



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 «ФАКУЛЬТЕТ»

КАФЕДРА «КАФЕДРА»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
НА ТЕМУ:

«Метод оценки безопасности водителя с использованием
глубоких нейронных сетей»

Студент группы **ИУ7-83Б**

(Подпись, дата)

Неумоин Д.Ю

(И.О. Фамилия)

Руководитель ВКР

(Подпись, дата)

Кузнецова О.В

(И.О. Фамилия)

Нормоконтролер

(Подпись, дата)

<Нормоконтролер>

(И.О. Фамилия)

РЕФЕРАТ

Расчетно-пояснительная записка 79 с., 27 рис., 1 табл., 18 ист., 4 прил.

В работе представлена разработка метода оценки безопасности водителя с использованием 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 использованием 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

ВВЕДЕНИЕ	9
1 Аналитический раздел	11
1.1 Задача оценки безопасности водителя	11
1.2 Задача классификации изображений	11
1.3 Введение в сверточные нейронные сети	11
1.3.1 Нейрон	11
1.3.2 Многослойный персептрон	12
1.4 Архитектура сверточных нейронных сетей	14
1.4.1 Сверточный слой	14
1.4.2 Слой пулинга	16
1.4.3 Полносвязный слой	18
1.4.4 Функция потерь	18
1.5 Технологии классификации изображений	19
1.5.1 AlexNet	19
1.5.2 VGGNet	21
1.5.3 GoogleNet	23
1.5.4 ResNet	26
1.5.5 Сравнение рассмотренных методов	28
1.6 Формализация постановки задачи	28
2 Конструкторский раздел	31
2.1 Требования к разрабатываемому методу	31
2.2 Требования к разрабатываемому программному комплексу	31
2.3 Особенности метода оценки	31
2.3.1 Модель нейронной сети	31
2.3.2 Общее описание метода оценки	32
2.3.3 Предварительная обработка данных	32
2.3.4 Алгоритм вычисления оценки безопасности	34
2.4 Структура разрабатываемого программного комплекса	36
2.4.1 Описание модулей программного комплекса	36
2.4.2 Архитектура программного комплекса	37
2.4.3 Данные для обучения модели	39

3	Технологический раздел	42
3.1	Средства реализации программного комплекса	42
3.1.1	Выбор языка программирования	42
3.1.2	Выбор библиотеки для работы с нейронной сетью	42
3.2	Реализация программного комплекса	42
3.2.1	Модуль пользовательского интерфейса	42
3.2.2	Модуль GoogleNet модели	44
3.2.3	Обучение модели	44
3.2.4	Результаты обучения модели	46
3.3	Примеры использования разработанного программного комплекса	48
4	Исследовательский раздел	54
4.1	Предмет исследования	54
4.2	Сравнение времени работы реализованного метода на разных объемах входных данных	54
4.3	Сравнение точности модели нейронной сети на изображениях разного размера .	56
4.4	Результаты исследования	57
	ЗАКЛЮЧЕНИЕ	58
	СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	59
	ПРИЛОЖЕНИЕ А	62
	ПРИЛОЖЕНИЕ Б	69
	ПРИЛОЖЕНИЕ В	75
	ПРИЛОЖЕНИЕ Г	79

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, единственным требованием для этого является камера в салоне автомобиля.

Цель работы – разработка метода оценки безопасности водителя.

Для достижения поставленной цели требуется решить следующие задачи:

- описать термины предметной области и формализовать задачу оценки безопасности водителя;
- рассмотреть и сравнить известные технологии, с помощью которых можно реализовать метод оценки безопасности водителя;
- разработать метод оценки безопасности водителя на основе глубоких нейронных сетей;
- разработать программный комплекс, реализующий данный метод;
- провести исследование зависимости точности модели от размера вход-

ных 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 (CNN).

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 рабо-

ты заключается в том, что множество входных значений подвергаются математическому преобразованию для получения выходного значения (рисунок 1.1). Соотношение математического преобразования между входным сигналом и выходным значением представлено формулой:

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

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

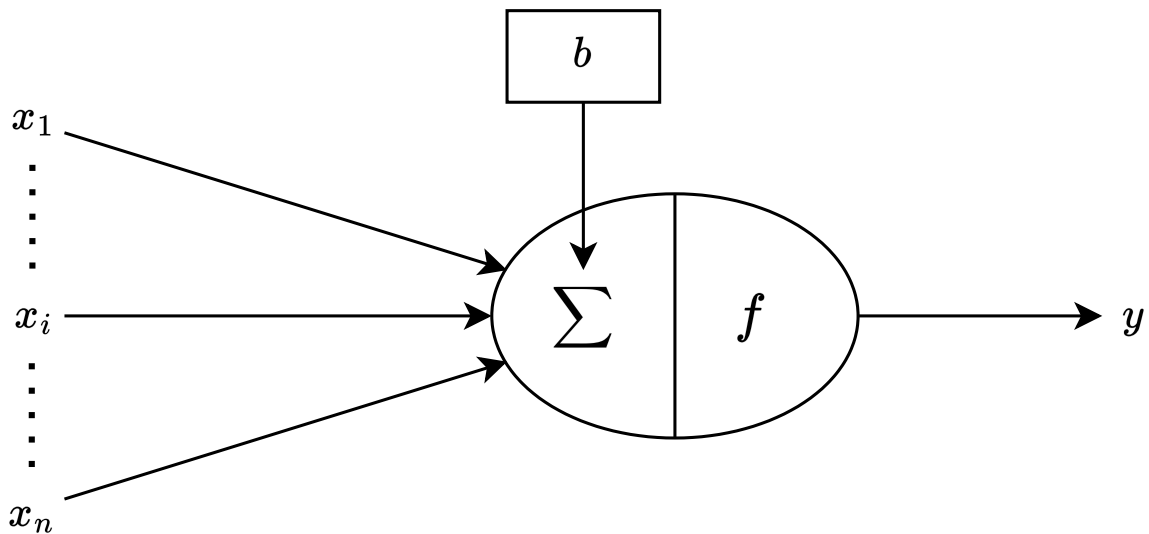


Рисунок 1.1 – Модель нейрона.

1.3.2 Многослойный персептрон

Многослойный персептрон состоит из входного слоя, скрытого слоя (одного или нескольких) и выходного слоя. Он содержит несколько базовых модульных нейронов, которые осуществляют передачу сигнала посредством полой проводимости между нейронами. На рисунке 1.2 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.

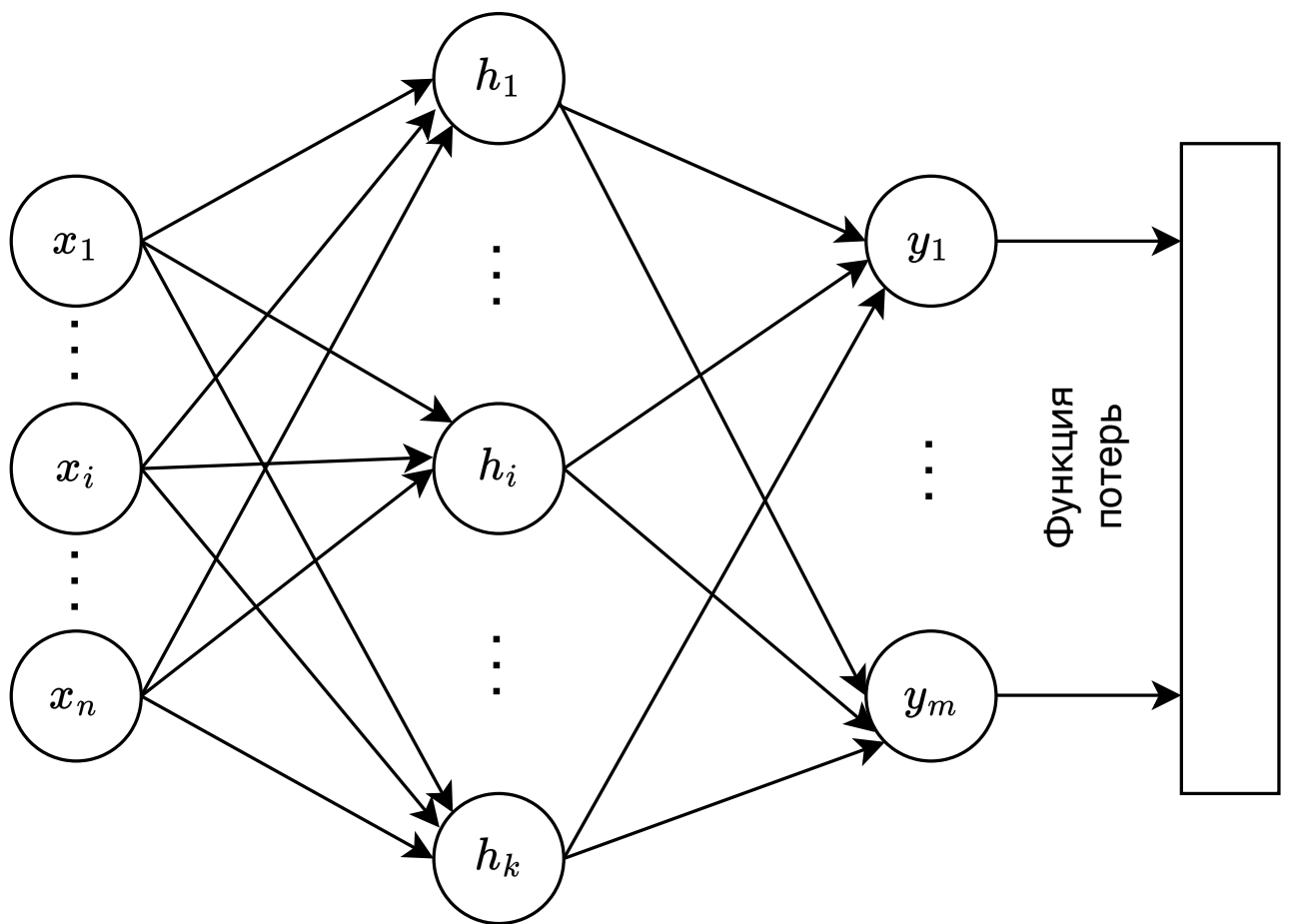


Рисунок 1.2 – Структура многослойного персептрона.

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 количеством 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 1.3 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:

$$feature_surface_{out} = f \left(\sum_{i=1}^3 M_i * W_i + B \right), \quad (2)$$

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.

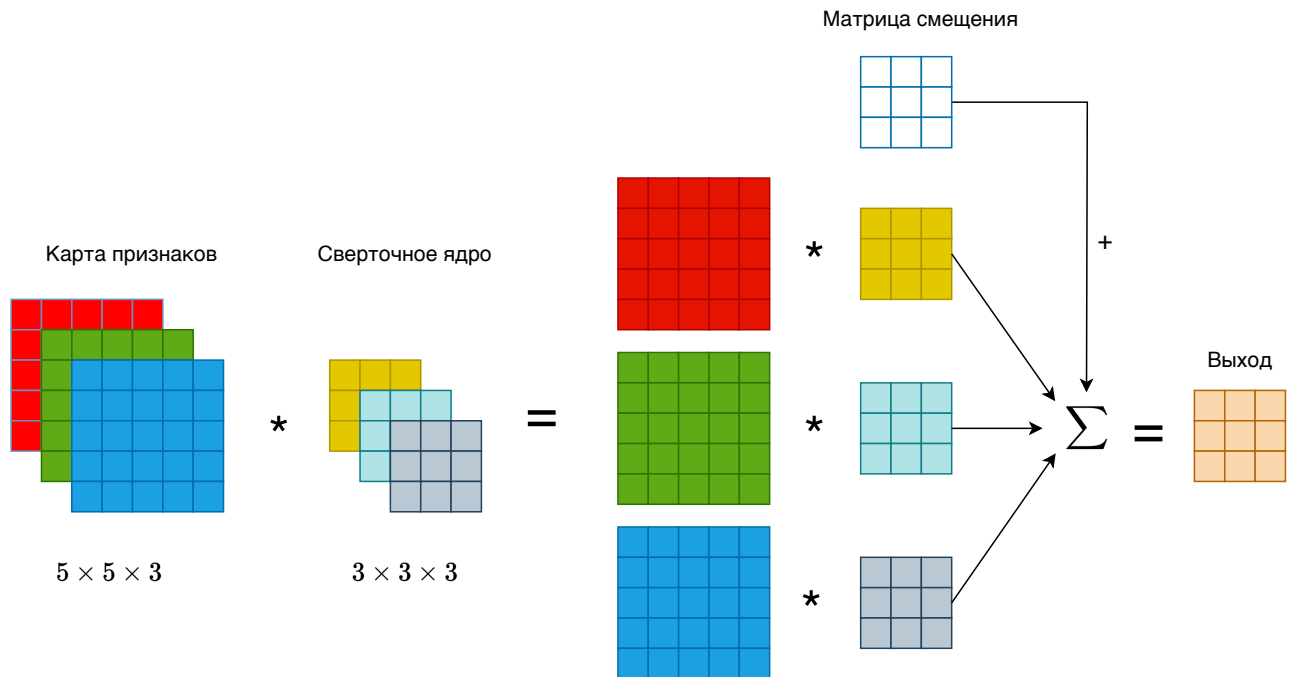


Рисунок 1.3 – Схематичная диаграмма процесса свертки.

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 1.4 показан процесс субдискретизации максимального и среднего пулинга.

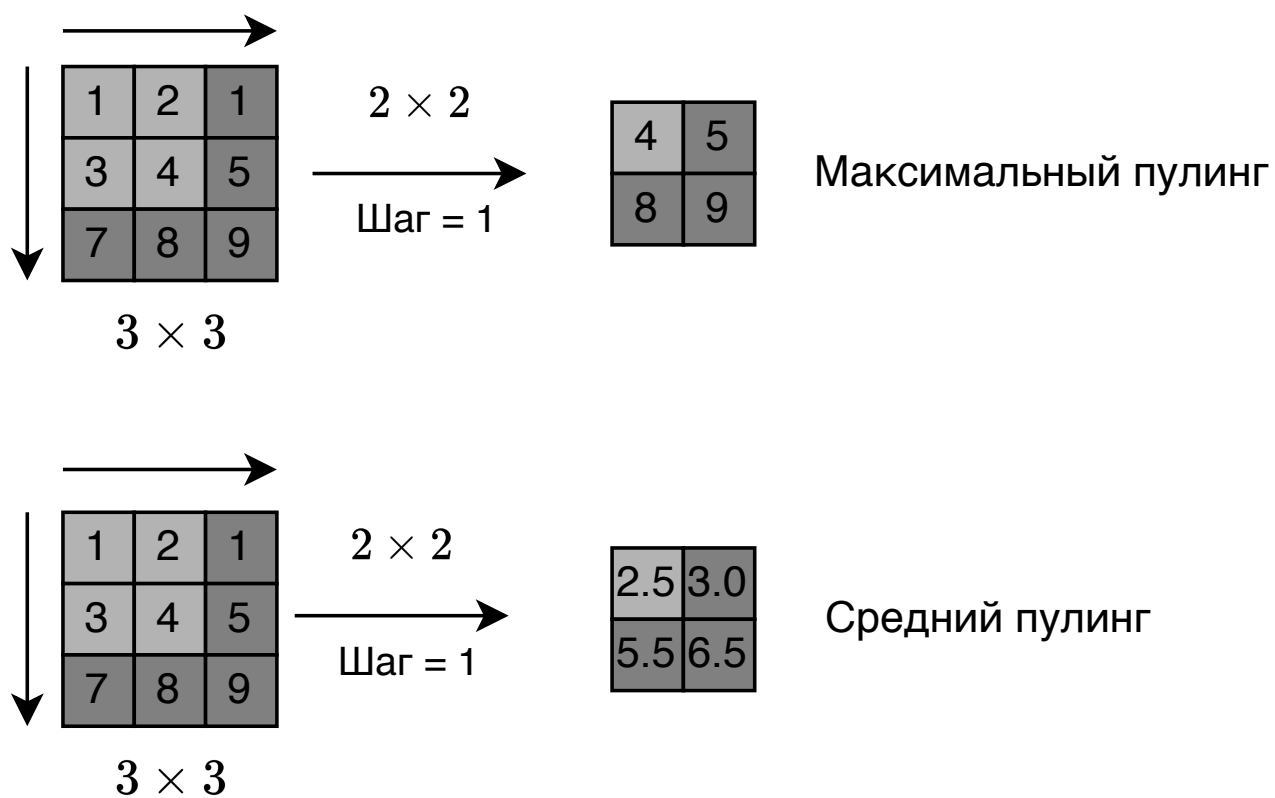


Рисунок 1.4 – Максимальный и средний пулинг.

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. Параметры 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 1.5.

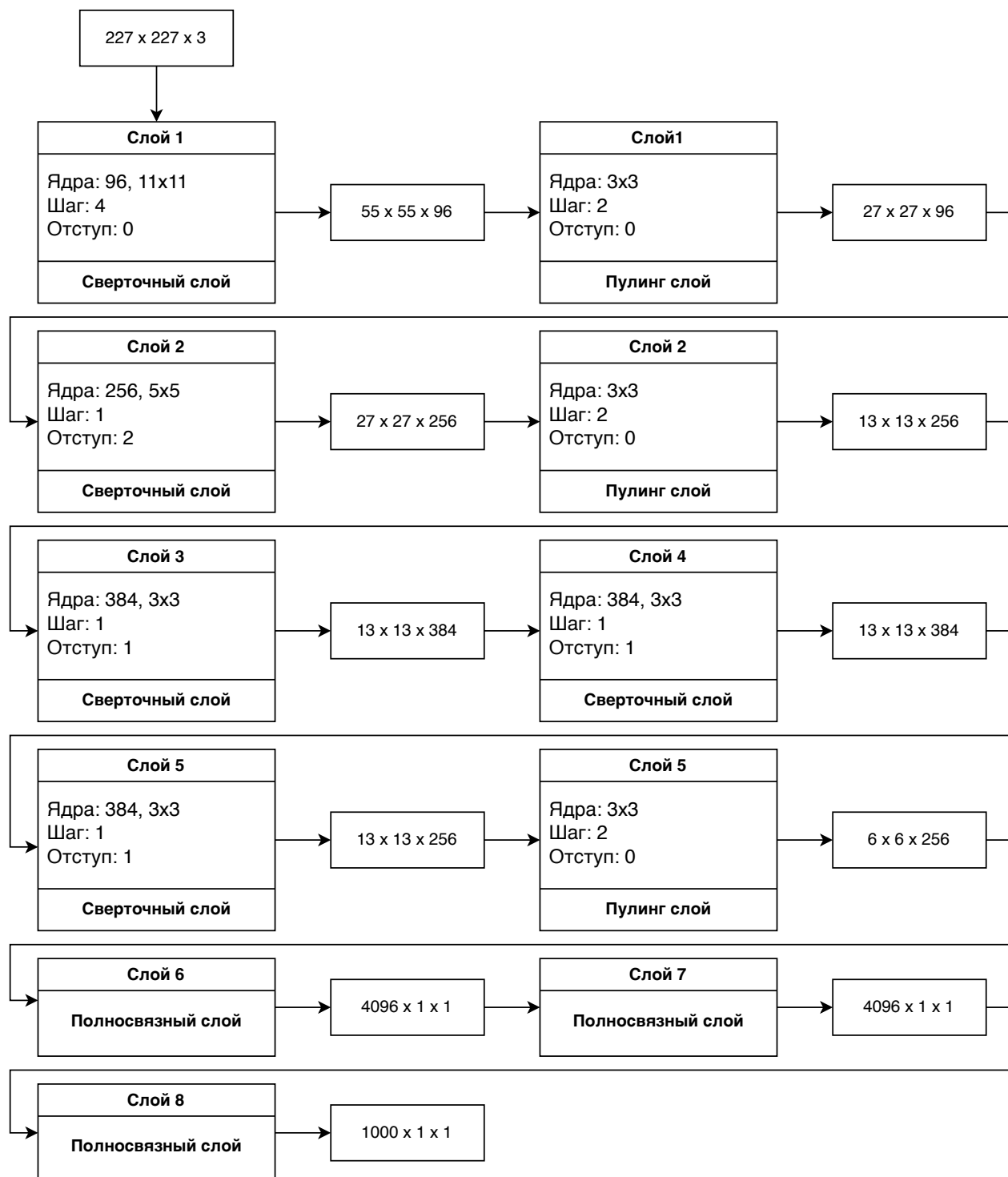


Рисунок 1.5 – Принцип работы AlexNet

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 1.6. 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.

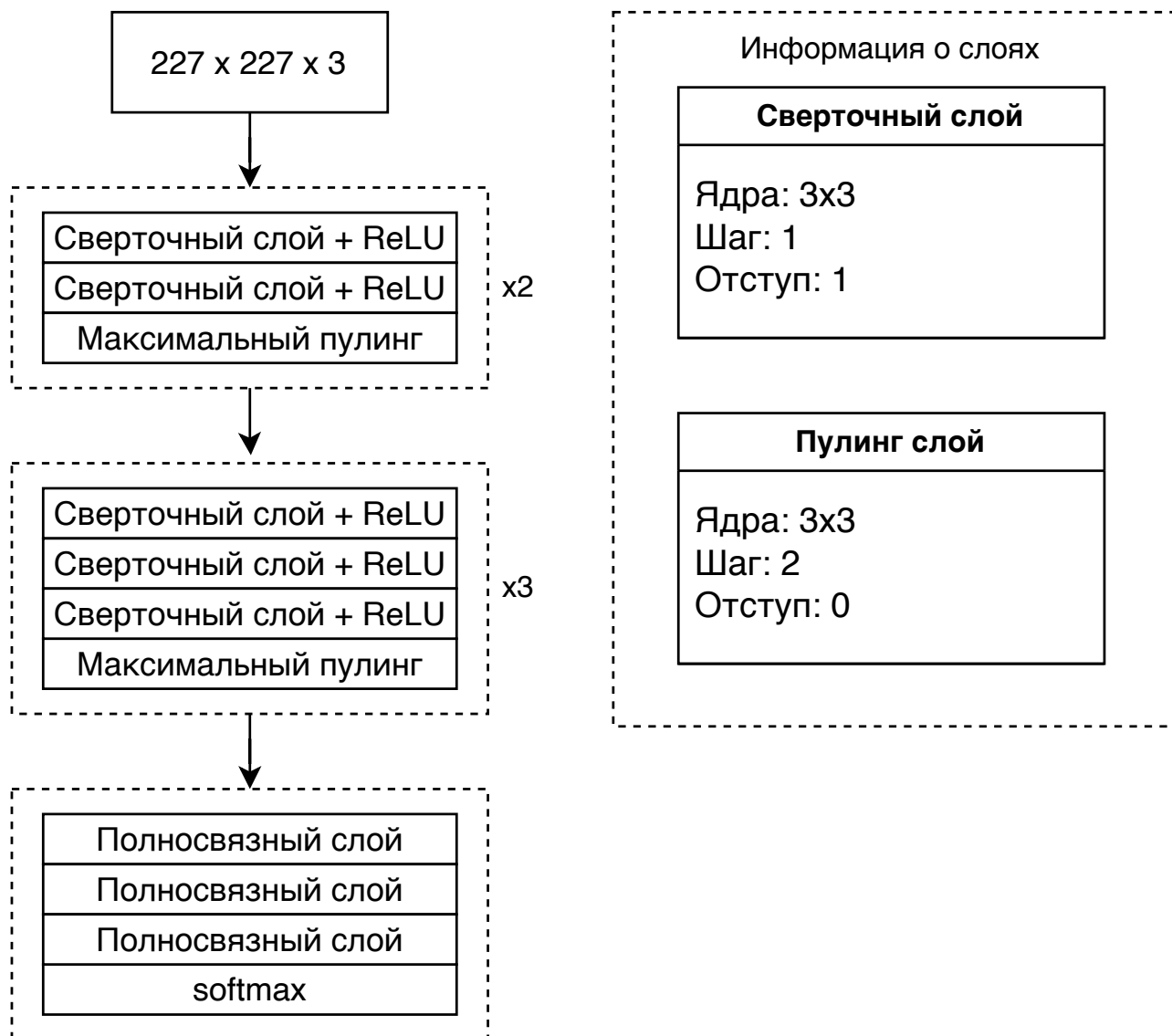


Рисунок 1.6 – Принцип работы VGG-16

1.5.3 GoogleNet

GoogleNet изменила структуру CNN, введя параллельное внутреннее соединение с помощью структуры *Inception*. Как показано на рисунке 1.7, 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: *inceptionV1*, *inceptionV2*, *inceptionV4*, *inceptionV4* [11], 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 1.8 представлен принцип работы GoogleNet, на рисунке присутствуют *Stem* блок и блок вспомогательного классификатора.

Stem блок является фундаментом 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 1.9.

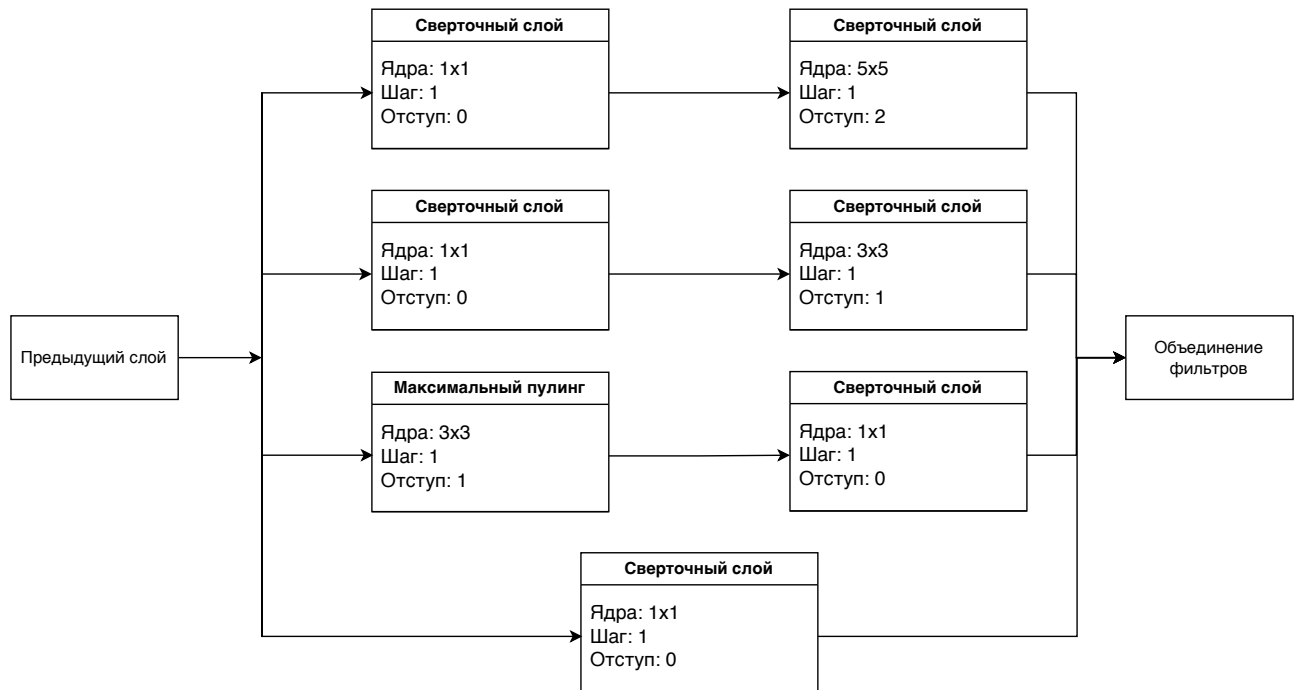


Рисунок 1.7 – Структура *inceptionV1*

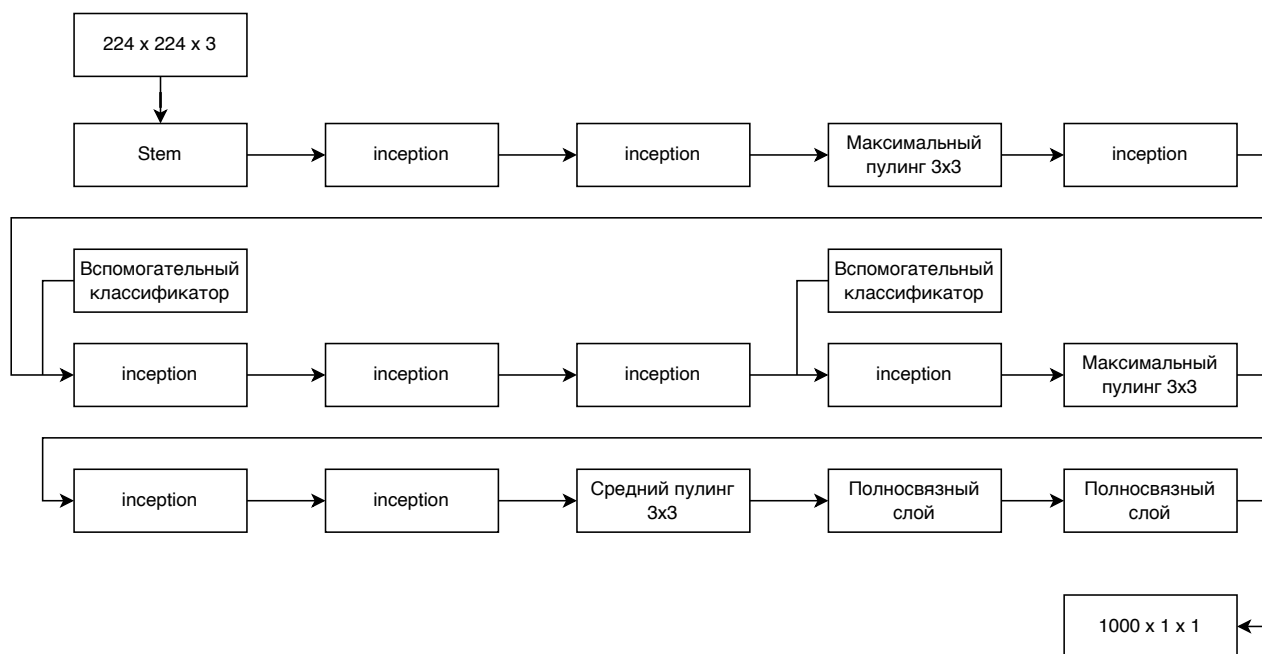


Рисунок 1.8 – Принцип работы GoogleNet

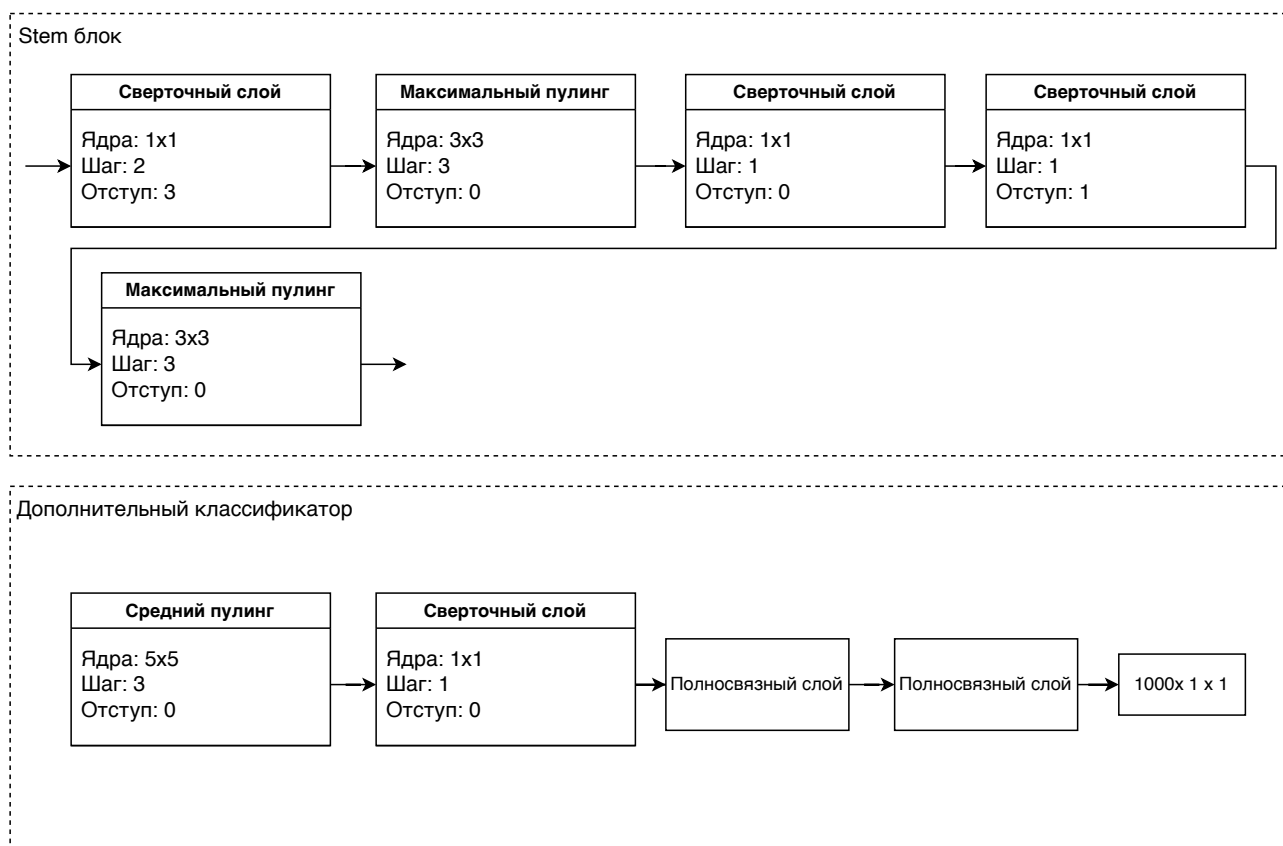


Рисунок 1.9 – Структура *Stem* блока и вспомогательного классификатора

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 1.10, 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.

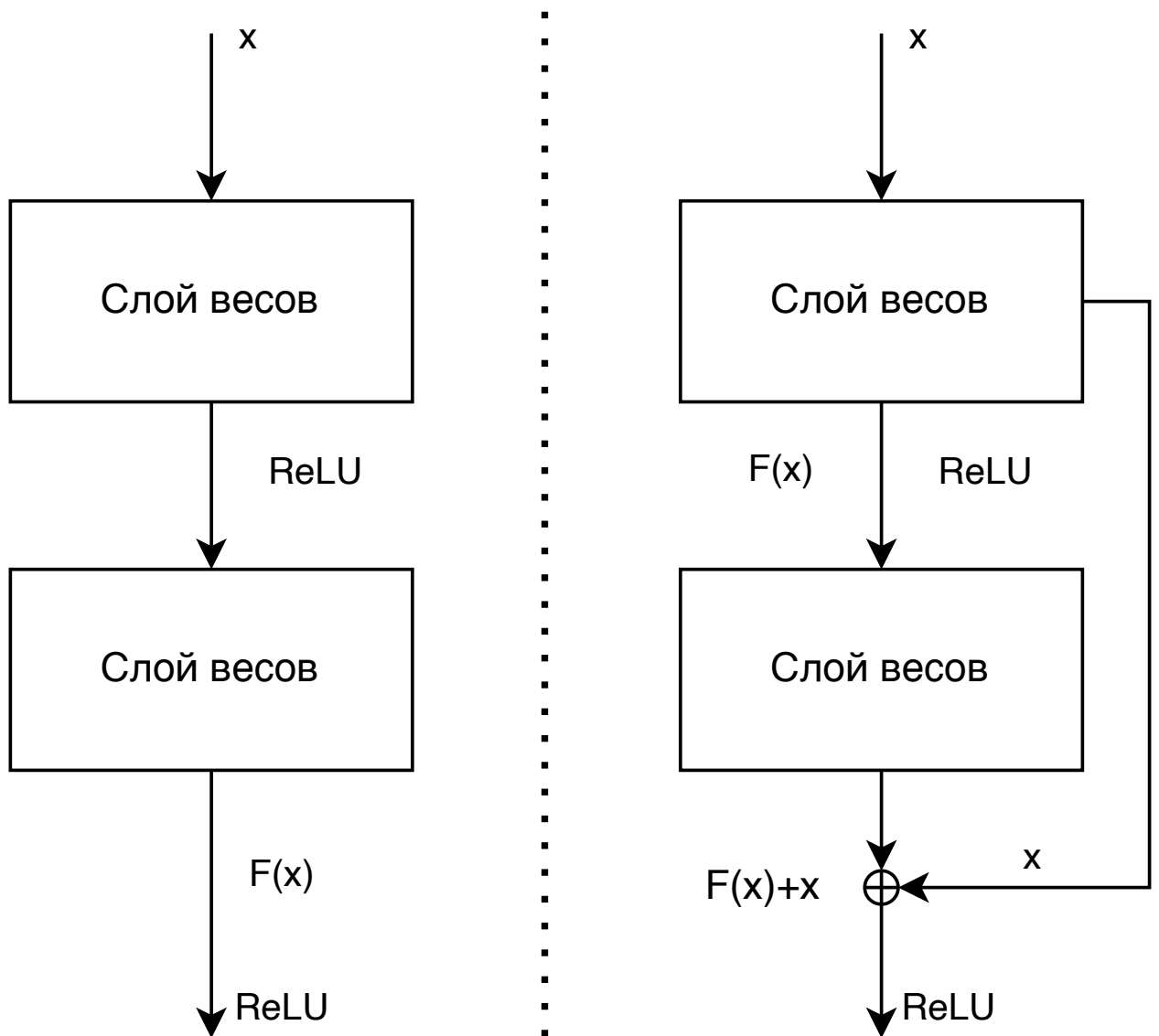


Рисунок 1.10 – Сравнение обычного обучения CNN (слева) и остаточного обучения (справа)

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 1.11.

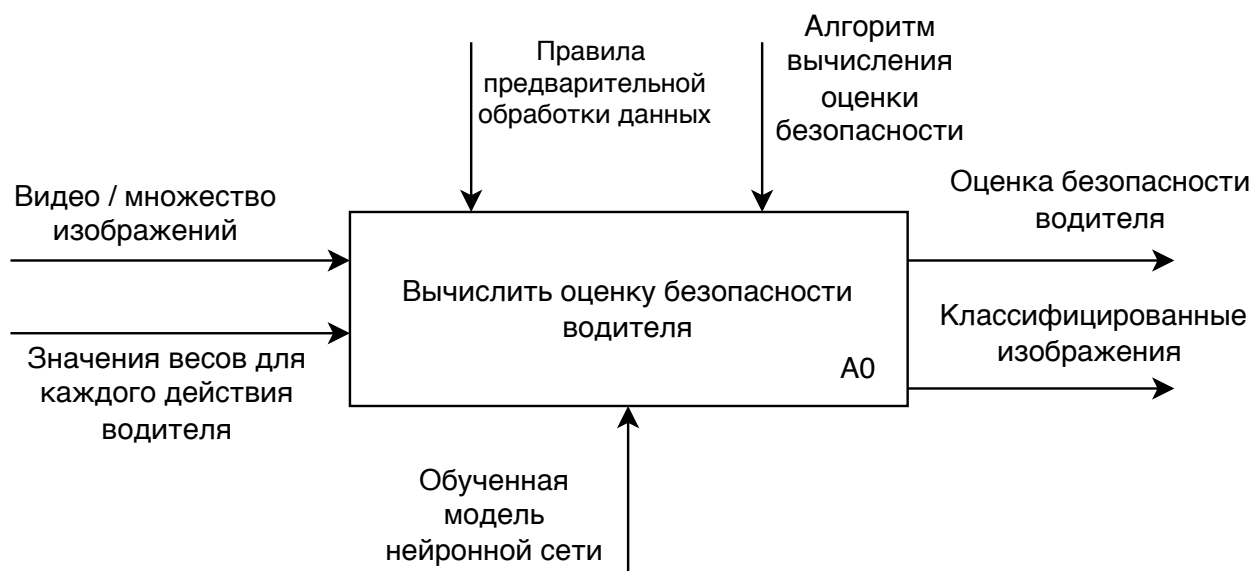


Рисунок 1.11 – Постановка задачи в виде IDEF0-диаграммы нулевого уровня

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 mv4, а также участвующие в оценке веса для каждого действия водителя;
- 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 раз-

меченном 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 2.1.

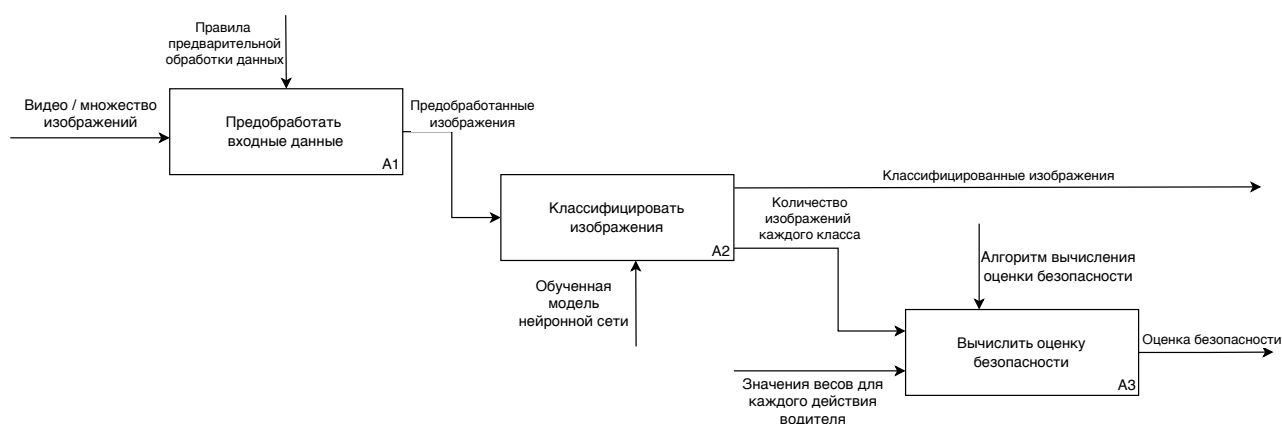


Рисунок 2.1 – Постановка задачи в виде IDEF0-диаграммы первого уровня

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 значений 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 2.2

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

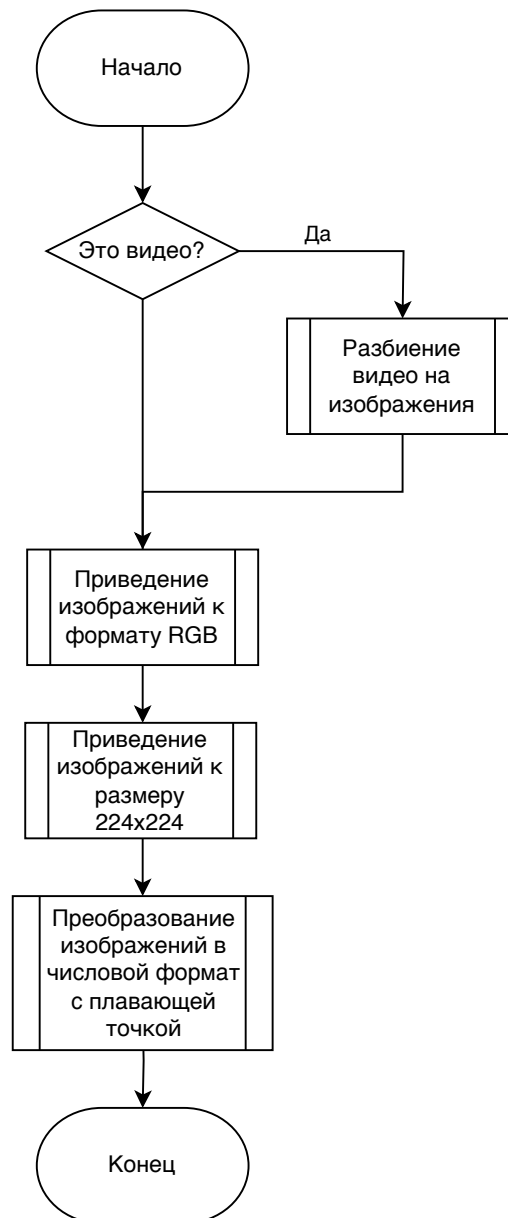


Рисунок 2.2 – Схема алгоритма предварительной обработки данных

2.3.4 Алгоритм вычисления оценки безопасности

Схема алгоритма вычисления оценки безопасности на основе количества изображений каждого класса представлена на рисунке 2.3.

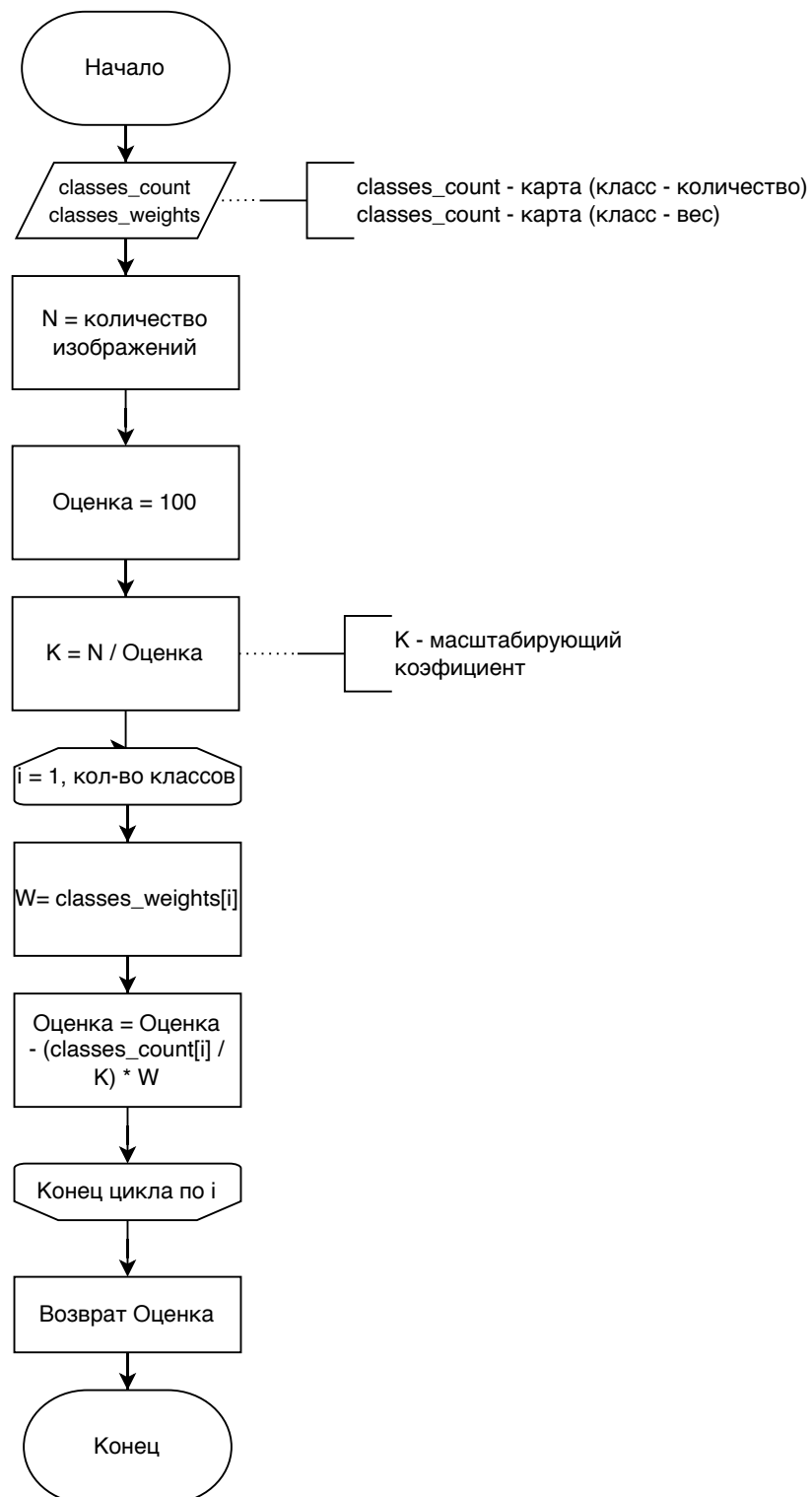


Рисунок 2.3 – Схема алгоритма вычисления оценки безопасности

2.4 Структура разрабатываемого программного комплекса

Программный комплекс 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 2.4 представлена схема работы модуля GoogleNet модели.

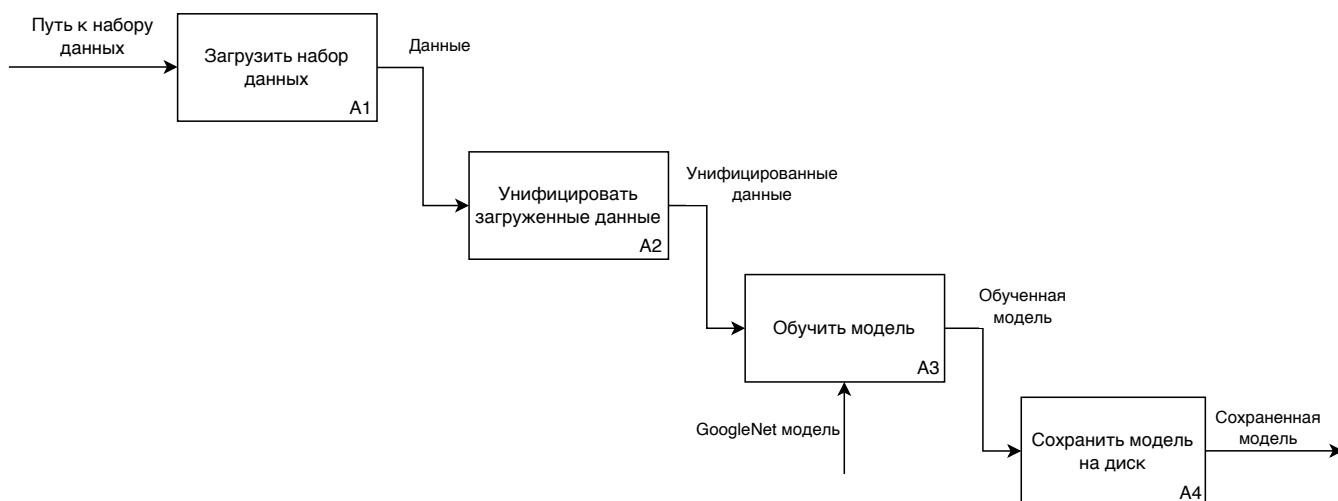


Рисунок 2.4 – Схема работы модуля GoogleNet модели

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.

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 комплекса

Схема архитектуры программного комплекса, реализующего метод оценки безопасности водителя на основе глубоких нейронных сетей, представлена на рисунке 2.5.

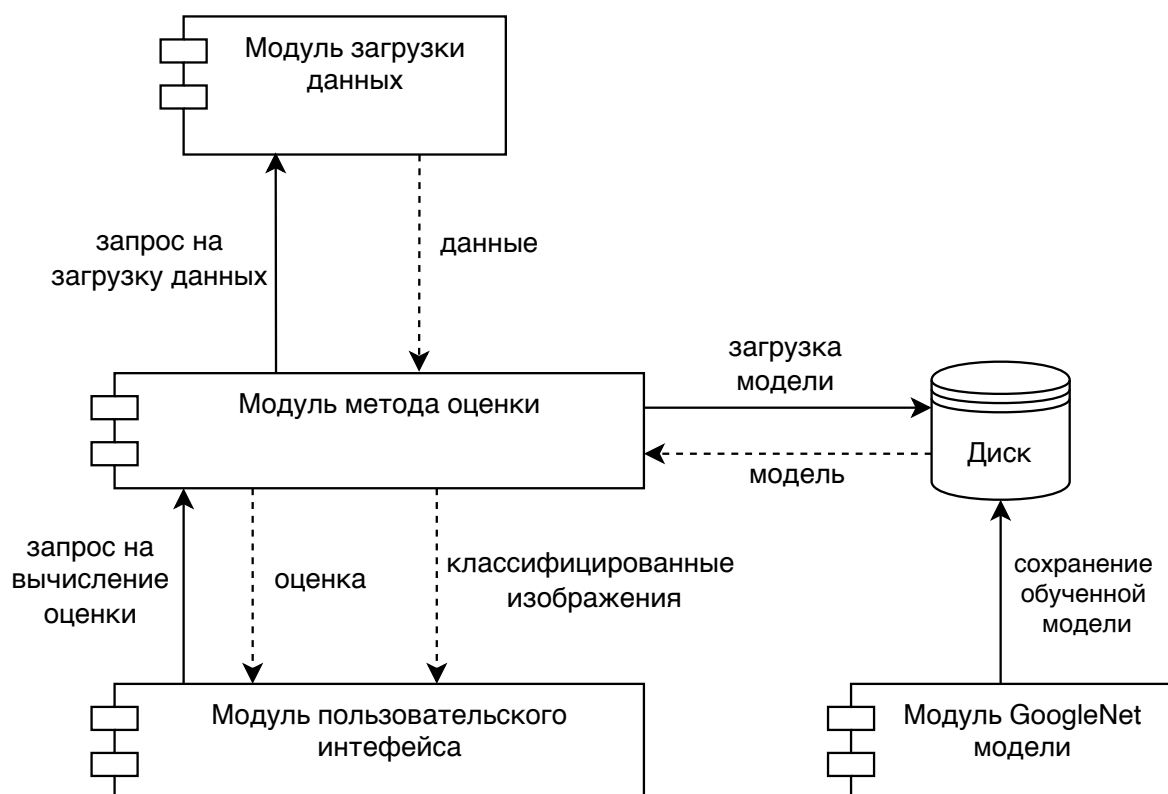


Рисунок 2.5 – Архитектура программного комплекса

2.4.3 Данные для обучения модели

В качестве данных для обучения модели был выбран набор данных[13], состоящий из 22424 фотографий 26ти 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 фотографий для классов «сосредоточен» и «печатает (правая рука)» представлены на рисунке 2.6

Сосредоточен:



Печатает (правая рука):



Рисунок 2.6 – Примеры фотографий из набора данных

Вывод

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

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

Описаны модули разрабатываемого программного комплекса.

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

3 Технологический раздел

3.1 Средства реализации программного комплекса

3.1.1 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.

3.1.2 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 меньше 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 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 3.1.

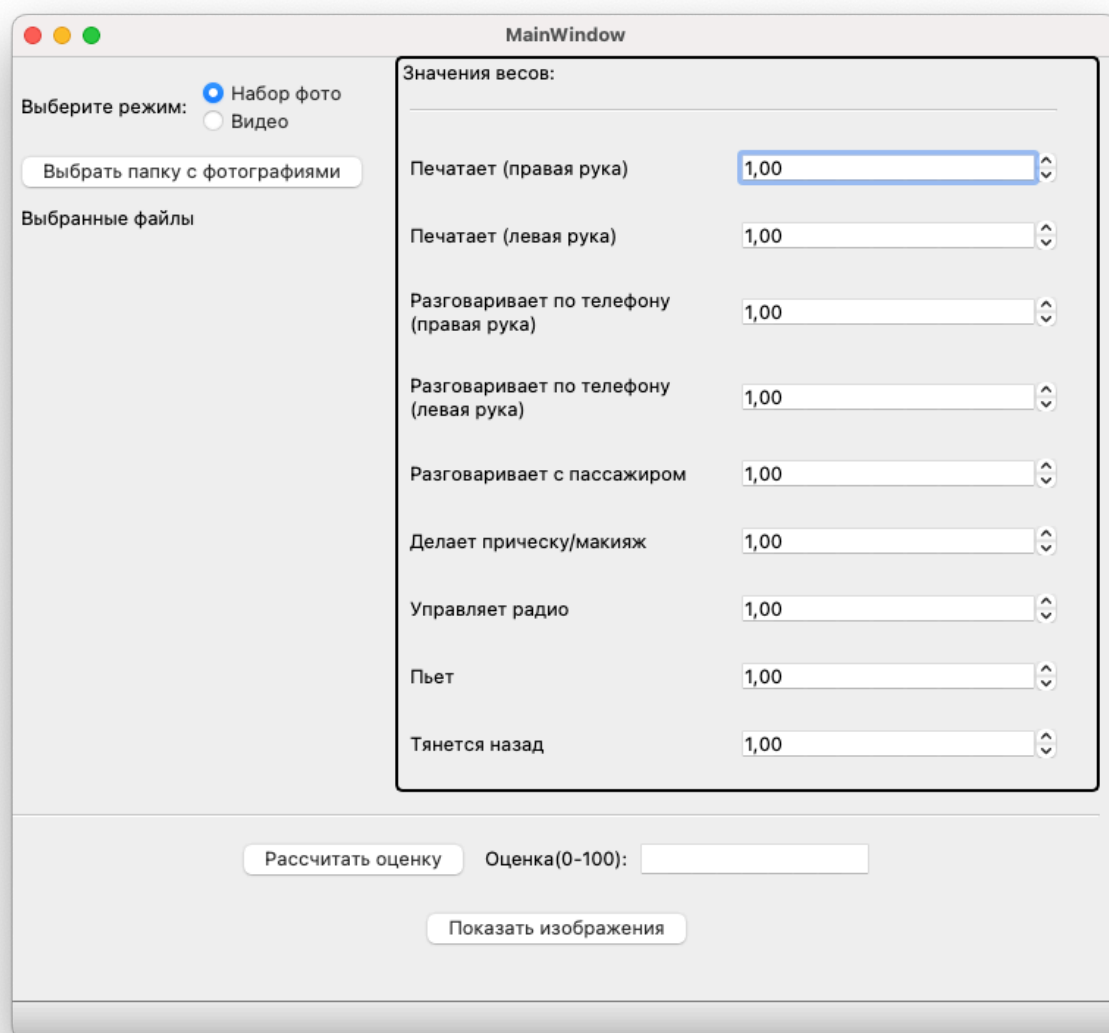


Рисунок 3.1 – Пользовательское приложение

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 A.1 приведена реализация модели GoogleNet, отвечающая за классификацию изображений.

У модели 3 выхода, 1 основной и 2 вспомогательных (засчет дополнительных 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 A.2 приведена реализация процесса обучения реализованной модели GoogleNet. Пример классификации представлен на рисунках 3.2 — 3.3

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



Рисунок 3.2 – Пример классификации, класс «Сосредоточен»



Рисунок 3.3 – Пример классификации, класс «Управляет радио»

эпохи на данных, неучаствующих в обучении.

3.2.4 Результаты обучения модели

На рисунках 3.4 и 3.5 представлены метрики точности на обучающих и валидационных 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.

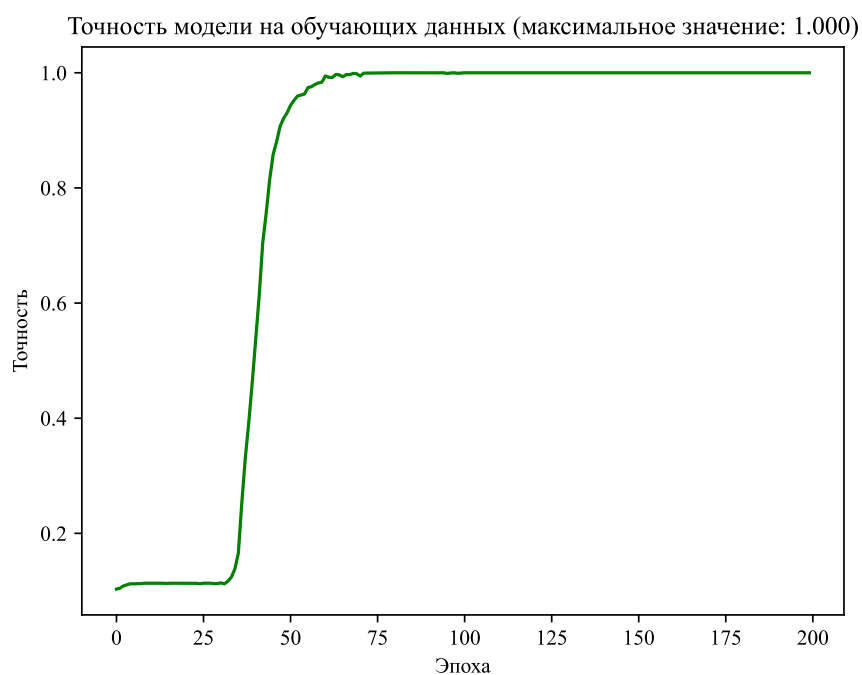


Рисунок 3.4 – Точность модели на обучающих данных

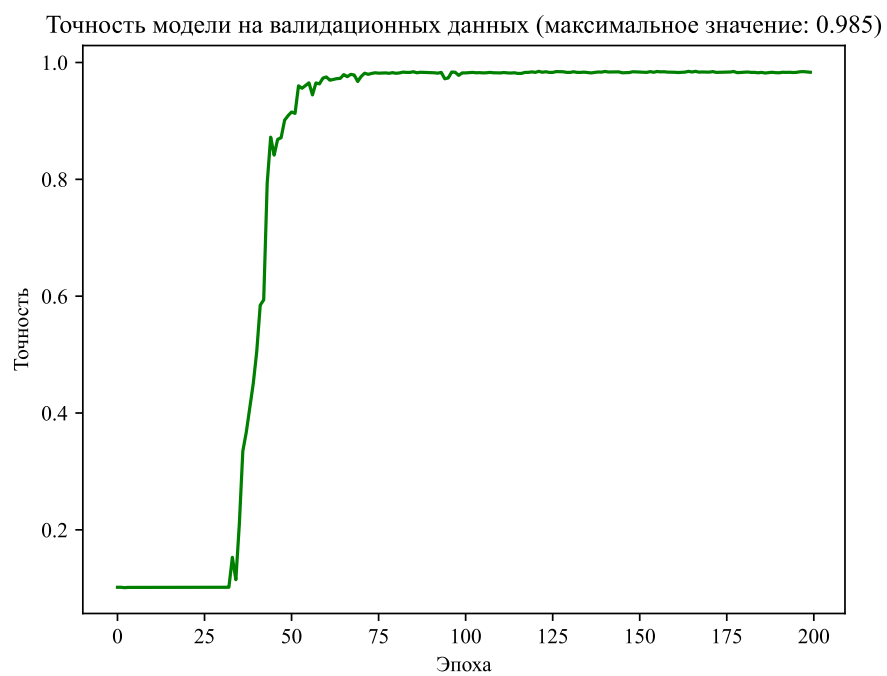


Рисунок 3.5 – Точность модели на валидационных данных

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 листинге 1. Различные параметры, такие как количество эпох, путь до папки с набором данных можно изменить непосредственно в программном коде, они представлены как константы.

Листинг 1: Запуск обучения модели

```
1 python main.py
2
3 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)
4 2024-04-27 01:39:16.375235: I tensorflow/core/platform/profile_utils/
  cpu_utils.cc:112] CPU Frequency: 2693670000 Hz
5 Epoch 1/200
6
7 Epoch 00001: LearningRateScheduler reducing learning rate to 0.01.
8 2024-04-27 01:39:19.291402: I tensorflow/stream_executor/platform/default/
  dso_loader.cc:49] Successfully opened dynamic library libcublas.so.11
9 2024-04-27 01:39:19.771270: I tensorflow/stream_executor/platform/default/
  dso_loader.cc:49] Successfully opened dynamic library libcublasLt.so.11
10 2024-04-27 01:39:22.171898: I tensorflow/stream_executor/platform/default/
  dso_loader.cc:49] Successfully opened dynamic library libcudnn.so.8
11
12 180/180 [=====] - 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
13 Epoch 2/200
14
```

```
15 ...
16
17 Epoch 00200: LearningRateScheduler reducing learning rate to
    0.0036039671685801802.
18 180/180 [=====] - 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 3.6.

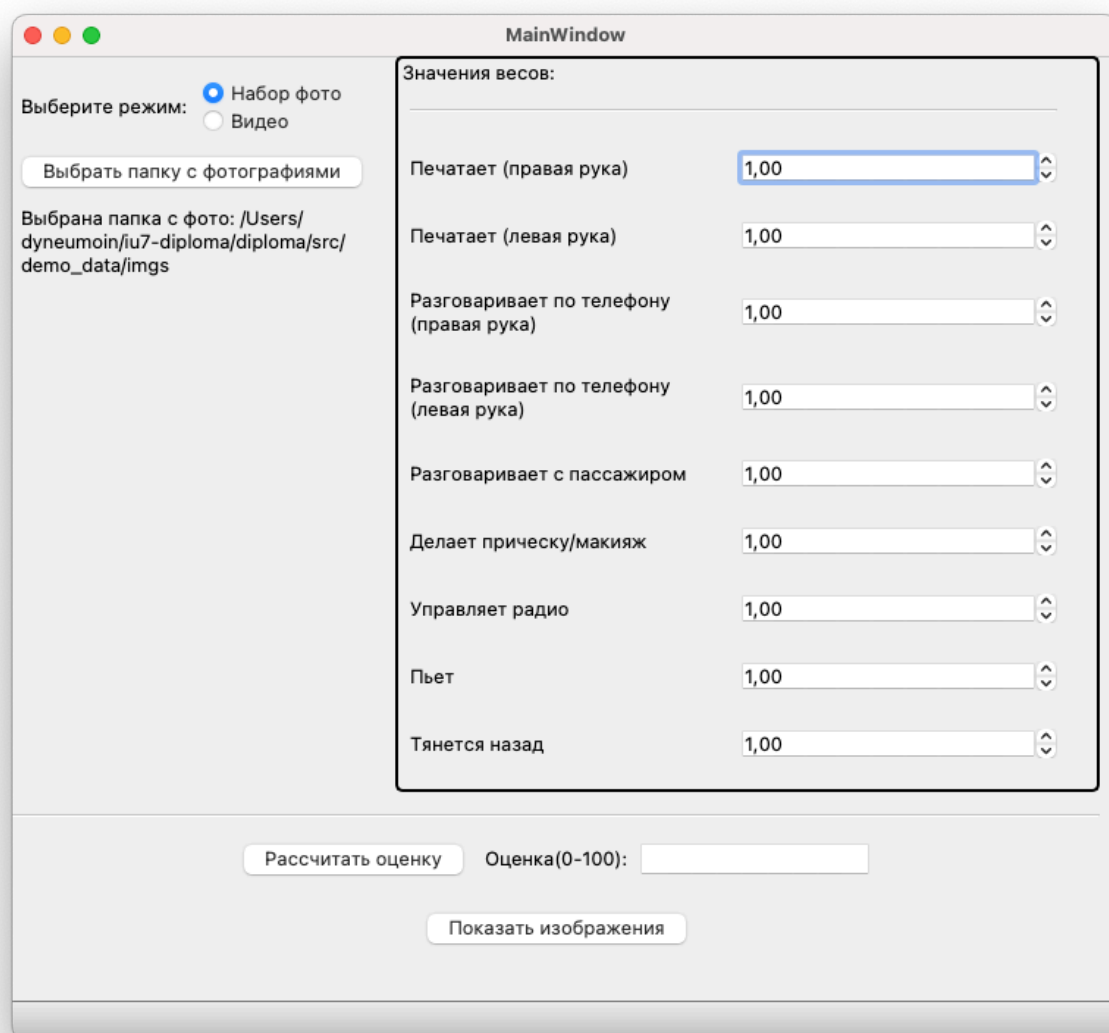


Рисунок 3.6 – Пользовательское приложение (выбран путь к изображениям)

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 3.7.

The screenshot shows a macOS-style window titled "MainWindow". On the left, there are radio buttons for "Набор фото" (selected) and "Видео". Below them is a button "Выбрать папку с фотографиями" and a text field showing the selected folder path: "/Users/dyuneuoin/iu7-diploma/diploma/src/demo_data/imgs". The main area is titled "Значения весов:" and contains a list of activities with corresponding weight input fields, all set to "1,00":

Activity	Weight
Печатает (правая рука)	1,00
Печатает (левая рука)	1,00
Разговаривает по телефону (правая рука)	1,00
Разговаривает по телефону (левая рука)	1,00
Разговаривает с пассажиром	1,00
Делает прическу/макияж	1,00
Управляет радио	1,00
Пьет	1,00
Тянется назад	1,00

At the bottom, there is a button "Рассчитать оценку" and a text field "Оценка(0-100):" containing the value "10.00". Below that is a button "Показать изображения".

Рисунок 3.7 – 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 3.8.

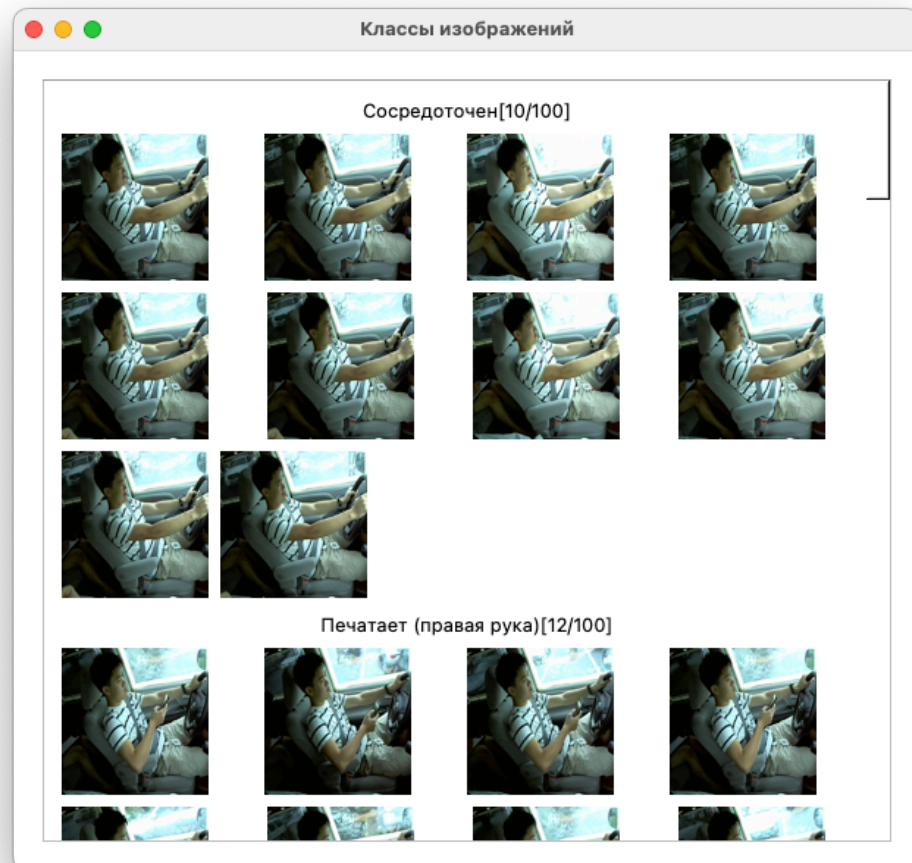


Рисунок 3.8 – Пользовательское приложение (диалоговое окно с изображениями)

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 4.1 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.

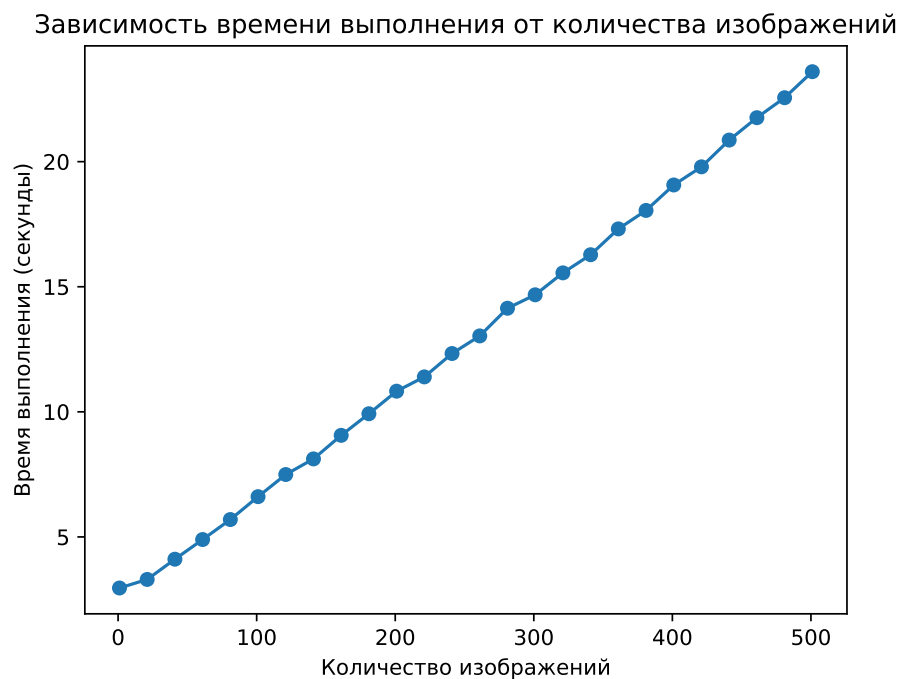


Рисунок 4.1 – Зависимость времени работы метода от количества изображений

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 4.2 приведен график зависимости точности модели от размера изображений.

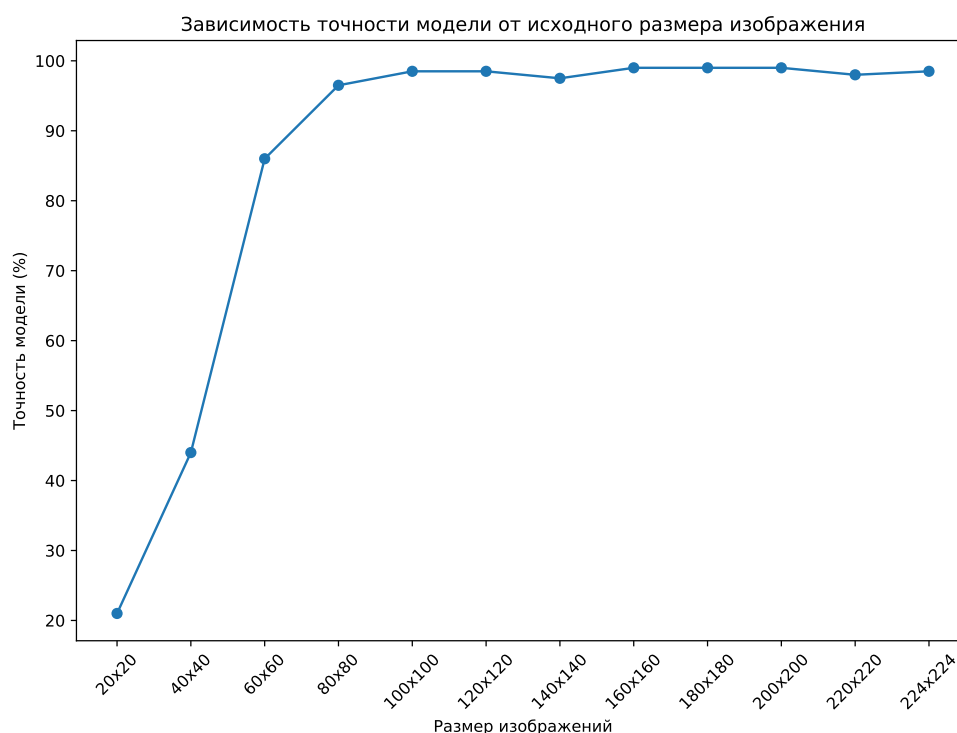


Рисунок 4.2 – Зависимость точности модели от размера изображений

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 позволит 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 известных технологии, с помощью которых можно реализовать метод оценки безопасности водителя;
- разработан метод метод оценки безопасности водителя на основе глубоких нейронных сетей;
- разработан программный комплекс, реализующий 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: <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: <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 // arXiv preprint arXiv:1712.05577. 2017.
13. State Farm Distracted Driver Detection [Электронный ресурс]. Режим доступа: <https://kaggle.com/competitions/state-farm-distracted-driver-detection> (дата обращения 01.04.2024).
14. Python [Электронный ресурс]. Режим доступа: <https://www.python.org/> (дата обращения 01.04.2024).
15. Tensorflow [Электронный ресурс]. Режим доступа: <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: <https://www.tensorflow.org/>

nvidia.com/ru-ru/data-center/dgx-systems/ (дата обращения 01.04.2024).

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

ПРИЛОЖЕНИЕ А

Модуль модели GoogleNet

Листинг А.1: Модель GoogleNet

```
1 import keras
2 from keras.models import Model
3 from keras.layers import Conv2D, MaxPool2D, Dropout, Dense, Input, concatenate
   , GlobalAveragePooling2D, AveragePooling2D, Flatten
4
5 kernel_init = keras.initializers.glorot_uniform()
6 bias_init = keras.initializers.Constant(value=0.2)
7
8 def inception_module(x,
9                       filters_1x1,
10                      filters_3x3_reduce,
11                      filters_3x3,
12                      filters_5x5_reduce,
13                      filters_5x5,
14                      filters_pool_proj,
15                      name=None):
16
17     conv_1x1 = Conv2D(filters_1x1, (1, 1), padding='same', activation='relu',
18                       kernel_initializer=kernel_init, bias_initializer=bias_init)(x)
19
20     conv_3x3_reduce = Conv2D(filters_3x3_reduce, (1, 1), padding='same', activation=
21     'relu', kernel_initializer=kernel_init, bias_initializer=bias_init)(x)
22     conv_3x3 = Conv2D(filters_3x3, (3, 3), padding='same', activation='relu',
23                       kernel_initializer=kernel_init, bias_initializer=bias_init)(conv_3x3_reduce)
24
25     conv_5x5_reduce = Conv2D(filters_5x5_reduce, (1, 1), padding='same', activation=
26     'relu', kernel_initializer=kernel_init, bias_initializer=bias_init)(x)
27     conv_5x5 = Conv2D(filters_5x5, (5, 5), padding='same', activation='relu',
28                       kernel_initializer=kernel_init, bias_initializer=bias_init)(conv_5x5_reduce)
29
30     pool_proj = MaxPool2D((3, 3), strides=(1, 1), padding='same')(x)
31     pool_proj = Conv2D(filters_pool_proj, (1, 1), padding='same', activation=
32     'relu', kernel_initializer=kernel_init, bias_initializer=bias_init)(
33     pool_proj)
```

```

27
28     output = concatenate([conv_1x1, conv_3x3, conv_5x5, pool_proj], axis=3,
29                             name=name)
30
31     return output
32
33 def GoogleNet():
34     input_layer = Input(shape=(224, 224, 3))
35
36     x = Conv2D(64, (7, 7), padding='same', strides=(2, 2), activation='relu',
37                 name='conv_1_7x7_2', kernel_initializer=kernel_init, bias_initializer=
38                 bias_init)(input_layer)
39     x = MaxPool2D((3, 3), padding='same', strides=(2, 2), name='
40     max_pool_1_3x3_2')(x)
41     x = Conv2D(64, (1, 1), padding='same', strides=(1, 1), activation='relu',
42                 name='conv_2a_3x3_1')(x)
43     x = Conv2D(192, (3, 3), padding='same', strides=(1, 1), activation='relu'
44                 , name='conv_2b_3x3_1')(x)
45     x = MaxPool2D((3, 3), padding='same', strides=(2, 2), name='
46     max_pool_2_3x3_2')(x)
47
48     x = inception_module(x,
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

```

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)
75     x1 = Dropout(0.7)(x1)
76     x1 = Dense(10, activation='softmax', name='auxilliary_output_1')(x1)
77
78     x = inception_module(x,
79                           filters_1x1=160,
80                           filters_3x3_reduce=112,
81                           filters_3x3=224,
82                           filters_5x5_reduce=24,
83                           filters_5x5=64,
84                           filters_pool_proj=64,
85                           name='inception_4b')
86
87     x = inception_module(x,
88                           filters_1x1=128,
89                           filters_3x3_reduce=128,
90                           filters_3x3=256,
91                           filters_5x5_reduce=24,
92                           filters_5x5=64,
93                           filters_pool_proj=64,
94                           name='inception_4c')
95
96     x = inception_module(x,
97                           filters_1x1=112,

```

```

98         filters_3x3_reduce=144,
99         filters_3x3=288,
100        filters_5x5_reduce=32,
101        filters_5x5=64,
102        filters_pool_proj=64,
103        name='inception_4d')
104
105
106    x2 = AveragePooling2D((5, 5), strides=3)(x)
107    x2 = Conv2D(128, (1, 1), padding='same', activation='relu')(x2)
108    x2 = Flatten()(x2)
109    x2 = Dense(1024, activation='relu')(x2)
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)
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 A.2: Обучение модели

```

1 import tensorflow as tf
2 import keras
3 from keras.optimizers import SGD
4 from keras.callbacks import LearningRateScheduler
5 import math
6 from model import GoogleNet
7
8 dataset_dir = "./data/train"
9 epochs = 200
10 batch_size = 100
11
12 def get_generators(base_dir_train, target_size=(224, 224)):
13     train_ds = tf.keras.preprocessing.image_dataset_from_directory(
14         base_dir_train,
15         validation_split=0.2,
16         subset="training",
17         seed=123,
18         label_mode='int',
19         image_size=target_size,
20         batch_size=batch_size
21     )
22
23     validation_ds = tf.keras.preprocessing.image_dataset_from_directory(
24         base_dir_train,
25         validation_split=0.2,

```

```

26         subset="validation",
27         label_mode='int',
28         seed=123,
29         image_size=target_size,
30         batch_size=batch_size
31     )
32
33
34     class_names = train_ds.class_names
35     print(class_names)
36     print("before aug: ", train_ds.cardinality().numpy() * batch_size)
37
38     data_augmentation = keras.Sequential(
39         [
40             tf.keras.layers.experimental.preprocessing.RandomFlip("horizontal
41             "),
42             tf.keras.layers.experimental.preprocessing.RandomRotation(0.1),
43             tf.keras.layers.experimental.preprocessing.RandomZoom(0.1),
44         ]
45     )
46
47     train_ds = train_ds.map(lambda x, y: (data_augmentation(x), y),
48                             num_parallel_calls=tf.data.AUTOTUNE)
49
50     normalization_layer = tf.keras.layers.experimental.preprocessing.
51     Rescaling(1./255)
52     normalized_train_ds = train_ds.map(lambda x, y: (normalization_layer(x),
53     y))
54     normalized_validation_ds = validation_ds.map(lambda x, y: (
55     normalization_layer(x), y))
56
57     AUTOTUNE = tf.data.AUTOTUNE
58
59     normalized_train_ds = normalized_train_ds.cache().prefetch(buffer_size=
60     AUTOTUNE)
61     normalized_validation_ds = normalized_validation_ds.cache().prefetch(
62     buffer_size=AUTOTUNE)
63
64     return normalized_train_ds.take(3000), normalized_validation_ds.take(500)

```



```

59 num_classes = 10
60 img_rows, img_cols = 224, 224
61
62 train_ds, test_ds = get_generators(dataset_dir, (img_rows, img_cols))
63
64 initial_lrate = 0.01
65
66 def decay(epoch, steps=100):
67     initial_lrate = 0.01
68     drop = 0.96
69     epochs_drop = 8
70     lrate = initial_lrate * math.pow(drop, math.floor((1+epoch)/epochs_drop))
71     return lrate
72
73
74 sgd = SGD(learning_rate=initial_lrate, momentum=0.9, nesterov=False)
75
76 lr_sc = LearningRateScheduler(decay, verbose=1)
77
78 model = GoogleNet()
79 model.compile(loss=keras.losses.sparse_categorical_crossentropy, loss_weights
    =[1, 0.3, 0.3], optimizer=sgd, metrics=['accuracy'])
80
81 # Model Training
82 train_ds = train_ds.map(lambda x, y: (x, (y, y, y)))
83 test_ds = test_ds.map(lambda x, y: (x, (y, y, y)))
84
85
86 history = model.fit(
87     train_ds,
88     validation_data=test_ds,
89     epochs=epochs,
90     batch_size=batch_size,
91     callbacks=[lr_sc]
92 )
93
94 model.save('model.keras')

```

ПРИЛОЖЕНИЕ Б

Модуль пользовательского интерфейса

Листинг Б.1: Интерфейс главного окна

```
1 from PyQt5 import uic
2 from PyQt5.QtWidgets import (QApplication, QMainWindow, QPushButton, QLabel,
    QFileDialog, QDoubleSpinBox, QLineEdit, QRadioButton)
3 import sys
4 import app.main
5 import images_window
6
7 class MainWindow(QMainWindow):
8     def __init__(self):
9         super(MainWindow, self).__init__()
10        self.ui = uic.loadUi("./ui/untitled.ui", self)
11
12        # Определим виджеты.
13        self.photo_button = self.findChild(QPushButton, 'choice_photo')
14        self.video_button = self.findChild(QPushButton, 'choice_video')
15        self.mode_photo_radioButton = self.findChild(QRadioButton, '
mode_photo_radioButton')
16        self.mode_video_radioButton = self.findChild(QRadioButton, '
mode_video_radioButton')
17        self.calc_button = self.findChild(QPushButton, 'calc')
18        self.files_label = self.findChild(QLabel, 'choiced_files_label')
19        self.assessment = self.findChild(QLineEdit, 'assessment_line')
20        self.show_images_button = self.findChild(QPushButton, '
show_images_button')
21
22        # Beca
23        self.texting_right_spin = self.findChild(QDoubleSpinBox, '
texting_right_spin')
24        self.talking_right_spin = self.findChild(QDoubleSpinBox, '
talking_right_spin')
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(self.on_radio_toggled)
38         self.mode_video_radioButton.toggled.connect(self.on_radio_toggled)
39         self.on_radio_toggled()
40         self.calc_button.clicked.connect(self.calc_button_clicked)
41         self.show_images_button.clicked.connect(self.
show_images_button_clicked)
42
43         # Контекст.
44         self.mode = None
45         self.file_path = ''
46         self.classes_images = []
47
48     def photo_button_clicked(self):
49         folder_dialog = QFileDialog()
50         folder_path = folder_dialog.getExistingDirectory(self, 'Выбрать
папку с фото')
51
52         if folder_path:
53             self.files_label.setText("Выбрана папка с фото: " + folder_path)
54             self.mode = app.main.DriverMode.PHOTO
55             self.file_path = folder_path
56
57     def video_button_clicked(self):
58         file_dialog = QFileDialog()
59         file_path, _ = file_dialog.getOpenFileName(self, 'Выбрать видео', '',
'Video files (*.mp4 *.avi)')

```

```

60
61     if file_path:
62         self.files_label.setText("Выбрано видео: " + file_path)
63         self.mode = app.main.DriverMode.VIDEO
64         self.file_path = file_path
65
66     def calc_button_clicked(self):
67         classes_weights = self.__get_classes_weights()
68         print(f'calc: file:{self.file_path}, weights:{classes_weights}')
69         cfg = app.main.DriverSafetyGetterConfig(classes_weights, self.mode,
self.file_path)
70         getter = app.main.DriverSafetyAssessmentGetter(cfg)
71         assessment = getter.get_driver_safety_assessment()
72         print(f'assesment: {assessment}')
73         self.assessment.setText(f"{assessment:.2f}")
74         self.classes_images = getter.get_classes_images()
75
76     def on_radio_toggled(self):
77         if self.mode_photo_radioButton.isChecked():
78             self.photo_button.setVisible(True)
79             self.video_button.setVisible(False)
80         elif self.mode_video_radioButton.isChecked():
81             self.photo_button.setVisible(False)
82             self.video_button.setVisible(True)
83
84     def __get_classes_weights(self):
85         classes_weights= {
86             'texting - right': self.texting_right_spin.value(),
87             'talking on the phone - right': self.talking_right_spin.value(),
88             'texting - left': self.texting_left_spin.value(),
89             'talking on the phone - left': self.talking_left_spin.value(),
90             'operating the radio': self.operating_radio_spin.value(),
91             'drinking': self.drinking_spin.value(),
92             'reaching behind': self.reaching_begind_spin.value(),
93             'hair and makeup': self.hair_and_makeup_spin.value(),
94             'talking to passenger': self.talking_to_passenger.value(),
95         }
96
97     return classes_weights
98

```

```

99     def show_images_button_clicked(self):
100         self.img_window = images_window.GalleryWindow(self.classes_images)
101         self.img_window.show()
102
103     def main():
104         app = QApplication(sys.argv)
105         window = MainWindow()
106         window.show()
107         return app.exec()
108
109
110 if __name__ == '__main__':
111     sys.exit(main())

```

Листинг Б.2: Интерфейс диалогового окна с изображениями

```

1 import sys
2 from PyQt5.QtWidgets import QApplication, QWidget, QPushButton, QVBoxLayout,
    QLabel, QScrollArea, QHBoxLayout, QMainWindow, QSpacerItem, QSizePolicy
3 from PyQt5.QtGui import QPixmap, QImage
4 from PyQt5.QtCore import Qt
5 from PIL import Image
6 from io import BytesIO
7
8 classes_en_ru_mapping = {
9     'safe driving': 'осредоточенС',
10    'texting - right': 'Печатает правая( рука)',
11    'talking on the phone - right': 'Разговаривает по телефону правая( рука)
12    ',
13    'texting - left': 'Печатает левая( рука)',
14    'talking on the phone - left': 'Разговаривает по телефону левая( рука)',
15    'operating the radio': 'Управляет радио',
16    'drinking': 'Пьет',
17    'reaching behind': 'Тянется назад',
18    'hair and makeup': 'Делает прическумакияж/',
19    'talking to passenger': 'Разговаривает с пассажиром',
20 }
21
22 class GalleryWindow(QWidget):
23     def __init__(self, images):

```

```

24         super().__init__()
25         self.images = images # Словарь с изображениями
26         self.total_elements = sum(len(images) for images in self.images.
values())
27         self.initUI()
28
29     def initUI(self):
30         self.setStyleSheet("background-color: white;")
31         layout = QVBoxLayout()
32
33         scroll = QScrollArea() # Создание области прокрутки
34         widget = QWidget() # Главный виджет, содержащий все остальные
виджеты
35         scroll.setWidget(widget) # Установка виджета в качестве содержимого
области прокрутки
36         scroll.setWidgetResizable(True) # Разрешение изменения размера
виджета
37
38         vbox = QVBoxLayout() # Вертикальное расположение для всех элементов
39
40         for class_name, imgs in self.images.items(): # Перебор всех классов
изображений
41             label = QLabel(f'{classes_en_ru_mapping[class_name]}[{len(imgs)
}]/[{self.total_elements}]') # Создание метки для названия класса
42             label.setAlignment(Qt.AlignCenter) # Центрирование текста
43             vbox.addWidget(label) # Добавление метки в вертикальное
расположение
44
45             imgs_in_row = 4 # Количество изображений в одной строке
46             row_counter = 0 # Счетчик для отслеживания количества
изображений в текущей строке
47             hbox = QHBoxLayout() # Горизонтальное расположение для
изображений
48             hbox.setSpacing(0) # Устанавливаем межэлементное расстояние
равным 0
49             hbox.setContentsMargins(0, 0, 0, 0) # Убираем отступы
50
51             for img in imgs: # Перебор изображений текущего класса
52                 if row_counter >= imgs_in_row: # Если достигнут предел
изображений в строке

```

```

53         vbox.addLayout(hbox) # Добавляем текущее горизонтальное
расположение в вертикальное
54         hbox = QHBoxLayout() # Создаем новое горизонтальное
расположение для следующей строки
55         row_counter = 0 # Сброс счетчика изображений в строке
56
57         pixmap = QPixmap.fromImage(self.convert_pil_image_to_qimage(
img)) # Создание изображения
58         img_label = QLabel(self) # Создание метки для изображения
59         img_label.setPixmap(pixmap) # Установка изображения в метку
60         hbox.addWidget(img_label) # Добавление метки в
горизонтальное расположение
61         row_counter += 1 # Увеличение счетчика изображений в строке
62
63         if row_counter > 0 and row_counter < 4:
64             spacer = QSpacerItem(100, 100, QSizePolicy.Expanding,
QSizePolicy.Minimum)
65             hbox.addSpacerItem(spacer)
66             if row_counter > 0: # Проверка, есть ли необработанные
изображения после выхода из цикла
67                 vbox.addLayout(hbox) # Добавление последнего
горизонтального расположения в вертикальное
68
69         widget.setLayout(vbox) # Установка вертикального расположения в
качестве основного для виджета
70         layout.addWidget(scroll) # Добавление области прокрутки в основное
вертикальное расположение
71         self.setLayout(layout) # Установка основного вертикального
расположения для self
72         self.setWindowTitle('Классы изображений') # Задание заголовка окна
73         self.resize(600, 400) # Установка начального размера окна
74
75     def convert_pil_image_to_qimage(self, pil_image):
76         pil_image = pil_image.resize((100, 100))
77         image_data = BytesIO()
78         pil_image.save(image_data, format='PNG')
79         qimage = QImage()
80         qimage.loadFromData(image_data.getvalue())
81         return qimage

```

ПРИЛОЖЕНИЕ В

Модуль метода оценки безопасности водителя

Листинг В.1: Метод оценки безопасности водителя

```
1 from enum import Enum
2
3 from . import get_images
4 import tensorflow as tf
5 import numpy as np
6
7 import matplotlib.pyplot as plt
8
9 classes = {
10     'c0': 'safe driving',
11     'c1': 'texting - right',
12     'c2': 'talking on the phone - right',
13     'c3': 'texting - left',
14     'c4': 'talking on the phone - left',
15     'c5': 'operating the radio',
16     'c6': 'drinking',
17     'c7': 'reaching behind',
18     'c8': 'hair and makeup',
19     'c9': 'talking to passenger'
20 }
21
22 default_classes_weights= {
23     'texting - right': 1,
24     'talking on the phone - right': 1,
25     'texting - left': 1,
26     'talking on the phone - left': 1,
27     'operating the radio': 1,
28     'drinking': 1,
29     'reaching behind': 1,
30     'hair and makeup': 1,
31     'talking to passenger': 1,
32 }
33
34 class DriverMode(Enum):
```



```

35     VIDEO = "video"
36     PHOTO = "photo"
37
38     class DriverSafetyGetterConfig:
39         def __init__(self, classes_weights, mode: DriverMode, path_to_files,
40             interval_sec=1):
41             self.classes_weights = classes_weights
42             self.mode = mode
43             self.path_to_files = path_to_files
44             self.interval_sec = interval_sec if mode == DriverMode.VIDEO else
45             None
46
47     class DriverSafetyAssessmentGetter:
48         def __init__(self, cfg: DriverSafetyGetterConfig):
49             self.cfg = cfg
50             self.model = tf.keras.models.load_model('./working/aug.keras')
51
52             self.images = []
53             self.input_datas = []
54             self.classes_count= {
55                 'safe driving': 0,
56                 'texting - right': 0,
57                 'talking on the phone - right': 0,
58                 'texting - left': 0,
59                 'talking on the phone - left': 0,
60                 'operating the radio': 0,
61                 'drinking': 0,
62                 'reaching behind': 0,
63                 'hair and makeup': 0,
64                 'talking to passenger': 0,
65             }
66
67             self.classes_images = {
68                 'safe driving': [],
69                 'texting - right': [],
70                 'talking on the phone - right': [],
71                 'texting - left': [],
72                 'talking on the phone - left': [],
73                 'operating the radio': [],
74                 'drinking': [],
75                 'reaching behind': [],

```

```

73         'hair and makeup': [],
74         'talking to passenger': [],
75     }
76
77     def get_driver_safety_assessment(self, limit=100): #TODO убрать лимит
после исследовательской
78         self.__set_images()
79         self.__prepare_input_datas()
80         self.input_datas = self.input_datas[:limit] # TODO: убрать
81
82         for i, input_data in enumerate(self.input_datas):
83             prediction = self.model.predict(input_data)
84             predicted_class = self.__get_class_by_prediction(prediction)
85             self.classes_images[predicted_class].append(self.images[i])
86             self.classes_count[predicted_class] += 1
87
88         # for image in classes_images['talking to passenger']:
89         #     plt.imshow(image)
90         #     plt.show()
91
92         return self.__get_assessment_by_classes_count()
93
94     def get_classes_images(self):
95         return self.classes_images
96
97     def __get_assessment_by_classes_count(self):
98         print(self.classes_count)
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(path_to_video=cfg.
path_to_files, interval_sec=cfg.interval_sec)
110         elif cfg.mode == DriverMode.PHOTO:

```

```

111         self.images = get_images.get_images_from_folder(path_to_images=
cfg.path_to_files)
112
113     def __prepare_input_datas(self):
114         input_datas = []
115         for i in self.images:
116             prepared_image = i.resize((224, 224))
117             if prepared_image.mode != 'RGB':
118                 prepared_image = prepared_image.convert('RGB')
119             input_data = tf.expand_dims(tf.image.convert_image_dtype(
prepared_image, tf.float32), 0)
120             input_datas.append(input_data)
121         self.input_datas = input_datas
122
123     def __get_class_by_prediction(self, prediction):
124         first_output = prediction[0][0]
125         prediction_array = np.array(first_output)
126         # Найдите индекс наибольшего элемента в массиве предсказаний
127         predicted_class_index = np.argmax(prediction_array)
128         predicted_class = classes[f'c{predicted_class_index}']
129         return predicted_class
130
131
132 # cfg = DriverSafetyGetterConfig(classes_weights, DriverMode.PHOTO, './
demo_data/imgs')
133 # getter = DriverSafetyAssessmentGetter(cfg)
134
135 # getter.get_driver_safety_assessment()

```

ПРИЛОЖЕНИЕ Г

Модуль загрузки данных

Листинг Г.1: Загрузка данных

```
1 import cv2
2 import os
3 from PIL import Image
4
5 import matplotlib.pyplot as plt
6
7 def get_images_from_video(path_to_video, interval_sec):
8     video_capture = cv2.VideoCapture(path_to_video)
9     frame_rate = video_capture.get(cv2.CAP_PROP_FPS)
10
11     images = []
12     current_frame = 0
13     while True:
14         success, frame = video_capture.read()
15         if not success:
16             break
17         if current_frame % int(frame_rate * interval_sec) == 0:
18             rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB) # Convert to
19             pil_image = Image.fromarray(rgb_frame)
20             images.append(pil_image)
21             current_frame += 1
22
23     video_capture.release()
24     return images
25
26 def get_images_from_folder(path_to_images):
27     images = []
28     for filename in os.listdir(path_to_images):
29         if filename.endswith(".jpg") or filename.endswith(".png"):
30             image_path = os.path.join(path_to_images, filename)
31             img = Image.open(image_path)
32             images.append(img)
33     return images
```