Dense(1024, activation='relu')(x2)
```

Листинг А.5 – Модель GoogleNet. Часть 5

```
110         x2 = Dropout(0.7)(x2)
111         x2 = Dense(10, activation='softmax', name='auxilliary_output_2')(x2)
112 
113         x = inception_module(x,
114                              filters_1x1=256,
115                              filters_3x3_reduce=160,
116                              filters_3x3=320,
117                              filters_5x5_reduce=32,
118                              filters_5x5=128,
119                              filters_pool_proj=128,
120                              name='inception_4e')
121 
122         x = MaxPool2D((3, 3), padding='same', strides=(2, 2),
            ↪  name='max_pool_4_3x3_2')(x)
123 
124         x = inception_module(x,
125                              filters_1x1=256,
126                              filters_3x3_reduce=160,
127                              filters_3x3=320,
128                              filters_5x5_reduce=32,
129                              filters_5x5=128,
130                              filters_pool_proj=128,
131                              name='inception_5a')
132 
133         x = inception_module(x,
134                              filters_1x1=384,
135                              filters_3x3_reduce=192,
136                              filters_3x3=384,
137                              filters_5x5_reduce=48,
138                              filters_5x5=128,
139                              filters_pool_proj=128,
140                              name='inception_5b')
141 
142         x = GlobalAveragePooling2D(name='avg_pool_5_3x3_1')(x)
```

Листинг А.6 – Модель GoogleNet. Часть 6

```
143
144         x = Dropout(0.4)(x)
145
146         x = Dense(10, activation='softmax', name='output')(x)
147
148         model = Model(input_layer, [x, x1, x2], name='inception_v1')
149         return model
```

Листинг А.7 – Обучение модели. Часть 1

```
import tensorflow as tf
import keras
from keras.optimizers import SGD
from keras.callbacks import LearningRateScheduler
from keras.models import load_model
import math
from model import GoogleNet
import argparse

epochs = 200
batch_size = 100

def get_generators(base_dir_train, target_size=(224, 224)):
    train_ds = tf.keras.preprocessing.image_dataset_from_directory(
        base_dir_train,
        validation_split=0.2,
        subset="training",
        seed=123,
        label_mode='int',
        image_size=target_size,
        batch_size=batch_size
    )

    validation_ds = tf.keras.preprocessing.image_dataset_from_directory(
        base_dir_train,
        validation_split=0.2,
        subset="validation",
        label_mode='int',
        seed=123,
        image_size=target_size,
        batch_size=batch_size
    )


    class_names = train_ds.class_names
```

Листинг А.8 – Обучение модели. Часть 2

```
        print(class_names)
        print("before aug: ", train_ds.cardinality().numpy() * batch_size)

        data_augmentation = keras.Sequential(
            [
                tf.keras.layers.experimental
                    .preprocessing.RandomFlip("horizontal"),
                tf.keras.layers.experimental
                    .preprocessing.RandomRotation(0.1),
                tf.keras.layers.experimental
                    .preprocessing.RandomZoom(0.1),
            ]
        )

        train_ds = train_ds.map(lambda x, y: (data_augmentation(x), y),
        ↪  num_parallel_calls=tf.data.AUTOTUNE)

        normalization_layer =
        ↪  tf.keras.layers.experimental.preprocessing.Rescaling(1./255)
        normalized_train_ds = train_ds.map(lambda x, y:
        ↪  (normalization_layer(x), y))
        normalized_validation_ds = validation_ds.map(lambda x, y:
        ↪  (normalization_layer(x), y))

        AUTOTUNE = tf.data.AUTOTUNE

        normalized_train_ds =
        ↪  normalized_train_ds.cache().prefetch(buffer_size=AUTOTUNE)
        normalized_validation_ds =
        ↪  normalized_validation_ds.cache().prefetch(buffer_size=AUTOTUNE)

        return normalized_train_ds.take(3000),
        ↪  normalized_validation_ds.take(500)

    def train(dataset_dir = "./data/train", is_add_train = False):
```

Листинг А.9 – Обучение модели. Часть 3

```
64      num_classes = 10
65      img_rows,img_cols = 224, 224
66
67      train_ds, test_ds = get_generators(dataset_dir, (img_rows,img_cols))
68
69      initial_lrate = 0.01
70
71      def decay(epoch, steps=100):
72          initial_lrate = 0.01
73          drop = 0.96
74          epochs_drop = 8
75          lrate = initial_lrate * math.pow(drop,
            ↪  math.floor((1+epoch)/epochs_drop))
76          return lrate
77
78
79      sgd = SGD(learning_rate=initial_lrate, momentum=0.9, nesterov=False)
80
81      lr_sc = LearningRateScheduler(decay, verbose=1)
82
83      if not is_add_train:
84          model = GoogleNet()
85          model.compile(loss=keras.losses.sparse_categorical_crossentropy,
            ↪  loss_weights=[1, 0.3, 0.3], optimizer=sgd,
            ↪  metrics=['accuracy'])
86      else:
87          model = load_model('model.keras')
88
89      # Moddel Training
90      train_ds = train_ds.map(lambda x, y: (x, (y, y, y)))
91      test_ds = test_ds.map(lambda x, y: (x, (y, y, y)))
92
93
94      history = model.fit(
95          train_ds,
```

Листинг А.10 – Обучение модели. Часть 4

```
            validation_data=test_ds,
            epochs=epochs,
            batch_size=batch_size,
            callbacks=[lr_sc]
        )

        model.save('model.keras')

    if __name__ == '__main__':
        parser = argparse.ArgumentParser()
        teeth = True

        parser.add_argument('-d', '--dataset', default='./data/train',
        ↪  help='Путь к набору данных')
        parser.add_argument('-m', '--mode', default='train', help='Режим
        ↪  работы: train/add_train (обучение/дообучение)')

        args = parser.parse_args()

        if args.mode == 'train':
            print('Running training:')
            train(args.dataset, is_add_train=False)
        elif args.mode == 'add_train':
            print('Running additional training:')
            train(args.dataset, is_add_train=True)
```

ПРИЛОЖЕНИЕ Б

Модуль пользовательского интерфейса

Листинг Б.1 – Интерфейс главного окна. Часть 1

```
from PyQt5 import uic
from PyQt5.QtWidgets import (QApplication, QMainWindow, QPushButton,
↪  QLabel, QFileDialog, QDoubleSpinBox, QLineEdit, QRadioButton)
import sys
import app.main
import images_window

class MainWindow(QMainWindow):
    def __init__(self):
        super(MainWindow, self).__init__()
        self.ui = uic.loadUi("./ui/untitled.ui", self)

        # Определим виджеты.
        self.photo_button = self.findChild(QPushButton, 'choice_photo')
        self.video_button = self.findChild(QPushButton, 'choice_video')
        self.mode_photo_radioButton = self.findChild(QRadioButton,
        ↪  'mode_photo_radioButton')
        self.mode_video_radioButton = self.findChild(QRadioButton,
        ↪  'mode_video_radioButton')
        self.calc_button = self.findChild(QPushButton, 'calc')
        self.files_label = self.findChild(QLabel, 'choiced_files_label')
        self.assessment = self.findChild(QLineEdit, 'assessment_line')
        self.show_images_button = self.findChild(QPushButton,
        ↪  'show_images_button')

        # Beca
        self.texting_right_spin = self.findChild(QDoubleSpinBox,
        ↪  'texting_right_spin')
        self.talking_right_spin = self.findChild(QDoubleSpinBox,
        ↪  'talking_right_spin')
```

Листинг Б.2 – Интерфейс главного окна. Часть 2

```
25            self.texting_left_spin = self.findChild(QDoubleSpinBox,
              ↪  'texting_left_spin')
26            self.talking_left_spin = self.findChild(QDoubleSpinBox,
              ↪  'talking_left_spin')
27            self.operating_radio_spin = self.findChild(QDoubleSpinBox,
              ↪  'operation_radio_spin')
28            self.drinking_spin = self.findChild(QDoubleSpinBox,
              ↪  'drink_spin')
29            self.reaching_begind_spin = self.findChild(QDoubleSpinBox,
              ↪  'behind_spin')
30            self.hair_and_makeup_spin = self.findChild(QDoubleSpinBox,
              ↪  'makeup_spin')
31            self.talking_to_passenger = self.findChild(QDoubleSpinBox,
              ↪  'talking_passanger_spin')
32
33
34            # Обработчики
35            self.photo_button.clicked.connect(self.photo_button_clicked)
36            self.video_button.clicked.connect(self.video_button_clicked)
37            self.mode_photo_radioButton.toggled.connect(
38                self.on_radio_toggled
39                )
40            self.mode_video_radioButton.toggled.connect(
41                self.on_radio_toggled
42                )
43            self.on_radio_toggled()
44            self.calc_button.clicked.connect(self.calc_button_clicked)
45            self.show_images_button.clicked.connect(
46                self.show_images_button_clicked
47                )
48
49            # Контекст.
50            self.mode = None
51            self.file_path = ''
52            self.classes_images = []
```

Листинг Б.3 – Интерфейс главного окна. Часть 3

```
52          self.classes_images = []
53
54      def photo_button_clicked(self):
55          folder_dialog = QFileDialog()
56          folder_path = folder_dialog.getExistingDirectory(self, 'Выбрать
            ↪  папку с фото')
57
58          if folder_path:
59              self.files_label.setText("Выбрана папка с фото: " +
                ↪  folder_path)
60              self.mode = app.main.DriverMode.PHOTO
61              self.file_path = folder_path
62
63      def video_button_clicked(self):
64          file_dialog = QFileDialog()
65          file_path, _ = file_dialog.getOpenFileName(self, 'Выбрать
            ↪  видео', '', 'Video files (*.mp4 *.avi)')
66
67          if file_path:
68              self.files_label.setText("Выбрано видео: " + file_path)
69              self.mode = app.main.DriverMode.VIDEO
70              self.file_path = file_path
71
72      def calc_button_clicked(self):
73          classes_weights = self.__get_classes_weights()
74          print(f'calc: file:{self.file_path}, weights:{classes_weights}')
75          cfg = app.main.DriverSafetyGetterConfig(classes_weights,
            ↪  self.mode, self.file_path)
76          getter = app.main.DriverSafetyAssessmentGetter(cfg)
77          assessment = getter.get_driver_safety_assessment()
78          print(f'assesment: {assessment}')
79          self.assessment.setText(f"{assessment:.2f}")
80          self.classes_images = getter.get_classes_images()
81
82      def on_radio_toggled(self):
```

Листинг Б.4 – Интерфейс главного окна. Часть 4

```
83          if self.mode_photo_radioButton.isChecked():
84              self.photo_button.setVisible(True)
85              self.video_button.setVisible(False)
86          elif self.mode_video_radioButton.isChecked():
87              self.photo_button.setVisible(False)
88              self.video_button.setVisible(True)
89
90      def __get_classes_weights(self):
91          classes_weights= {
92              'texting - right': self.texting_right_spin.value(),
93              'talking on the phone - right':
                ↪  self.talking_right_spin.value(),
94              'texting - left': self.texting_left_spin.value(),
95              'talking on the phone - left':
                ↪  self.talking_left_spin.value(),
96              'operating the radio': self.operating_radio_spin.value(),
97              'drinking': self.drinking_spin.value(),
98              'reaching behind': self.reaching_begind_spin.value(),
99              'hair and makeup': self.hair_and_makeup_spin.value(),
100             'talking to passenger': self.talking_to_passenger.value(),
101         }
102
103         return classes_weights
104
105     def show_images_button_clicked(self):
106         self.img_window =
            ↪  images_window.GalleryWindow(self.classes_images)
107         self.img_window.show()
108
109 def main():
110     app = QApplication(sys.argv)
```

Листинг Б.5 – Интерфейс главного окна. Часть 5

```
111         window = MainWindow()
112         window.show()
113         return app.exec()
114 
115 
116     if __name__ == '__main__':
117         sys.exit(main())
```

Листинг Б.6 – Интерфейс диалогового окна с изображениями. Часть 1

```
import sys
from PyQt5.QtWidgets import (
    QApplication,
    QWidget,
    QPushButton,
    QVBoxLayout,
    QLabel,
    QScrollArea,
    QHBoxLayout,
    QMainWindow,
    QSpacerItem,
    QSizePolicy,
)
from PyQt5.QtGui import QPixmap, QImage
from PyQt5.QtCore import Qt
from PIL import Image
from io import BytesIO

classes_en_ru_mapping = {
    "safe driving": "Сосредоточен",
    "texting - right": "Печатает (правая рука)",
    "talking on the phone - right": "Разговаривает по телефону (правая
    ↪  рука)",
    "texting - left": "Печатает (левая рука)",
    "talking on the phone - left": "Разговаривает по телефону (левая
    ↪  рука)",
    "operating the radio": "Управляет радио",
    "drinking": "Пьет",
    "reaching behind": "Тянется назад",
    "hair and makeup": "Делает прическу/макияж",
    "talking to passenger": "Разговаривает с пассажиром",
}


class GalleryWindow(QWidget):
```

Листинг Б.7 – Интерфейс диалогового окна с изображениями. Часть 2

```
34      def __init__(self, images):
35          super().__init__()
36          self.images = images  # Словарь с изображениями
37          self.total_elements = sum(len(images) for images in
            ↪  self.images.values())
38          self.initUI()
39
40      def initUI(self):
41          self.setStyleSheet("background-color: white;")
42          layout = QVBoxLayout()
43
44          scroll = QScrollArea()  # Создание области прокрутки
45          widget = QWidget()  # Главный виджет, содержащий все остальные
            ↪  виджеты
46          scroll.setWidget(
47              widget
48          )  # Установка виджета в качестве содержимого области прокрутки
49          scroll.setWidgetResizable(True)  # Разрешение изменения размера
            ↪  виджета
50
51          vbox = QVBoxLayout()  # Вертикальное расположение для всех
            ↪  элементов
52
53          for class_name, imgs in self.images.items():  # Перебор всех
            ↪  классов изображений
54              label = QLabel(
55                  f"{classes_en_ru_mapping[class_name]}"+
56                  f"[{len(imgs)}/{self.total_elements}]"
57              )
58              label.setAlignment(Qt.AlignCenter)
59              vbox.addWidget(label)
60
61              imgs_in_row = 4
62              row_counter = 0
63              hbox = QHBoxLayout()
```

Листинг Б.8 – Интерфейс диалогового окна с изображениями. Часть 3

```
64                  hbox.setSpacing(0)
65                  hbox.setContentsMargins(0, 0, 0, 0)
66
67                  for img in imgs:
68                      if row_counter >= imgs_in_row:
69                          vbox.addLayout(hbox)
70                          hbox = QHBoxLayout()
71                          row_counter = 0
72
73                      pixmap = QPixmap.fromImage(
74                          self.convert_pil_image_to_qimage(img)
75                      )  # Создание изображения
76                      img_label = QLabel(self)  # Создание метки для
                        ↪  изображения
77                      img_label.setPixmap(pixmap)  # Установка изображения в
                        ↪  метку
78                      hbox.addWidget(
79                          img_label
80                      )  # Добавление метки в горизонтальное расположение
81                      row_counter += 1  # Увеличение счетчика изображений в
                        ↪  строке
82
83                  if row_counter > 0 and row_counter < 4:
84                      spacer = QSpacerItem(
85                          100, 100, QSizePolicy.Expanding, QSizePolicy.Minimum
86                      )
87                      hbox.addSpacerItem(spacer)
88                  if (
89                      row_counter > 0
90                  ):  # Проверка, есть ли необработанные изображения после
                    ↪  выхода из цикла
91                      vbox.addLayout(
92                          hbox
93                      )  # Добавление последнего горизонтального расположения
                        ↪  в вертикальное
```

Листинг Б.9 – Интерфейс диалогового окна с изображениями. Часть 4

```
94
95          widget.setLayout(
96              vbox
97          )  # Установка вертикального расположения в качестве основного
            ↪  для виджета
98          layout.addWidget(
99              scroll
100         )  # Добавление области прокрутки в основное вертикальное
            ↪  расположение
101         self.setLayout(
102             layout
103         )  # Установка основного вертикального расположения для self
104         self.setWindowTitle("Классы изображений")  # Задание заголовка
            ↪  окна
105         self.resize(600, 400)  # Установка начального размера окна
106
107     def convert_pil_image_to_qimage(self, pil_image):
108         pil_image = pil_image.resize((100, 100))
109         image_data = BytesIO()
110         pil_image.save(image_data, format="PNG")
111         qimage = QImage()
112         qimage.loadFromData(image_data.getvalue())
113         return qimage
```

ПРИЛОЖЕНИЕ В

Модуль метода оценки безопасности водителя

Листинг В.1 – Метод оценки безопасности водителя. Часть 1

```
from enum import Enum

from . import get_images
import tensorflow as tf
import numpy as np

import matplotlib.pyplot as plt

classes = {
    'c0': 'safe driving',
    'c1': 'texting - right',
    'c2': 'talking on the phone - right',
    'c3': 'texting - left',
    'c4': 'talking on the phone - left',
    'c5': 'operating the radio',
    'c6': 'drinking',
    'c7': 'reaching behind',
    'c8': 'hair and makeup',
    'c9': 'talking to passenger'
}

default_classes_weights= {
    'texting - right': 1,
    'talking on the phone - right': 1,
    'texting - left': 1,
    'talking on the phone - left': 1,
    'operating the radio': 1,
    'drinking': 1,
    'reaching behind': 1,
    'hair and makeup': 1,
    'talking to passenger': 1,
```

Листинг В.2 – Метод оценки безопасности водителя. Часть 2

```
    }

    class DriverMode(Enum):
        VIDEO = "video"
        PHOTO = "photo"

    class DriverSafetyGetterConfig:
        def __init__(self, classes_weights, mode: DriverMode, path_to_files,
        ↪  interval_sec=1):
            self.classes_weights = classes_weights
            self.mode = mode
            self.path_to_files = path_to_files
            self.interval_sec = interval_sec if mode == DriverMode.VIDEO
            ↪  else None

    class DriverSafetyAssessmentGetter:
        def __init__(self, cfg: DriverSafetyGetterConfig):
            self.cfg = cfg
            self.model = tf.keras.models.load_model('./working/aug.keras')

            self.images = []
            self.input_datas = []
            self.classes_count= {
                'safe driving': 0,
                'texting - right': 0,
                'talking on the phone - right': 0,
                'texting - left': 0,
                'talking on the phone - left': 0,
                'operating the radio': 0,
                'drinking': 0,
                'reaching behind': 0,
                'hair and makeup': 0,
                'talking to passenger': 0,
            }
            self.classes_images = {
```

Листинг В.3 – Метод оценки безопасности водителя. Часть 3

```
            'safe driving': [],
            'texting - right': [],
            'talking on the phone - right': [],
            'texting - left': [],
            'talking on the phone - left': [],
            'operating the radio': [],
            'drinking': [],
            'reaching behind': [],
            'hair and makeup': [],
            'talking to passenger': [],
        }

    def get_driver_safety_assessment(self, limit=100): #TODO убрать
    ↪ лимит после исследовательской
        self.__set_images()
        self.__prepare_input_datas()
        self.input_datas = self.input_datas[:limit] # TODO: убрать

        for i, input_data in enumerate(self.input_datas):
            prediction = self.model.predict(input_data)
            predicted_class = self.__get_class_by_prediction(prediction)
            self.classes_images[predicted_class].append(self.images[i])
            self.classes_count[predicted_class] += 1

        # for image in classes_images['talking to passenger']:
        #     plt.imshow(image)
        #     plt.show()

        return self.__get_assessment_by_classes_count()

    def get_classes_images(self):
        return self.classes_images

    def __get_assessment_by_classes_count(self):
        print(self.classes_count)
```

Листинг В.4 – Метод оценки безопасности водителя. Часть 4

```
99          assesment = 100
100         k = len(self.input_datas) / assesment
101         for class_name, weight in self.cfg.classes_weights.items():
102             assesment -= (self.classes_count[class_name] / k) * weight
103         print(assesment)
104         return assesment
105
106     def __set_images(self):
107         cfg = self.cfg
108         if cfg.mode == DriverMode.VIDEO:
109             self.images = get_images.get_images_from_video(
110                 path_to_video=cfg.path_to_files,
                    ↪  interval_sec=cfg.interval_sec
111                 )
112         elif cfg.mode == DriverMode.PHOTO:
113             self.images = get_images.get_images_from_folder(
114                 path_to_images=cfg.path_to_files
115                 )
116
117     def __prepare_input_datas(self):
118         input_datas = []
119         for i in self.images:
120             prepared_image = i.resize((224, 224))
121             if prepared_image.mode != 'RGB':
122                 prepared_image = prepared_image.convert('RGB')
123             input_data =
                ↪  tf.expand_dims(tf.image.convert_image_dtype(prepared_image,
                ↪  tf.float32), 0)
124             input_datas.append(input_data)
125         self.input_datas = input_datas
126
127     def __get_class_by_prediction(self, prediction):
128         first_output = prediction[0][0]
129         prediction_array = np.array(first_output)
130         # Найдите индекс наибольшего элемента в массиве предсказаний
```

Листинг В.5 – Метод оценки безопасности водителя. Часть 5

```
131             predicted_class_index = np.argmax(prediction_array)
132             predicted_class = classes[f'c{predicted_class_index}']
133             return predicted_class
134
135
136     # cfg = DriverSafetyGetterConfig(classes_weights, DriverMode.PHOTO,
        ↪  './demo_data/imgs')
137     # getter = DriverSafetyAssessmentGetter(cfg)
138
139     # getter.get_driver_safety_assessment()
```

ПРИЛОЖЕНИЕ Г

Модуль загрузки данных

Листинг Г.1 – Загрузка данных

```
import cv2
import os
from PIL import Image
import matplotlib.pyplot as plt

def get_images_from_video(path_to_video, interval_sec):
    video_capture = cv2.VideoCapture(path_to_video)
    frame_rate = video_capture.get(cv2.CAP_PROP_FPS)
    images = []
    current_frame = 0
    while True:
        success, frame = video_capture.read()
        if not success:
            break
        if current_frame % int(frame_rate * interval_sec) == 0:
            rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)  # Convert to RGB
            pil_image = Image.fromarray(rgb_frame)
            images.append(pil_image)
        current_frame += 1
    video_capture.release()
    return images

def get_images_from_folder(path_to_images):
    images = []
    for filename in os.listdir(path_to_images):
        if filename.endswith(".jpg") or filename.endswith(".png"):
            image_path = os.path.join(path_to_images, filename)
            img = Image.open(image_path)
            images.append(img)
    return images
```

ПРИЛОЖЕНИЕ Д

Презентация