|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Студент | Михеев Никита Алексеевич | | | | /…………………/ | | |
|  |  | | | |  | | |
| Факультет | ИВТ | | Группа | | | ИП-713 |  |
| Руководитель | Ракитский Антон Андреевич | | | / …………..……/ | | | |
|  |  |  | |  | | | |
|  | |  | |  | | | |
|  |  | | |  | | | |

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА**

«Разработка и исследование методов на основе машинного обучения для распознавания дорожных знаков ограничения скорости»

Пояснительная записка

Кафедра ПМиК

Допустить к защите

зав. кафедрой: проф., д.т.н.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Фионов А.Н.

**Министерство цифрового развития, связи и массовых коммуникаций**

**Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Сибирский государственный университет телекоммуникаций и информатики»**

**(СибГУТИ)**

Новосибирск 2021 г.

**Министерство цифрового развития, связи и массовых коммуникаций**

**Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Сибирский государственный университет телекоммуникаций и информатики»**

**(СибГУТИ)**

Форма утверждена

научно-методическим советом

ФГОБУ ВПО «СибГУТИ»

протокол № 2 от 04.03. 2014 г.

**КАФЕДРА Прикладной математики и кибернетики**

**ЗАДАНИЕ**

**НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА**

|  |  |
| --- | --- |
| СТУДЕНТУ Михееву Н.А. | ГРУППЫ ИП-713 |

УТВЕРЖДАЮ

**«\_\_\_\_\_» \_\_\_**\_\_\_\_\_\_\_\_\_\_\_\_\_\_

зав. кафедрой ПМиК

**\_\_\_\_\_\_\_\_\_\_\_/** \_\_\_\_\_\_\_\_\_\_\_\_\_\_/

Новосибирск 2021 г

**1. Тема выпускной квалификационной работы бакалавра**

Разработка и исследование методов на основе машинного обучения для

распознавания дорожных знаков ограничения скорости

утверждена приказом СибГУТИ от «19» января 2021 г. № **4/37о-21**

**2. Срок сдачи студентом законченной работы «**21» июня 2021 г.

**3. Исходные данные к работе**

1 Mattmann A. Chris. Machine Learning with TensorFlow. Simon and Schuster, 2021. 456 p.

2 Статья «Сверточная нейронная сеть, часть 1: структура, топология, функции активации и обучающее множество» [Электронный ресурс]. Хабр. URL: [https://habr.com/ru/post/348000/](https://habr.com/ru/post/348000/%20)

|  |  |
| --- | --- |
| **4.Содержание пояснительной записки**  **(перечень подлежащих разработке вопросов)** | **Сроки выполнения по разделам** |
| Работа с библиотечными фондами, сбор и анализ материалов по тебе выпускной квалификационной работы | 01.02.21 – 27.02.21 |
| Введение | 01.03.21 – 17.03.21 |
| Анализ аналогичных аппаратных средств и ПО | 19.03.21 – 03.04.21 |
| Формирование требований к продукту | 05.04.21 – 17.04.21 |
| Выбор программных средств | 19.04.21 – 24.04.21 |
| Поиск подходящего набора данных | 26.04.21 – 05.05.21 |
| Разработка системы | 06.05.21 – 15.05.21 |
| Подбор оптимальных параметров в свёрточных нейронных сетях | 17.05.21 – 25.05.21 |
| Тестирование готовой системы | 26.05.21 – 05.06.21 |

Дата выдачи задания «19» января 2021 г.

Руководитель Ракитский Антон Андреевич - доцент кафедры ПМиК,

кандидат технических наук \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*подпись*

Задание принял к исполнению «1» февраля 2021 г.

Студент - Михеев Никита Алексеевич \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Сибирский государственный университет телекоммуникаций и информатики»**

**(СибГУТИ)**

Форма утверждена научно-методическим

советом университета

протокол № 3 от 19.02.2015 г.

**ОТЗЫВ**

на выпускную квалификационную работу студента Михеева Н.А. по теме «Разработка и исследование методов на основе машинного обучения для распознавания дорожных знаков ограничения скорости»

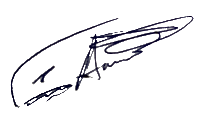
В рамках выполнения выпускной квалификационной работы Михеевым Н.А. были исследованы методы распознавания и классификации изображений и на основе проведённого исследования была разработана архитектура свёрточной нейронной сети для распознавания дорожных знаков. Разработанная сеть была качественно исследована и использована в представленном автором приложении, которое позволяет выделять и распознавать дорожные знаки на изображениях или видео.

Выпускная квалификационная работа выполнена Михеевым Н.А. на высоком уровне, в полном соответствии с поставленным заданием, все сроки были соблюдены. Доля оригинальности текста ВКР по результатам проверки на сайте antiplagiat.ru составляет 92,5%. Считаю, что представленная ВКР заслуживает оценки “отлично”, а Михеев Н.А. присуждения квалификации бакалавр.

Оценка уровней сформированности общекультурных и профессиональных компетенций обучающегося:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Компетенции | | Уровень сформированности  компетенции | | |
| Высокий | Средний | Низкий |
| Общекультурные | ОК-7 | + |  |  |
| ОК-8 | + |  |  |
| ОК-9 |  | + |  |
| Профессиональные | ОПК-4 | + |  |  |
| ОПК-5 | + |  |  |
| ПК-1 | + |  |  |
| ПК-2 | + |  |  |
| ПК-3 | + |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Работа имеет практическую ценность | + | Тема предложена предприятием |  |
| Работа внедрена |  | Тема предложена студентом | + |
| Рекомендую работу к внедрению | + | Тема является фундаментальной | + |
| Рекомендую работу к опубликованию | + | Рекомендую студента в магистратуру | + |
| Работа выполнена с применением ЭВМ | + | Рекомендую студента в аспирантуру |  |



Руководитель бакалаврской работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ к.т.н., доцент Ракитский Антон Андреевич 11.06.2021

**АННОТАЦИЯ**

Выпускной квалификационной работы студента Михеева Н.А.

(Фамилия,И.О.)

по теме «*Разработка и исследование методов на основе машинного обучения для распознавания дорожных знаков ограничения скорости*»

Объём работы - 36 страниц, на которых размещены 25рисунков и 9 листингов кода. При написании работы использовалось 10источников.

Ключевые слова: Python, свёрточные нейронные сети, YOLO, TensorFlow, Keras, PyQt, компьютерное зрение.

Работа выполнена Кафедра ПМиК СибГУТИ

(название предприятия, подразделения)

Руководитель доцент каф. ПМиК, кандидат технических наук, Ракитский А.А.

(должность, уч.степень, звание, Фамилия Имя Отчество*)*

Основные результаты

Были исследованы и разработаны свёрточные нейронные сети, с их помощью было разработано приложение, определяющее дорожные знаки ограничения скорости.

Содержание

[ВВЕДЕНИЕ 3](#_Toc74508811)

[1 ПОСТАНОВКА ЗАДАЧИ 5](#_Toc74508812)

[1.1 Анализ аналогичных аппаратных средств и ПО 5](#_Toc74508813)

[1.2 Основные требования к программному продукту 5](#_Toc74508814)

[2 Выбор программных средств 7](#_Toc74508815)

[2.1 Общие положения 7](#_Toc74508816)

[2.2 Выбор языка программирования 7](#_Toc74508817)

[2.3 Выбор среды разработки 8](#_Toc74508818)

[2.4 Выбор библиотек 9](#_Toc74508819)

[2.5 Выбор подхода к решению задач обнаружения и классификации 10](#_Toc74508820)

[3 разработка системы 13](#_Toc74508821)

[3.1 Поиск подходящего набора данных 13](#_Toc74508822)

[3.2 Программная реализация модулей системы 15](#_Toc74508823)

[3.2.1 Файловая структура 15](#_Toc74508824)

[3.2.2 Создание моделей 16](#_Toc74508825)

[3.2.3 Создание графической части приложения 21](#_Toc74508826)

[4 Тестирование системы 26](#_Toc74508827)

[4.1 Проверка работоспособности программы 26](#_Toc74508828)

[зАКЛЮЧЕНИЕ 35](#_Toc74508829)

[Список источников 36](#_Toc74508830)

[приложение А 37](#_Toc74508831)

[Приложение Б 38](#_Toc74508832)

# ВВЕДЕНИЕ

В современном автомобилестроении все чаще встречаются новые технические решения, направленные на обеспечение безопасности дорожного движения. Среди них особое распространение начали получать системы распознавания дорожных знаков: они облегчают задачу водителю, позволяют ему лучше сосредоточиться на самом процессе вождения, ведь потеря концентрации и внимания может привести к опасным для здоровья и жизни последствиям.

Следует отметить, что задачи по распознаванию объектов являются достаточно сложными ввиду постоянного изменения условий (освещение, видимость, местоположение и различность окружения), усложняют задачу распознавания. Области машинного обучения и компьютерного зрения на сегодняшний день являются одними из самых развивающихся и актуальных, однако до сих пор не существует какого-то единого и оптимального подхода для решения поставленной задачи. Существует большое количество нерешенных трудностей, которые сильно снижают эффективность обнаружения:

* Низкое входное разрешение изображения;
* Сложный и неординарный фон изображений;
* Различные сложные ракурсы создающие визуальные искажения на изображениях;
* Частичное перекрытие знака на изображении листвой деревьев, электрическими проводами и иными другими объектами.

В настоящее время для распознавания дорожных знаков ограничения скорости в современных автомобилях используются закрытые, коммерческие системы, которые поставляются вместе с автомобилем как дорогостоящая опция без возможности модификации. К таким системам относятся [1]: «Speed Limit Information Function» от KIA, «Road Sign Assist» от Toyota, «Traffic Sign Recognition» от Mazda. При анализе предметной области было выявлено, что даже указанные выше комплексы не в полной мере удовлетворяют решению поставленной задачи. Эффективность данных систем резко снижается в реальных условиях.

Исходя из вышесказанного можно сделать вывод, что тема дипломного проекта остается актуальна, особенно в развивающемся автомобилестроительном бизнесе.

Цель данной работы – разработать и исследовать надежный распознаватель знаков дорожного движения путем построения, обучения и настройки свёрточных нейронных сетей. Готовое приложение, в отличие от систем, встраиваемых в автомобили, имеет преимущества, такие как открытость, отсутствие привязки к определенному автомобильному бренду, стоимость. Для успешной реализации дипломного проекта необходимо выполнить определенный набор шагов:

1. Изучить теоретическую часть о настройке, обучении и создании свёрточных нейронных сетей.
2. Провести анализ аналогичных средств и программного обеспечения, существующих для распознавания знаков дорожного движения ограничения скорости.
3. Сформировать четкие требования к готовому программному продукту.
4. Рассмотреть языки программирования, на которых создаются нейронные сети, выбрать наиболее подходящий из них.
5. Провести анализ средств для создания и обучения свёрточных нейронных сетей.
6. Определить структуру приложения, подготовить набор входных данных, разработать пользовательский интерфейс.
7. Провести тестирование готового продукта на правильность и скорость работы.

Пояснительная записка состоит из четырех основных частей.

В первом разделе рассматривается вся предметная область, проводится исследование аналогичных популярных систем, ставятся цели и задачи для разработки.

Во втором разделе определяется конечный набор инструментов для разработки, включающий в себя язык программирования, среду разработки, вспомогательные библиотеки для обучения свёрточных нейронных сетей и создания пользовательского интерфейса, подход к решению задачи обнаружения и классификации.

В третьем разделе описаны этапы проектирования, разработки моделей свёрточных нейронных сетей, пользовательского интерфейса и создание готового программного продукта.

В четвертом разделе описано тестирование программы на реальных данных и демонстрация работы.

# ПОСТАНОВКА ЗАДАЧИ

## Анализ аналогичных аппаратных средств и ПО

Стоит отметить, что целью работы является не копирование существующих решений в области обнаружения и классификации знаков дорожного движения, а создание своей с помощью сверточных нейросетей и компьютерного зрения.

На сегодняшний день большинство автопроизводителей имеют свои подобные системы. Но они не всегда способны быстро и точно определить дорожный знак. Некоторые из таких систем имеют дело только с ограниченным количеством знаков. Например, «Speed Limit Information Function» от Kia и «Road Sign Assist» от Toyota способны определять только знаки ограничения скорости и знак запрета обгона. Также точность перечисленных средств резко снижается при наклоне, частичном перекрытии, загрязненности или повороте знака. Существует проблема распознавания знаков в слепых зонах автомобиля, особенно когда данные знаки установлены с нарушениями дорожных норм.

На рынке также представлено множество различных «радар-детекторов», которые занимаются не только обнаружением дорожных скоростных радаров, но и имеют дополнительный функционал, предназначенный для определения знаков ограничения скорости. Однако, так как данный функционал не является основным, точность и скорость работы подобных устройств оставляет желать лучшего. По этой причине использовать такие устройства для обеспечения безопасности дорожного движения нежелательно.

Отличительная особенность разрабатываемого проекта – возможность расширения списка знаков для обнаружения и точность работы. А поскольку решение не привязано к какой-либо коммерческой системе, то при наличии необходимого оборудования есть возможность легко переносить и масштабировать программу.

## Основные требования к программному продукту

Согласно выбранной теме выпускного квалификационного проекта необходимо сделать программное обеспечение для распознавания знаков ограничения скорости с помощью свёрточных нейронных сетей. Разделить данную задачу можно на два простых этапа:

* Обнаружение области расположения самого дорожного знака на изображении;
* Классификация дорожного знака по имеющимся в базе классам.

Найти и распознать знак можно будет на заранее сделанном фото, видео или в реальном времени с помощью веб-камеры. Взаимодействовать с программой можно будет с помощью оконного интерфейса.

В разрабатываемой программе должны быть реализованы следующие основные функции:

* понятный пользовательский графический интерфейс;
* переключение режима работы программы: с заранее созданными файлами или с веб-камерой в режиме «реального времени»;
* загрузка графического изображения, видеофайла и их обработка;
* получение изображения с веб-камеры и его обработка;
* демонстрация обработанных входных данных прямо внутри приложения;
* возможность оценить скорость работы программы по выводимым метрикам.

Для работы с программой необходимо функционирование графического интерфейса на любой современной операционной системе (например, Windows 7/10, Linux). Предпочтительной является реализация, основанная на мультиплатформенной библиотеке для разработки интерфейсов, по причине возможности охвата максимального количества устройств. Также работа нейронных сетей предполагает большое задействование вычислительных мощностей, потребуется вычислительная машина с графическим ускорителем с большим объемом видео памяти.

# Выбор программных средств

## Общие положения

Перед началом реализации проекта необходимо определиться с языками программирования и средами разработки программного продукта, как как этот выбор может повлиять на выбор сторонних библиотек, средств разработки пользовательских интерфейсов и т.д.

Выбор программных средств должен основываться на принципах стабильной работы используемого стороннего программного обеспечения, современных их реализациях и использовании актуальных методов разработки программного обеспечения.

## Выбор языка программирования

Выбор языка программирование определяет весь инструментарий, который будет доступен во время реализации проекта. Он включает в себя среды разработки, библиотеки, фреймворки и другие вещи, характерные для того или иного языка. Для совершения правильного выбора можно выделить следующие критерии:

1. Предыдущий опыт
2. Аппаратная платформа программы
3. Распространенность
4. Производительность
5. Актуальность

В настоящее время Python [2] – один из самых распространенных и часто используемых языков программирования ввиду своей простоты, удобочитаемости, обширной поддержки сторонних модулей с большим сообществом разработчиков. Также из достоинств можно выделить легкую интеграцию с веб-сервисами, динамическую типизацию и интуитивно понятные для пользователя структуры данных.

Язык C является, вероятно, самым старейшим из часто используемых языков программирования на данный момент. C++ является расширенной его версией и их можно отнести в одну категорию. Они достаточно трудны для изучения новичкам в разработке ввиду своей сложности и избыточности. Тем не менее, оба языка широко распространены в компьютерной науке и программировании. С целью большего разнообразия для них существует большое количество компиляторов, делающих приложения легко переносимыми и хорошо портируемыми. Оба языка считаются одними из самых высокопроизводительных.

Язык программирования Java следует парадигме объектно-ориентированного программирования и подходу «Написал один раз – используешь везде». Java используется в настольных, сетевых, мобильных и корпоративных приложениях. Язык представляет из себя целую экосистему инструментария, который может понадобиться в разработке, включающую в себя:

Java Developer Kit – комплект для написания и компиляции кода на языке Java;

Java Runtime Environment – исполняющая среда программ на языке программирования Java.

Отталкиваясь от данных критериев и свойств языков программирования, выбор был сделан в пользу на Python. Он проще для изучения, чем C и С++, а написанный код выглядит более лаконично, чем объектно-ориентированный как на Java.

Также наличие большого количества сторонних библиотек (пакетов) для работы с нейронными сетями, машинным обучением и графической информацией, которые впоследствии понадобятся при разработке и тестировании системы распознавания знаков дорожного движения, стало решающим фактором при выборе языка.

## Выбор среды разработки

Выбор среды разработки чаще всего основывается на спецификации разрабатываемого проекта. На выбор могут повлиять такие факторы, как удобство использования среды разработки, интерфейсы, дополнительные возможности при разработке, операционная система или оборудование, для которого пишется та или иная программа, и так далее.

Для разработки приложений на языке Python [3] чаще всего используется среда разработки JetBrains PyCharm. PyCharm – одна из лучших профессиональных, полнофункциональных IDE. Существует как бесплатный (open-source) версия Community, так и платный Professional вариант среды разработки. PyCharm доступен на операционных системах семейств Windows, Linux, Mac OS. Есть поддержка веб-фреймворков, таких как Django, Flask, Google App Engine и других.

Кроме PyCharm, существуют и другие среды разработки, используемые при разработке программ на языке Python.

Spyder – специальная среда разработки для научных расчетов, которая ввиду своей узкой специализации и ограниченного функционала для разработки нейронных сетей и компьютерного зрения не подходит для проекта.

Visual Studio + Python – знакомая и понятная среда разработки Visual Studio от Microsoft с плагином для разработки на языке Python. Имеет огромнейший функционал, но слабо подходит для машинного обучения и компьютерного зрения ввиду того, что данная среда изначально разрабатывалась под написание программ на языках C и C++.

Jupyter Notebook – действительно мощный инструментарий для интерактивной разработки с большим функционалом для работы в области наук о данных и машинном обучении. Файлы исходного кода на языке Python для данного «блокнота» обладают особым расширением «.ipynb». В нем все содержимое хранится в формате JSON, для которого каждая ячейка исходного кода, вложенные изображения входы и выходы вычислений, математика и пояснительный текст преобразуются в строки текста с некоторыми метаданными. Это необходимо для облегчения работы с системами контроля версий.

Все перечисленные среды разработки обладают меньшим функционалом, чем PyCharm, из-за чего скорость и качество работы может существенно снижаться.

Конечный выбор IDE остановился на PyCharm благодаря повсеместному использованию, знакомому и удобному интерфейсу, обширным возможностям по работе с нейронными сетями и машинному обучению, поддержке Jupyter формата «.ipynb», удобному рефакторингу кода с помощью встроенной инспекции кода (Code Inspection) для более эффективной разработки и хорошей поддержки системы контроля версий Git.

## Выбор библиотек

Выбор библиотек, а именно включения в программу разработанного вне основного проекта программного кода, также очень важен.

Для разработки нейросети-классификатора была выбрала библиотека TensorFlow с реализацией Keras API [4]. TensorFlow – открытая программная библиотека для машинного обучения, разрабатывается и поддерживается компанией Google для решения задач построения и тренировки нейронных сетей. Применяется как для исследований, так и для разработки собственных продуктов от Google и сообществом. Основная реализация API предназначена для работы на языке Python, но также существуют и реализации для работы на C++, Java, Go, Swift и Haskell. Исходя из перечисленных качеств, альтернатив данной библиотеке просто нет.

Для быстрых векторных вычислений и более быстрых типов данных, которые будут использоваться для обучения модели, выбрана была библиотека NumPy [5]. Она обладает открытым исходным кодом, поддержкой больших многомерных массивов данных и матриц, вместе с большой библиотекой математических функций и операций для работы с этими массивами.

Для обнаружения знаков дорожного движения была выбрана библиотека OpenCV [6]. Реализована на C++ с оберткой для языка программирования Python, также есть обертки для Java, Ruby, Lua и других языков. Насчитывает в себе более 2500 оптимизированных алгоритмов для разработки программ на основе компьютерного зрения и машинного обучения. С этой библиотекой работают почти все крупные компании, такие как Google, Microsoft, Intel, IBM, Sony.

Для создания пользовательского интерфейса существует встроенная в Python библиотека Tkinter, сторонние библиотеки PyQt5, PyGTK, PyForms и фреймворк Kivy [7]. Для создания мультиплатформенного приложения из данного набора библиотек и фреймворков подходят только Tkinter, PyGTK и Kivy. Инструментарий для создания пользовательского интерфейса должен обладать знакомым и понятным набором возможностей, быть прост в разработке.

В конечном итоге был выбран набор библиотек для создания графических интерфейсов на базе знакомой платформы Qt5 от компании Digia под названием PyQt5. PyQt5 реализован в виде набора Python-модулей. Эта библиотека имеет более 600 классов и 6000 функций и методов. Также отличительной особенностью можно назвать то, что библиотека является мультиплатформенной и работает на всех основных операционных системах, в том числе Linux, Windows и Mac OS.

## Выбор подхода к решению задач обнаружения и классификации

Процесс распознавания и классификации дорожных знаков подразумевает под собой два важных этапа. Первый из них – обнаружение дорожного знака на имеющемся изображении или стоп-кадре видеосигнала. На этом этапе используются методы, основывающиеся на особенностях цвета, формы и иных признаков дорожных знаков. Данные обнаружители, или детекторы, определяют и сегментируют области входного кадра, цвета которых входят в заранее заданные диапазоны. Для решения подобных задач существует большое количество различных архитектур [8]:

* R-CNN (Region-based Convolutional Neural Networks. От англ.: свёрточные нейронные сети на основе регионов). Одна из первых таких моделей. Работает как обычный классификатор изображений, которому на вход подаются разные его регионы. Для этих сегментов делается предсказание. Так как в данном случае необходимо каждое изображение проходить по несколько тысяч раз, реализация является достаточно медленной.
* Fast R-CNN (Fast Region-based Convolutional Neural Networks. От англ.: быстрые свёрточные нейронные сети на основе регионов). Улучшенная версия предыдущего алгоритма, работающая по очень похожему принципу, но главное отличие заключается в том, что все изображение целиком подается на вход сверточной нейронной сети и только потом уже генерируются специальные регионы. Однако по-прежнему данная реализация остается достаточно медленной для решения задач в реальном времени.
* YOLO (You Only Look Once. От англ.: смотрите только один раз). Данный метод работает совсем по другому принципу так как в нем не используются регионы вообще. Один из самых быстрых из ныне существующих. В YOLO свёрточная нейронная сеть применяется один раз и сразу все изображение делит на своеобразную сетку, на каждой из полученных ячеек предсказываются так называемые bounding boxed (от англ. Ограничивающие рамки) и просчитываются вероятности того, что там есть искомый объект для каждого участка изображения.

Для разработки приложения был выбран подход YOLO из-за своей скорости и точности работы. Данный метод быстрее R-CNN в 1000 раз и в 100 раз быстрее Fast R-CNN.

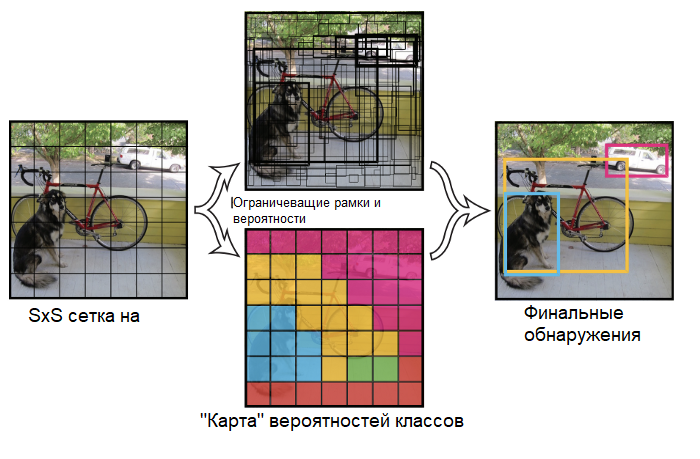
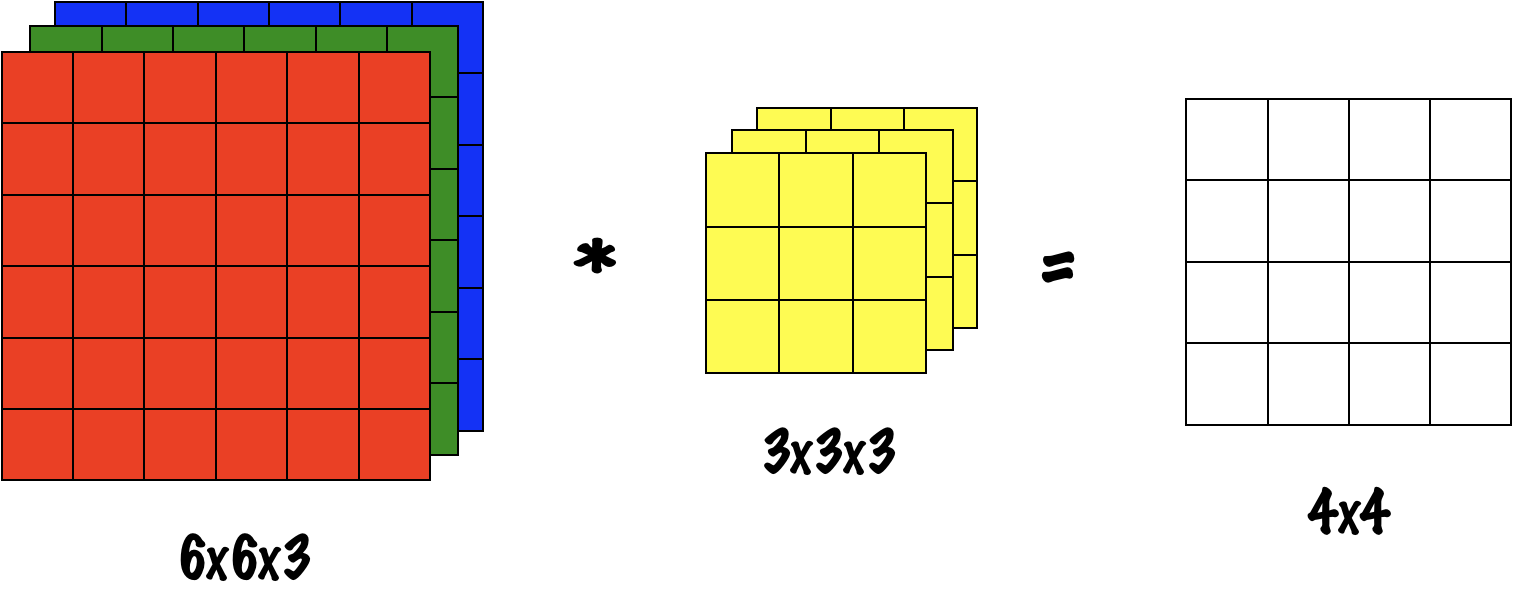


Рисунок 2.1 - Процесс работы модели YOLO.

Вторым важным этапом является классификация найденного знака. Наибольшее распространение тут получили классификаторы, основывающиеся на свёрточных нейронных сетях. В основе их принципа лежит процесс «свертки» - простой операции, в ходе которой матрица небольшого размера (также ее называют фильтр) перемещается по входной матрице, и результат их умножения записывается в соответствующую ячейку другой выходной матрицы.



На рисунке входная матрица размером 6х6, фильтр размера 3х3 и результирующая матрица.

Рисунок 2.2 – пример работы слоя сверточной нейронной сети.

Данная операция производится на всех свёрточных слоях. В свою очередь каждый из фильтров состоит из такого количества слоев, сколько подается каналов на вход. Обычно это три цветовых канала – Красный, Синий и Зеленый (модель RGB – Red Green Blue). Благодаря этой особенности фильтр в процессе свёртки может реагировать на все подаваемые каналы и отдавать одну результирующую матрицу.

Типичная свёрточная нейронная сеть состоит из совокупности таких свёрточных слоев и слоев субдискретизации (иначе называют подвыборкой), которые чередуются между собой.

Плюсом данной архитектуры перед нейронными сетями с полной связью можно назвать значительное сокращение вычислений, необходимых во время работы. В связи с этим снижается объем требуемой памяти вычислительной машины. Также данная модель обладает готовым фреймворком под названием DarkNet, с помощью которого можно легко провести как обучение нейросети на собственных данных, так и ее настройку.

# разработка системы

## Поиск подходящего набора данных

Свёрточным нейронным сетям необходимо большое количество входящих данных, состоящих из обучающих и валидационных изображений, для формирования оптимально настроенной модели, которая будет показывать себя достойно на практике с точки зрения точности и производительности, измеренных на обучающей и тестовой выборках. Так как программа будет состоять из двух частей, необходимо подготовить два разных набора данных для нейросети-детектора и нейросети-классификатора.

Важно, чтобы нейросети не только получили как можно больше изображений, но и были как можно более разнообразны для оптимального обучения. Например, если нейросети получат на вход большое количество изображений с определенным дорожным знаком, но меньше с другими, то обучение нейросети-детектора для конкретного знака будет легкой задачей, но тогда данная нейросеть будет хорошо отрабатывать только с одним знаком дорожного движения. И такой плохой детектор не сможет быть частью системы автономного вождения, призванной обеспечивать безопасность дорожного движения.

Обучение свёрточных нейронных сетей предполагает создание определённой архитектуры, которая включает в себя количество слоев, их типы, порядок и так далее, а потом позволяет оптимизатору обработать тысячи входящих изображений в обучающем наборе, чтобы оценить оптимальные значения для параметров в этих слоях с помощью метода обратного распространения ошибки. Данный результат сильно зависит от качества входящего набора данных. Важно отметить, что количество данных может измеряться от нескольких тысяч изображений до нескольких миллионов, в теории даже до бесконечности. Все зависит от множества факторов, таких как сложность обнаружения конкретного знака, доступные вычислительные мощности, количество имеющихся данных для обучения.

Одним из таких готовых наборов данных можно отметить Российскую базу автодорожных знаков от Лаборатории компьютерной графики и мультимедиа [9]. В ней размечено более 150000 кадров и дано 156 классов дорожных знаков. База распространяется свободно, и доступна для всех желающих. Она включает в себя два поднабора: rtsd-d1 – набор с изображениями, полученными с камер и авторегистраторов и rtsd-r1 – набор с близкими изображениями знаков (рисунки 3.1 и 3.2).



Рисунок 3.1 – примеры изображений в наборе rtsd-r1.

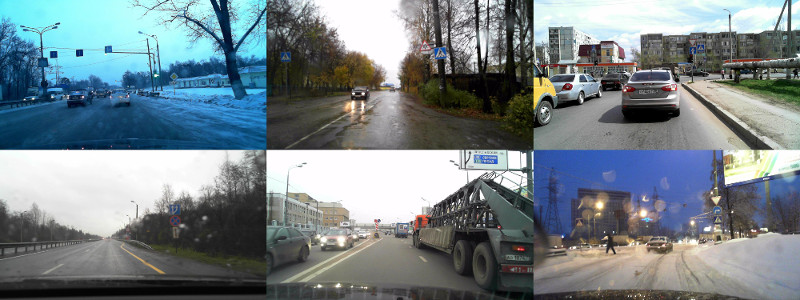


Рисунок 3.2 – примеры изображений в наборе rtsd-d1.

Первый поднабор из имеющихся изображений и аннотаций к ним с помощью небольшого скрипта на языке python можно легко преобразовать в YOLO формат для обучения нейросети-детектора [10, раздел «How to train (to detect your custom objects»]. Так как на данном этапе нет необходимости в обучении на всех классах дорожных знаков, было решено разделить все знаки на 4 основные категории: знаки опасности, предупреждающие, разрешающие и прочие.

Для того, чтобы выполнить преобразования, сначала считывались все аннотации к изображениям (листинг 3.1).

Листинг 3.1 – загрузка данных из файлов и подсчет значений, необходимых для формата YOLO:

|  |
| --- |
| # Загрузка данных из скачанного датасета  ann1 = pd.read\_csv(full\_path\_to\_ts\_dataset + '\\' + 'rtsd-d1-gt\\danger\\test\_gt.csv')  ann2 = pd.read\_csv(full\_path\_to\_ts\_dataset + '\\' + 'rtsd-d1-gt\\mandatory\\test\_gt.csv')  ann3 = pd.read\_csv(full\_path\_to\_ts\_dataset + '\\' + 'rtsd-d1-gt\\prohibitory\\test\_gt.csv')  ann = pd.concat([ann1, ann2, ann3], join='inner')  # Подсчитаем центр места нахождения каждого знака по X и Y.  ann['center x'] = (ann['x\_from'] + (ann['x\_from'] + ann['width'])) / 2  ann['center y'] = (ann['y\_from'] + (ann['y\_from'] + ann['height'])) / 2 |

Далее вычислялась ширина и длина знака и сохранялось изображение и полученная аннотация в YOLO формате с одинаковыми именами в отдельную папку (листинг 3.2).

Лиситнг 3.2 – вычисление длины и ширины знаков, сохранение полученных данных:

|  |
| --- |
| sub\_r['width'] = sub\_r['width'] / w  sub\_r['height'] = sub\_r['height'] / h  resulted\_frame = sub\_r.loc[:, ['class id', 'center x', 'center y', 'width', 'height']].copy()  path\_to\_save = full\_path\_to\_ts\_dataset + '\\yolo\_formated\\' + image\_name+ '.txt'  resulted\_frame.to\_csv(path\_to\_save, header=False, index=False, sep=' ')  path\_to\_save = full\_path\_to\_ts\_dataset + '\\yolo\_formated\\' + image\_name + '.jpg'  cv2.imwrite(path\_to\_save, image) |

Второй поднабор сразу готов к использованию, дополнительных работ с ним не требуется.

## Программная реализация модулей системы

### Файловая структура

Исходя из задания, зададим необходимую файловую структуру для проекта:

1. В корневом каталоге:

* Main.py – основной модуль для запуска приложения;
* Processing.py – модуль с функциями обработки изображения, видео и потока с веб-камеры;
* Design.py – модуль с описанием графического интерфейса, основанного на библиотеке PyQt5.

1. Каталог \additional\_tools содержит в себе вспомогательные скрипты на языке программирования Python для превращения датасета в формат YOLO и для разбиения полученных данных на обучающую и тестовую выборки
2. Каталог \data для хранения датасета, из которого программа во время обучения и тестирования будет получать данные
3. Каталог \model для хранения полученных в ходе обучения моделей для дальнейшего их использования в основной программе
4. Каталог \notebooks для хранения файлов исходных кодов обучения и тестирования моделей в формате «.ipynb».

### Создание моделей

Для корректной работы готового приложения необходимо обучить две модели нейросетей: модель-детектор и модель-классификатор.

Для модели-детектора был выбран формат YOLO. Для того, чтобы обучить данную модель на подготовленных данных, был задействован фреймворк под названием DarkNet – «теневая сеть», где заранее определены 106 сверточных слоев. После скачивания данного фреймворка был создан конфигурационный файл «yolov3\_train.cfg» в котором определялось количество итераций, задавалось количество классов данных, указывались пути нахождения набора данных. Далее уже был запущен сам процесс обучения (рисунок 3.3).

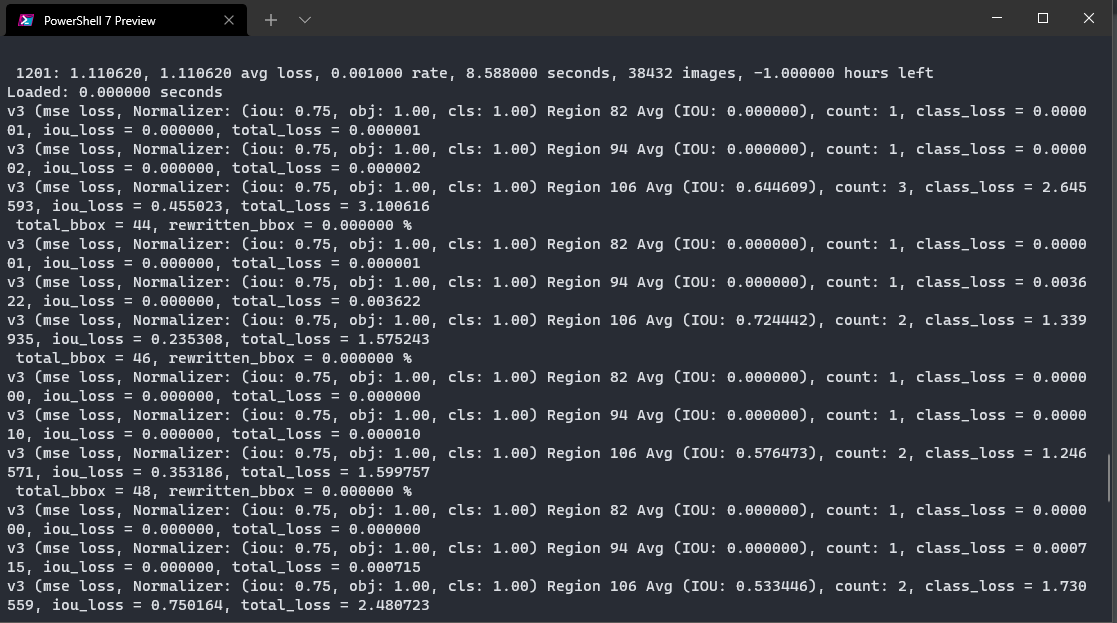


Рисунок 3.3 – процесс обучения модели YOLO, где «1201» в левом верхнем углу – номер эпохи.

Для того, чтобы полностью обучить нейронную сеть на всех входных данных, потребовалось достаточно длительное время ввиду большого количества итераций, порядка 8 тысяч. Каждая итерация занимает порядка 5 секунд, на все время обучения модели ушло больше 10 часов. Их количество было отражено в конфигурационном файле и задавалось по формуле:

Формула 3.1 – вычисление количества итераций для обучения:

*Max\_batches = classes \* 2000,*

где max\_batches – количество итераций, а classes – количество классов данных.

После того, как все итерации были пройдены, на выходе был получен файл в формате «.weights». В данном файле находились все веса для обученной модели.

Дальнейший процесс заключался в обучении и создании модели-классификатора. Для этого был разработан специальный модуль на языке Python с использованием библиотек NumPy и TensorFlow совместно с Keras API. Для начала был загружен набор изображений rtsd-r1 вместе с обозначениями всех классов дорожных знаков. Затем загруженный набор необходимо было разделить на обучающую и тестовую выборки. Сделано это было с помощью самописного скрипта на языке Python. В нем считываются пути всех изображений, затем 15% из них отправляются в тестовую выборку, а остальные в обучающую. Информация о составах выборок сохраняется в текстовые файлы под названиями «train.txt» - для обучающей выборки и «test.txt» - для тестовой. Листинг данного скрипта можно найти в приложении Б под номером 5.

Дальнейшая работа заключалась в создании модели и нахождении оптимальной конфигурации ее параметров (листинг 3.3). Данный процесс также занял длительное время, так как с изменением даже одного из множества параметров для проверки требовалось переобучить всю модель целиком. Методом проб и ошибок была подобрана такая модель. В качестве оптимизатора всей модели был выбран Adam (Adaptive Moment Estimation. От англ.: Адаптивная оценка момента) как наиболее удачный из имеющихся и протестированных. Функцией потерь была выбрана разреженная категориальная перекрестная энтропия, обеспечивающая наибольшую точность, когда дело касается взаимоисключающих классов знаков, как в данном случае.

Листинг 3.3 – оптимальная модель свёрточной нейронной сети классификатора

|  |
| --- |
| *# Определение модели и ее типа - Секвентальная см линейным стеком слоев.* model = Sequential([  # Шаг №1 - Слой свертки. Форма на вход: 32х32х3, где 3 - количетсво каналов *(RGB).*  Conv2D(32, kernel\_size=(3, 3), input\_shape=(32, 32, 3), activation='relu'),  *# Шаг №2 - Слой подвыборки с извлечением максимума. Размеры матрицы: 2х2.*  MaxPooling2D(2, 2),  *# Шаг №3 - Исключение. Регуляризируем дабы не было переобучения.*  Dropout(0.2),  *# Шаг №4 - Второй слой свертки.*  Conv2D(64, kernel\_size=(3, 3), activation='relu'),  *# Шаг №5 - Снова подвыборка после второго сверточного слоя и также с извлечением максимума.*  MaxPooling2D(2, 2),  *# Шаг №6 - Снова исключение.*  Dropout(0.2),  *# Шаг №7 - Выравниваем последний полносвязный слой, конвертируем в меньшую размерность*  Flatten(),  *# Шаг №8 - Получаем информацию со всех предыдущих слоев в слой с 256 нейронами.*  Dense(256, activation='relu'),  *# Шаг №9 - Исключение.*  Dropout(0.2),  *# Шаг №10 - Тот же слой, только нейронов меньше - 128.*  Dense(128, activation='relu'),  *# Шаг №11 - Исключение.*  Dropout(0.2),  *# Шаг №12 - Последний Dense слой с 64 нейронами и другим активатором.*  Dense(64, activation='softmax') ]) *# Собираем модель, оптимизатор Adam - как наиболее удачный из протестированных, # функция потерь - разреженная категориальная перекрестная энтропия,*  *# метрики только на точность модели на каждой из эпох* model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy']) |

После этапа сборки параметры полученной модели были выведены на экран методом model.summary():

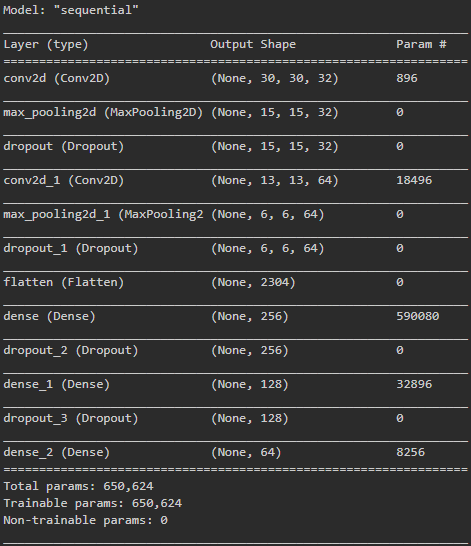


Рисунок 3.4 – отчет о готовой модели нейронной сети

Как видно из рисунка, всего модель имеет 650624 параметра. Тип каждого слоя и форма выходной матрицы видна на рисунке.

Также были заданы дополнительные метрики и проверки (листинг 3.4) для того, чтобы обучение модели проходило как можно оптимальнее, и точка сохранения модели только с лучшими результатами показателей функции потерь.

Листинг 3.4 – пример одной из функций для ранней остановки

|  |
| --- |
| # Функция ранней остановки, если модель достаточно обучена и дальнейшее обучение не требуется early\_stopping = EarlyStopping(monitor='val\_loss', restore\_best\_weights=True, patience=5, mode='min') |

Затем начался процесс обучения модели (рисунок 3.5), было задано 20 эпох, но благодаря функциям остановки, которые отслеживали показатели потерь, эпох оказалось достаточно всего 12-16.

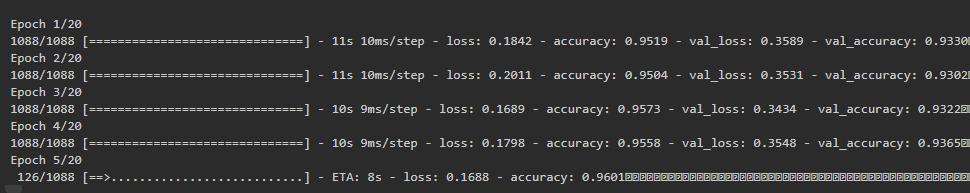


Рисунок 3.5 – процесс обучения нейронной сети

Готовая на выходе модель обладает точность в примерно 96.951% на обучающей выборке и в 92.96% на тестовой.

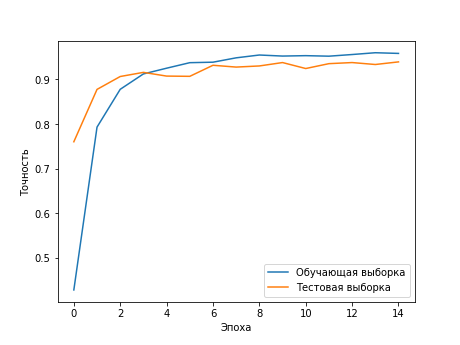


Рисунок 3.6 – график зависимостей точностей обучающей и тестовой выборок от количества эпох.

После того, как была получена готовая модель, было решено провести ее тестирование на 25 случайных изображениях из набора (листинг 3.5). Модель справилась хорошо, из 25 случайных рисунков всего один был распознан ошибочно, но даже при рассмотрении вживую данный пример оказался трудноразличимым.

Листинг 3.5 – функция проверки тестовых изображений и составления из них рисунка

|  |
| --- |
| # Задаем размер рисунка plt.figure(figsize=(15, 15)) for i in range(25):  # Случайный индекс изображения  start\_index = random.randint(0, 10000)  plt.subplot(5, 5, i + 1)  plt.xticks([])  plt.yticks([])  # Берем изображение на вход  x\_input = X\_test[start\_index:start\_index+1]  # Составляем предсказание  prediction = labels\_dict[np.argmax(model.predict(x\_input))]  # Получаем класс знака на самом деле  actual = labels\_dict[Y\_test[start\_index]]  col = 'g' if prediction == actual else 'r'  # Подписываем с определенным цветом текста в зависимости от правильности предсказания  plt.xlabel(f'На деле: {actual}\nМодель: {prediction}', color=col)  plt.imshow(X\_test[start\_index]) # Выводим рисунок и сохраняем plt.show() plt.savefig('../training\_examples.png') |



Рисунок 3.7 – демонстрация работы проверочной функции

### Создание графической части приложения

После того, как обе модели были обучены и протестированы, было реализовано графическое приложение с помощью библиотеки PyQt5. Сначала в программе, поставляемой вместе с библиотекой, QtCreator был создан дизайн графического приложения. С помощью специальных инструментов на макете, или виджете, были расположены графические элементы: текстовые лейблы, кнопки, элемент переключения вкладок в приложении. Данный макет был сохранен в виде модуля для языка Python под названием «design.py» для того, чтобы в дальнейшем данный дизайн можно было легко использовать.

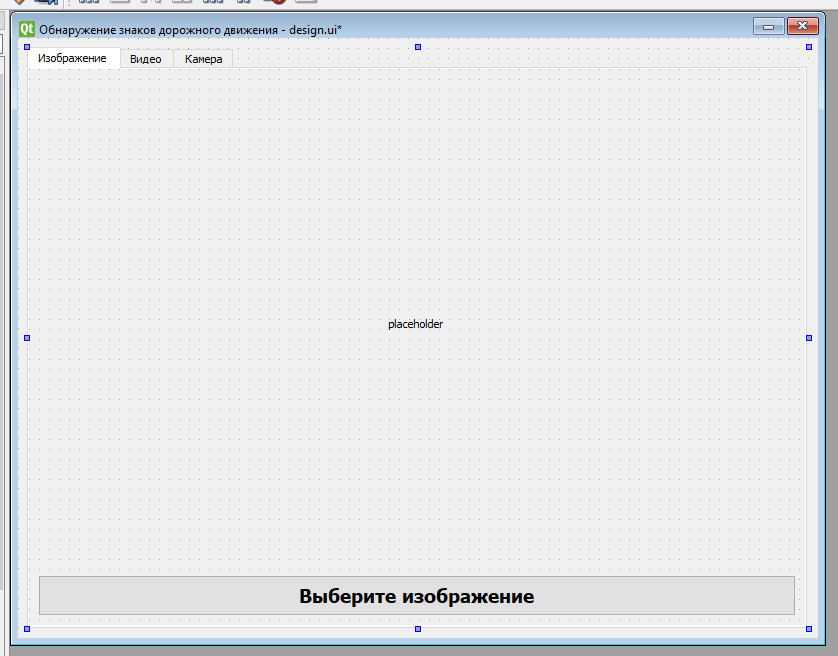


Рисунок 3.8 – макет графической части приложения в инструменте QtDesigner.

Уже в проекте данный модуль был подключен с использованием директивы «import». С помощью методов библиотеки был инициализирован основной класс приложения. В нем подключается главное окно и расставляются все элементы согласно макету (листинг 3.6).

Листинг 3.6 -

|  |
| --- |
| # Основной класс главного окна class MainApp(QtWidgets.QMainWindow, design.Ui\_MainWindow):   # Инициализирующая функция  def \_\_init\_\_(self):  # Ключевая точка для старта  super().\_\_init\_\_()  # Установка дизайна  self.setupUi(self)  # Установка начальной вкладки после запуска приложения  self.tabWidget.setCurrentIndex(0)  # Установка фукнции обработки изображения на сигнал нажатия кнопки  self.pushButton.clicked.connect(self.process\_image)  # Установка фукнции обработки видео на сигнал нажатия кнопки  self.pushButton\_2.clicked.connect(self.process\_video)  # Установка функции обработки видео с камеры на сигнал переключения соотвествующей вкладки  self.tabWidget.currentChanged.connect(self.selector)  # Установка начальных значений информационных полей в программе  self.label.setText('')  self.label\_2.setText('') |

После того, как инициализация графической части была готова, был реализован модуль processing в котором происходит обработка изображения, видео и потока с камеры устройства. Для того, чтобы обработка происходила быстрее, в модуле заранее идет загрузка модели-детектора и модели-классификатора с помощью специальных методов из библиотеки OpenCV и TensorFlow (листинг 3.7). Также подключаются опциональные настройки для использования графических ускорителей с поддержкой программно-аппаратной архитектуры параллельных вычислений от Nvidia.

Листинг 3.7 – загрузка моделей нейронных сетей в программе

|  |
| --- |
| # Считываем все названия классов labels = pd.read\_csv*(*'data/label\_names\_eng.csv'*)* # Загружаем готовую модель-детектор знаков с конфигурационным файлом network = cv2.dnn.readNetFromDarknet*(*'data/yolov3\_ts\_test.cfg', 'data/znaki.weights'*)* # Включаем дополнительные настройки графического ускорителя для увеличения скорости рабоыт программы network.setPreferableBackend*(*cv2.dnn.DNN\_BACKEND\_CUDA*)* network.setPreferableTarget*(*cv2.dnn.DNN\_TARGET\_CUDA\_FP16*)*  ...  # Загружаем модель-классификатор model = load\_model*(*'models/traffic\_model.h5'*)* |

Далее все функции из этого модуля начинают свою работу только после вызова в графическом окне программы. Разберем один из методов обработки поэтапно на примере обработки изображения.

Для начала изображение считывается по пути, который был передан из основой функции программы. Затем полученное изображение берется как специальный объект под названием «блоб» с помощью библиотеки OpenCV, в этом объекте хранится информация о всех группах пикселей с общими атрибутами, например, значение серого (листинг 3.8).

Далее этот объект передается на вход модели-детектору, которая, пропуская эти группы пикселей через свои слои, находит все вероятности нахождения знаков дорожного движения (листинг 3.8). Чтобы отсечь лишние обнаружения применялся алгоритм подавления немаксимумов, в котором с помощью заранее определенных констант были выбраны минимальная вероятность нахождения знака и минимальный порог фильтрации для слабых граничных рамок (листинг 3.8).

Листинг 3.8 – обработка входного изображения

|  |
| --- |
| def process\_image*(*path*)*:  *""" Метод, в котором обрабатывается изображение """* # Считываем изображение  image\_BGR = cv2.imread*(*path*)* # Получаем высоту и ширину изображения  h, w = image\_BGR.shape*[*:2*]* # Получаем объект с необработанными данными из изображения  blob = cv2.dnn.blobFromImage*(*image\_BGR, 1 / 255.0, *(*416, 416*)*, swapRB=True, crop=False*)*   # Проходимся по изображению через слои нашей модели   network.setInput*(*blob*)*  ...output\_from\_network = network.forward*(*layers\_names\_output*)*  *...* # Определяем границы найденых знаков и отсекаем их в отдельный массив,  # чтобы в дальнейшем использовать на модели-классификаторе  bounding\_boxes = *[]* confidences = *[]* class\_numbers = *[]*  ...  scores = detected\_objects*[*5:*]* class\_current = np.argmax*(*scores*)* confidence\_current = scores*[*class\_current*]* ...  # Применяем алгоритм подавления немаксимумов для отсечения ошибочно найденных объектов  # и уточнения границ рамок  results = cv2.dnn.NMSBoxes*(*bounding\_boxes, confidences, probability\_minimum, threshold*)* |

После применения алгоритма подавления немаксимумов и уточнения границ рамок знаков дорожного движения в дело вступает модель-классификатор, которая, смотря на части изображения по полученным координатам, определяет класс знака (листинг 3.9). В этом же участке кода происходит отрисовка цветного квадрата, ограничивающего область знака, и подписи с названием класса знака. Полученное готовое изображение с результатами работы обработчика отдельно сохраняется в файл и выводится пользователю в приложении (листинг 3.9).

Листинг 3.9 – добавление предсказаний на изображение и сохранение в файл.

|  |
| --- |
| # Смотрим, осталось ли что-то после подавления if len*(*results*)* > 0:  # Проходимся по все результатам  for i in results.flatten*()*:  ...  # Получаем фрагмент изображения на основе предсказания предыдущей модели  c\_ts = image\_BGR*[*y\_min: y\_min + int*(*box\_height*)*, x\_min: x\_min + int*(*box\_width*)*, :*]  ...* # Если все хорошо - получаем blob этого фрагмента  else:  blob\_ts = cv2.dnn.blobFromImage*(*c\_ts, 1 / 255.0, size=*(*32, 32*)*, swapRB=True, crop=False*)*  ...  # Получаем все предсказания классов  scores = model.predict*(*blob\_ts*)* # Выбираем максимальное из них  prediction = np.argmax*(*scores*)  ...* cv2.imwrite*(*'result.png', image\_BGR*)* |

Для обработки видео и сигнала с камеры основным отличием можно выделить только то, что все вышеописанные действия происходят в цикле до тех пор, пока не будет полностью обработаны все кадры входящего видеопотока. Еще одно отличие обработки видео с камеры устройства заключается в том, что вывод готового изображения с рамками знаков и именами классов можно увидеть только в отдельном окне, которое отрывается после выбора соответствующей вкладки в программе. И закрыть данное окно можно нажав на крестик, либо нажав клавишу «Й», или «Q», на клавиатуре.

Сразу после обработки данных одним из возможных способов в программе выводится информационное окно со временем работы алгоритма, а также с количеством обработанных кадров и скоростью, измеряемой в кадрах в секунду (FPS), в случаях видео и камеры.

С полным листингом программы с комментариями можно ознакомиться в приложении Б.

# Тестирование системы

## Проверка работоспособности программы

Перед тем, как начать использовать программу, необходимо провести комплексное тестирование всех модулей, которые принимают участие в работе данной системы, и проверить реакцию программы в различных случаях с разными реальными данными, подаваемыми на вход. В больших и сложных системах с нейронными сетями зачастую содержатся ошибки, не выявленные в процессе отладки. Некоторые из них не всегда находятся на виду и могут возникать во время работы программы. Так как готовое приложение призвано сохранять безопасность дорожного движения, тестирование как нельзя актуально.

Для проверки использовался компьютер с графическим ускорителем от компании Nvidia GTX1650 ti, так как в программе реализована опциональная, при наличии, поддержка обработки изображений с помощью программно-аппаратной архитектуры параллельных вычислений CUDA (аббр. от англ. Compute Unified Device Architecture – унифицированная архитектура вычислительных устройств). С использованием этого инструментария скорость работы приложения увеличивается в разы, по сравнению с обработкой на центральном процессоре.

Также программа протестирована на двух операционных системах – Windows 10 и GNU/Linux, а конкретно Ubuntu 20.04. Было установлено, что продукт без проблем запускается, полностью выполняется весь имеющийся функционал, интерфейс остается неизменным при использовании в той или иной системе, что обеспечивает удобство конечному пользователю и не затрачивает лишнее время на привыкание к интерфейсу.

Приложение представляет из себя окно, в котором находятся вкладки под различные виды входящих данных: изображение, видео и камера.

Разработанное приложение открывается запуском главного python файла в корневом каталоге «main.py». Перед пользователем возникает стандартная первая вкладка, где пользователю предлагается выбрать изображение для обработки системой (рисунок 4.1).

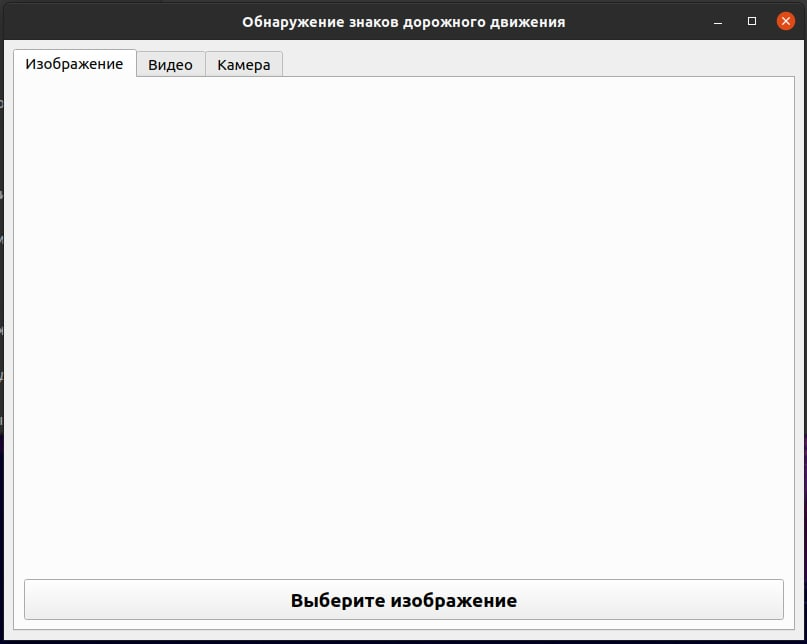


Рисунок 4.1 – главное окно программы

В зависимости от формата входных данных, можно выбрать соответствующую вкладку в приложении. Во вкладке изображение после нажатия кнопки «Выберите изображение» открывается файловый менеджер и предлагается выбор (рисунок 4.2).

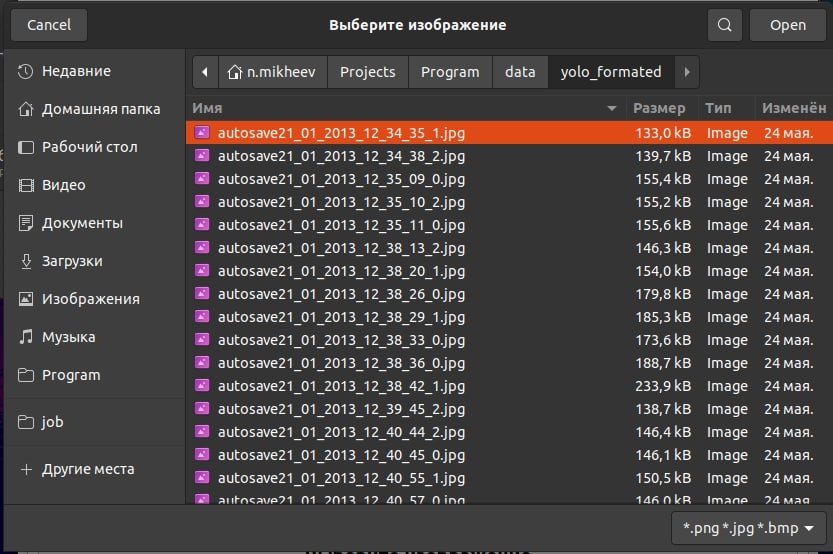


Рисунок 4.2 – выбор изображения.

Для удобства пользователя, в файловом менеджере ограничен выбор форматов файлов до «.png, .jpg, .bmp», которые может обработать программа. Если изображение было выбрано – в главном окне во время его обработки высвечивается надпись: «Выполняю…» (рисунок 4.3), если не выбрано, была нажата клавиша «отмена», - высвечивается надпись: «Вы не выбрали изображение. Попробуйте еще раз.» (рисунок 4.4).

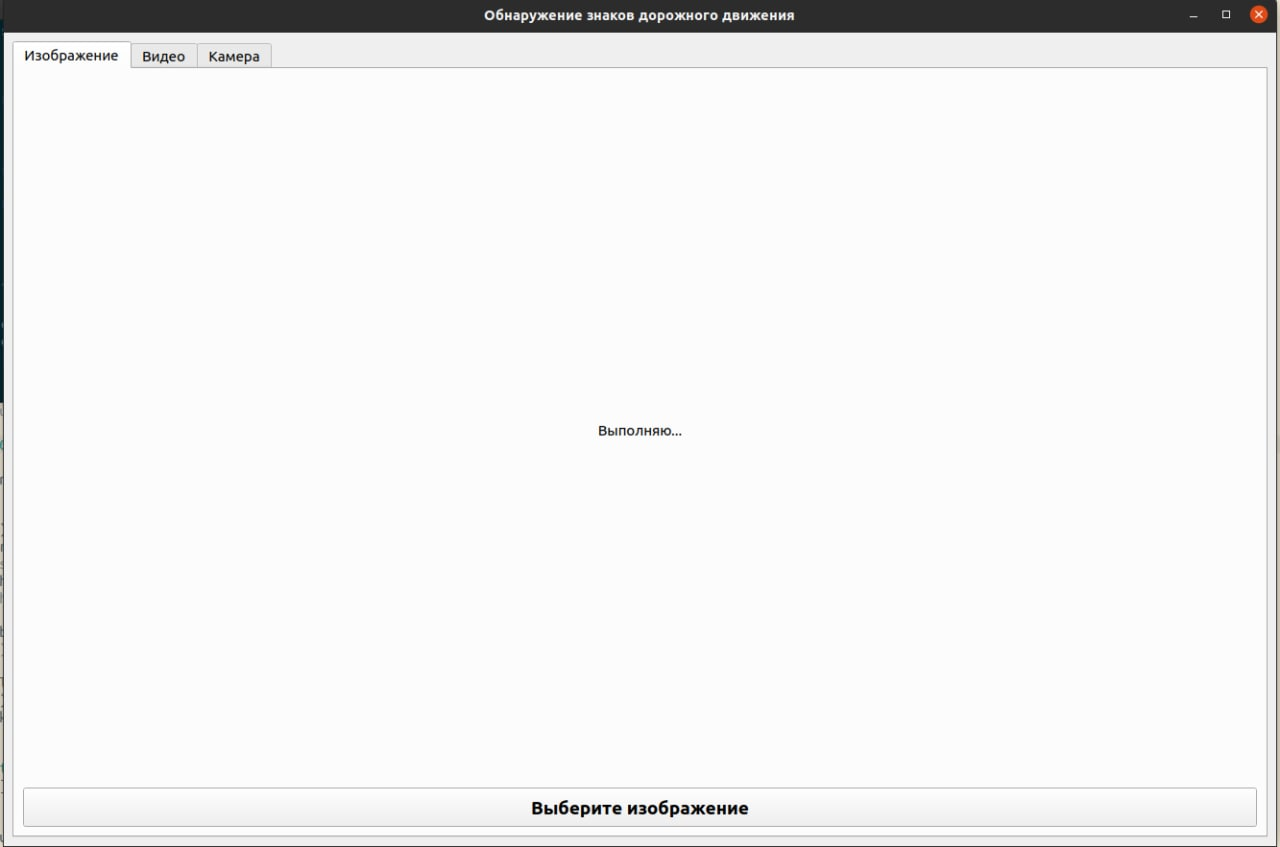


Рисунок 4.3 – выполнение обработки входных данных.

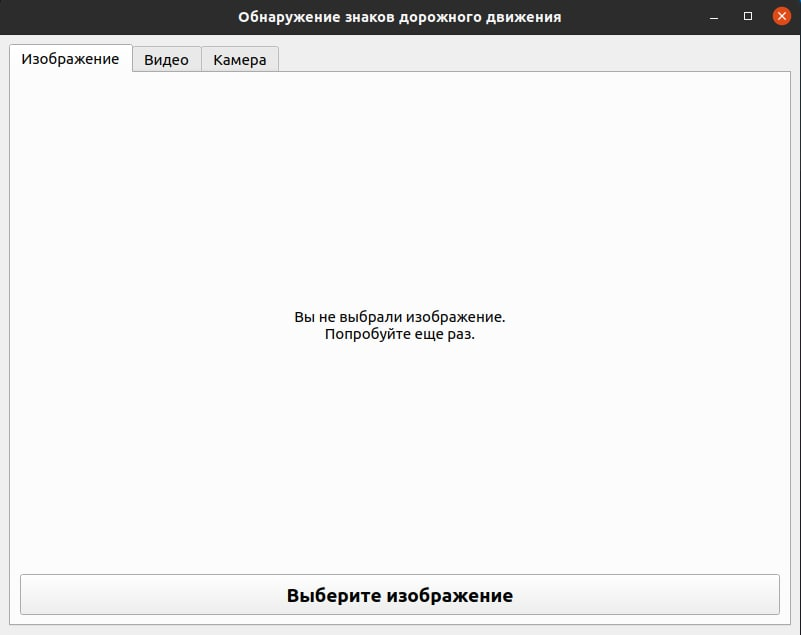


Рисунок 4.4 – вывод предупреждения, что файл с изображением не выбран.

Если изображение было выбрано, то после его обработки оно выводится на экран в этой же вкладке (рисунок 4.5). Это удобно так как пользователю нет необходимости следовать в какие-либо директории и искать файл с получившимся результатом. Также размер самого окна подстраивается под размер итогового изображения, и нет необходимости вручную изменять его размер, но такая возможность имеется при необходимости. В таком случае все элементы сохраняют свои места, приложение адаптировано под различные размеры экранов.

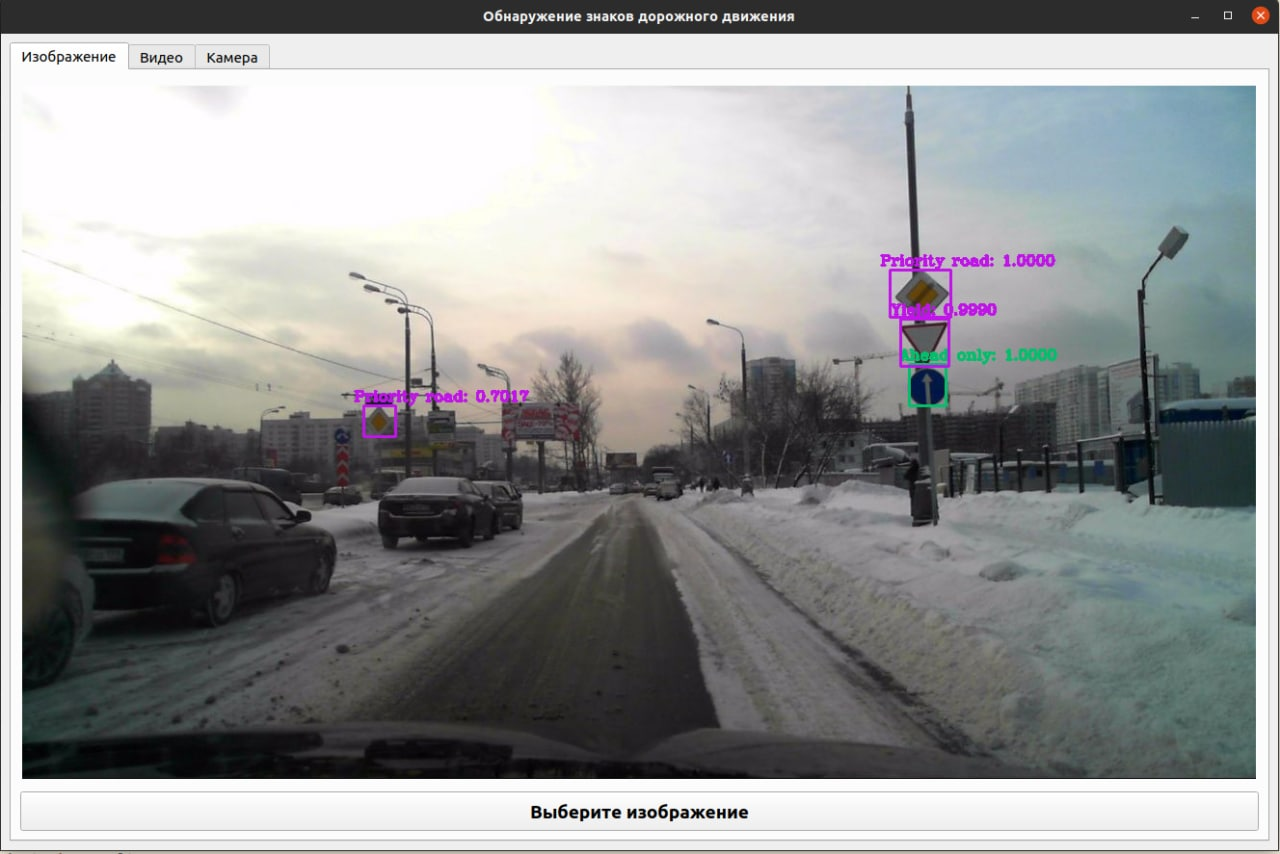


Рисунок 4.5 – вывод обработанного изображения.

Также выводится информационное сообщение с временем обработки изображения:

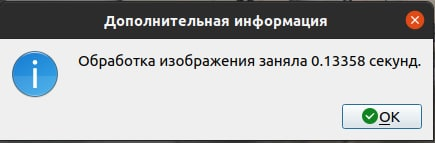


Рисунок 4.6 – информационное окно после обработки изображения.

Если выбрана вкладка «Видео», то действия схожи. Только теперь после нажатия кнопки «Выберите видео» формат входящих файлов будет ограничен лишь видео-форматом «.mp4» (рисунок 4.7). Окно ожидания обработки схожее (рисунок 4.8).

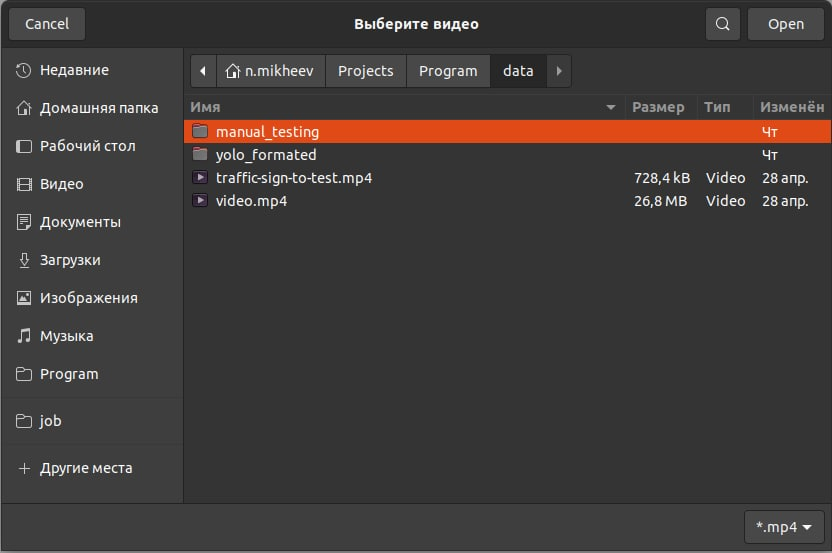


Рисунок 4.7 – окно выбора видео

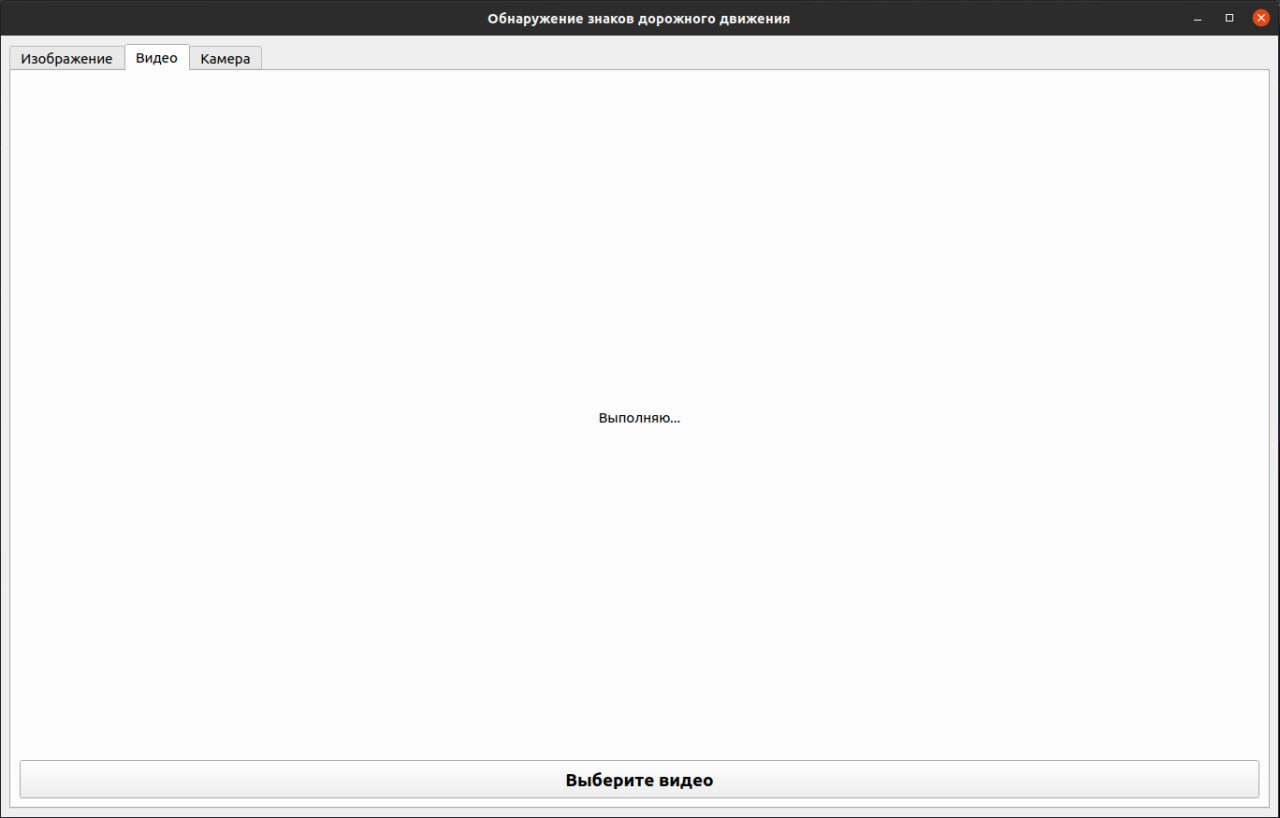


Рисунок 4.8 – ожидание обработки видео

После того как видео было обработано, на экране отображается окно со вспомогательной информацией, где указано количество обработанных кадров, среднее время обработки одного кадра и скорость в кадрах в секунду (рисунок 4.9). Продолжив нажатием клавиши «Ок», открывается окно с видео - результатом обработки (рисунок 4.10).

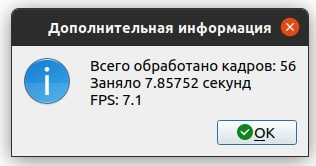
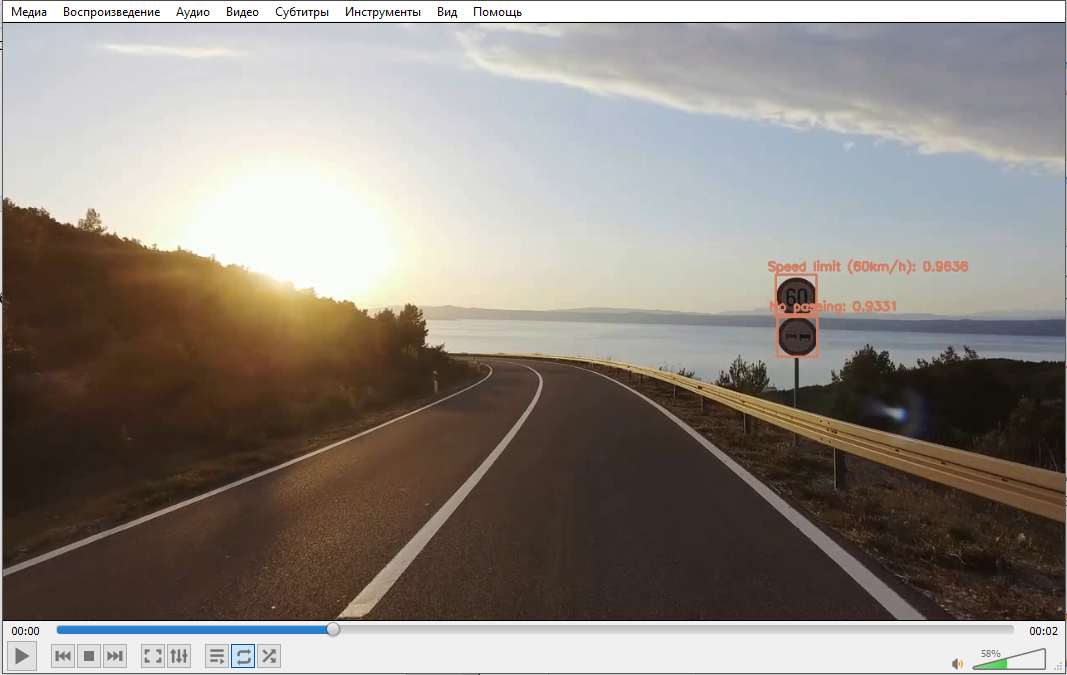


Рисунок 4.9 – дополнительная информация после обработки видео



На видео есть два знака: ограничение скорости 60 км\ч и обгон запрещен. Как видно из текстовых полей вокруг рамок – программа отработала корректно.

Рисунок 4.10 – результат работы программы

Во вкладке изменяется отображаемое сообщение, в котором теперь говорится об успешности прошедшей обработки и предлагается выбрать новое видео:

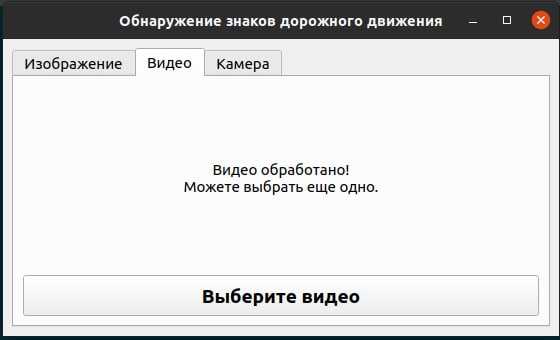


Рисунок 4.11 – изменения во вкладке «Видео»

Переключившись на вкладку «Камера», пользователя встречает сообщение о том, что необходимо посмотреть в новое открывшееся окно, где представлено изображение с камеры устройства (рисунок 4.12). На нем сразу отображаются найденные обработчиком знаки дорожного движения в рамке и с подписью (рисунок 4.13). Завершив работу, можно закрыть окно камеры нажатием клавиши «Й» или «Q» на клавиатуре или нажатием крестика в правом верхнем углу.

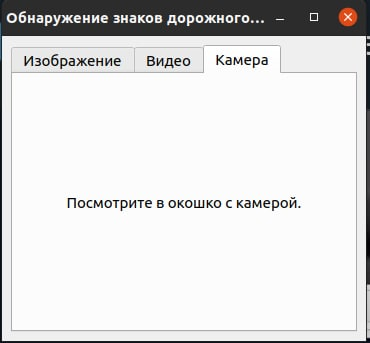


Рисунок 4.12 – сообщение во вкладке «Камера»

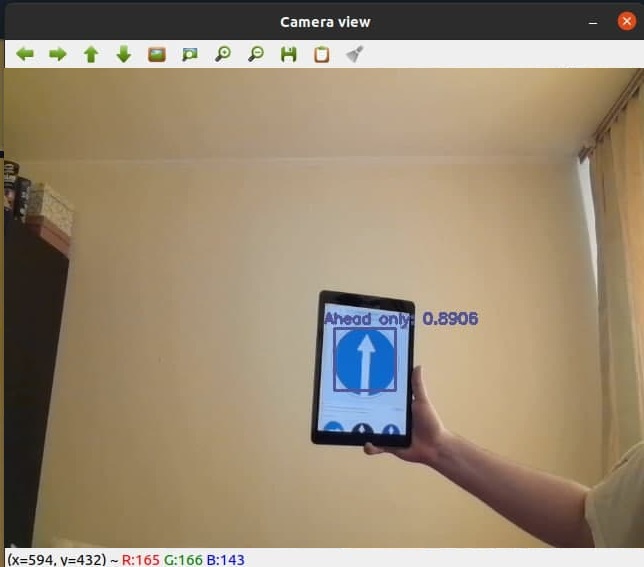


Рисунок 4.13 – изображение с камеры

После работы обработчика на экране также отображается информационной окно с информацией об объеме проведенной работы приложения (рисунок 4.14).

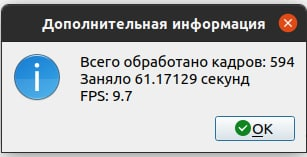


Рисунок 4.14 – дополнительная информация после обработки видео с камеры

В результате проведенного тестирования установлено, что готовое приложение полностью удовлетворяет поставленным целям. Ошибок в работе программы обнаружено не было. Что касается точности работы программы на тестовых данных, можно оценить ее на оценку «Хорошо». Программа отрабатывала точно и быстро, но не всегда все представленные на изображении знаки находились и правильно классифицировались из-за частичного перекрытия, плохого угла обзора на сам знак и иных факторов. Но тем не менее продукт полностью готов к использованию и примеры его работы можно найти на рисунке 4.15.

Также нельзя не отметить производительность программы на оборудовании среднего и домашнего уровня. Так, при наличии графического ускорителя производства компании Nvidia, система успевает обработать изображение и выдать результат менее чем за 0.2 секунды, а если обрабатывается видео или сигнал с камеры, то средняя скорость работы программы составляет 10 кадров в секунду, что вполне достаточно для работы системы в реальном времени.



На рисунке: 1. Результат работы системы на изображении, все знаки определены и классифицированы корректно; 2-4 работа системы с сигналом с камеры устройства с различными видами знаков, как видно – отработано корректно.

Рисунок 4.15 – демонстрация работы программы

# зАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы было спроектировано и программно реализовано приложение для определения дорожных знаков ограничения скорости и других. Для этого были изучены и использованы свёрточные нейронные сети, и на их основе были созданы две модели – модель-обнаружитель и модель-классификатор. Приложение может работать с различными типами подаваемых данных на вход: изображение, видео или сигнал с веб-камеры персонального компьютера.

Также были проанализированы схожие по функционалу и возможностям системы, на основе этого были выделены их достоинства и недостатки, которые были учтены при разработке выпускной квалификационной работы.

В ходе работы были изучены и выбраны основные программные средства, с помощью которых разрабатывалось приложение. Их выбор повлиял на скорость разработки, удобство, а также на взаимодействие предполагаемого пользователя с программой.

Готовый продукт может использоваться в качестве системы помощи для водителя, так как обладает достаточным функционалом, хорошей скоростью работы и высоким уровнем точности, что было подтверждено на этапе тестирования программы.

В дальнейшем можно будет расширить функционал системы до обнаружения цветов сигналов светофоров, добавить звуковую и голосовую озвучку для предупреждения водителя об обнаруженных объектах.

# Список источников

1. Как работает система распознавания дорожных знаков [Электронный ресурс]. URL: https://techautoport.ru/sistemybezopasnosti/aktivnaya/sistema-raspoznavaniya-dorozhnyhznakov.html (Дата обращения: 20.04.2021).
2. Любанович Билл. Простой Python. Современный стиль программирования.: ООО «Питер Пресс», 2017. 480 с.
3. Выбираем самый удобный редактор кода Python [Электронный ресурс]. URL: https://habr.com/ru/company/skillfactory/blog/521838/ (дата обращения: 20.04.2021).
4. Mattmann A. Chris. Machine Learning with TensorFlow, 2021. 456 p.
5. NumPy в Python. Часть 1. [Электронный ресурс]. URL: https://habr.com/ru/post/352678/ (дата обращения: 20.04.2021).
6. Introduction to OpenCV. [Электронный ресурс]. URL: https://docs.opencv.org/master/d1/dfb/intro.html (дата обращения 20.04.2021).
7. GUI Programming in Python. [Электронный ресурс]. URL: https://wiki.python.org/moin/GuiProgramming (дата обращения 20.04.2021).
8. Francois Chollet. Deep Learning with Python, 2018. 386 p.
9. Владислав Шахуро и Антон Конушин. Российская база изображений автодорожных знаков. Компьютерная оптика, 2016. 294-300 с, ISSN-0134-2452.
10. How to train (to detect your custom objects) [Электронный ресурс]. URL: <https://github.com/AlexeyAB/darknet#how-to-train-to-detect-your-custom-objects> (дата обращения: 10.05.2021).

# приложение А

Наиболее употребляемые текстовые сокращения

YOLO – You Only Look Once (c англ. Смотрите только один раз), RGB – Red, Green, Blue – аддитивная цветовая модель, CUDA - Compute Unified Device Architecture (c англ. Унифицированная архитектура вычислительных устройств) – программно-аппаратная архитектура параллельных вычислений, FPS – frames per second (с англ. Кадров в секунду)

# Приложение Б

Листинг программы

Листинг Б.1 – вспомогательная программа «convert\_to\_yolo.py»:

import pandas as pd

import os

import cv2

import numpy as np

os.cur

full\_path\_to\_ts\_dataset = 'C:\\Users\\Lolimpo\\Desktop\\dataset'

# Считываем всю информацию о дата-сете

ann1 = pd.read\_csv(full\_path\_to\_ts\_dataset + '\\' + 'rtsd-d1-gt\\danger\\test\_gt.csv')

ann2 = pd.read\_csv(full\_path\_to\_ts\_dataset + '\\' + 'rtsd-d1-gt\\mandatory\\test\_gt.csv')

ann3 = pd.read\_csv(full\_path\_to\_ts\_dataset + '\\' + 'rtsd-d1-gt\\prohibitory\\test\_gt.csv')

ann = pd.concat([ann1, ann2, ann3], join='inner')

classes = pd.read\_csv(full\_path\_to\_ts\_dataset + '\\rtsd-r1\\numbers\_to\_classes.csv', header=0, index\_col=0, squeeze=True).to\_dict()

print(classes)

ann['center x'] = ''

ann['center y'] = ''

ann['class id'] = ''

ann['center x'] = (ann['x\_from'] + (ann['x\_from'] + ann['width'])) / 2

ann['center y'] = (ann['y\_from'] + (ann['y\_from'] + ann['height'])) / 2

values = []

for i in ann['sign\_class']:

for k, v in classes.items():

if i == v:

values.append(k)

print(i, k)

ann['class id'] = values

print(ann.head(10))

r = ann.loc[:, ['filename',

'class id',

'center x',

'center y',

'width',

'height']].copy()

print(r.head(5))

# Переходим в другую директорию

os.chdir(full\_path\_to\_ts\_dataset + '\\rtsd-d1-frames')

# Проходимся по всем файлам, подсчитываем значения для модели-YOLO, сохраняем аннотацию рядом с исходным изображением.

for cur\_dir, dirs, files in os.walk('..'):

for f in files:

if f.endswith('.jpg'):

image\_name = f[:-4]

image = cv2.imread(os.getcwd() + '\\test\\' + f)

h, w = image.shape[:2]

sub\_r = r.loc[r['filename'] == f].copy()

sub\_r['center x'] = sub\_r['center x'] / w

sub\_r['center y'] = sub\_r['center y'] / h

sub\_r['width'] = sub\_r['width'] / w

sub\_r['height'] = sub\_r['height'] / h

resulted\_frame = sub\_r.loc[:, ['class id',

'center x',

'center y',

'width',

'height']].copy()

if resulted\_frame.isnull().values.all():

continue

path\_to\_save = full\_path\_to\_ts\_dataset + '\\yolo\_formated\\' + image\_name + '.txt'

resulted\_frame.to\_csv(path\_to\_save, header=False, index=False, sep=' ')

path\_to\_save = full\_path\_to\_ts\_dataset + '\\yolo\_formated\\' + image\_name + '.jpg'

cv2.imwrite(path\_to\_save, image)

Листинг Б.2 – обучение модели-классификатора «classificator.ipynb»:

import random  
  
import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import pickle as pkl  
  
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau  
  
#%%  
  
df\_labels = pd.read\_csv*(*'../data/label\_names.csv'*)*train\_dir = "../data/yolo\_formated"

datagen = ImageDataGenerator(validation\_split=0.2, rescale=1./255)

train\_generator = datagen.flow\_from\_directory(

train\_dir,

subset='training',

target\_size=(75,75),

batch\_size=32,

color\_mode='rgb',

shuffle=True,

seed=42,

class\_mode='categorical')

val\_generator = datagen.flow\_from\_directory(

train\_dir,

subset='validation',

target\_size=(75,75),

batch\_size=32,

color\_mode='rgb',

shuffle=True,

seed=42,

class\_mode='categorical')

#%%  
  
labels\_dict = dict*(*zip*(*list*(*df\_labels.ClassId*)*, list*(*df\_labels.SignName*)))*print*(*labels\_dict*)*#%%  
  
X\_train = train\_imgs*[*'features'*]*Y\_train = train\_imgs*[*'labels'*]*X\_val = valid\_imgs*[*'features'*]*Y\_val = valid\_imgs*[*'labels'*]*X\_test = test\_imgs*[*'features'*]*Y\_test = test\_imgs*[*'labels'*]*#%%  
  
def shapes*(*X, Y*)*:  
 print*(*'X: ', X.shape*)* print*(*'Y: ', Y.shape*)*shapes*(*X\_train, Y\_train*)*shapes*(*X\_val, Y\_val*)*shapes*(*X\_test, Y\_test*)*#%%  
  
fig, ax = plt.subplots*(*3*)*ax*[*0*]*.imshow*(*X\_train*[*0*])*ax*[*1*]*.imshow*(*X\_val*[*20*])*ax*[*2*]*.imshow*(*X\_test*[*3*])*#%%  
  
# Определение модели и ее типа - Секвентальная см линейным стеком слоев.  
model = Sequential*([* # Шаг №1 - Слой свертки. Форма на вход: 32х32х3, где 3 - количетсво каналов (RGB).  
 Conv2D*(*32, kernel\_size=*(*3, 3*)*, input\_shape=*(*32, 32, 3*)*, activation='relu'*)*,  
 # Шаг №2 - Слой подвыборки с извлечением максимума. Размеры матрицы: 2х2.  
 MaxPooling2D*(*2, 2*)*,  
 # Шаг №3 - Исключение. Регуляризируем дабы не было переобучения.  
 Dropout*(*0.2*)*,  
 # Шаг №4 - Второй слой свертки.  
 Conv2D*(*64, kernel\_size=*(*3, 3*)*, activation='relu'*)*,  
 # Шаг №5 - Снова подвыборка после второго сверточного слоя и также с извлечением максимума.  
 MaxPooling2D*(*2, 2*)*,  
 # Шаг №6 - Снова исключение.  
 Dropout*(*0.2*)*,  
 # Шаг №7 - Выравниваем последний полносвязный слой, конвертируем в меньшую размерность  
 Flatten*()*,  
 # Шаг №8 - Получаем информацию со всех предыдущих слоев в слой с 256 нейронами.  
 Dense*(*256, activation='relu'*)*,  
 # Шаг №9 - Исключение.  
 Dropout*(*0.2*)*,  
 # Шаг №10 - Тот же слой, только нейронов меньше - 128.  
 Dense*(*128, activation='relu'*)*,  
 # Шаг №11 - Исключение.  
 Dropout*(*0.2*)*,  
 # Шаг №12 - Последний Dense слой с 64 нейронами и другим активатором.  
 Dense*(*64, activation='softmax'*)  
])*# Собираем модель, оптимизатор Adam - как наиболее удачный из протестированных,  
# функция loss по-стандарту и метрики на точность  
model.compile*(*optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=*[*'accuracy'*])*#%%  
  
model.summary*()*#%%  
  
# Уменьшаем показатель переобучения отслеживая параметр val\_loss.  
reduce\_LR = ReduceLROnPlateau*(*monitor='val\_loss', factor=0.3, mode='min', verbose=1, min\_lr=1e-5*)*# Функция задания чекпоинта с сохраненим лучших результатов модели в файл  
checkpoint = ModelCheckpoint*(*'traffic\_model.h5', save\_best\_only=True, monitor='val\_loss', mode='min'*)*# Функция ранней остановки, если модель достаточно обучена и дальнейшее обучение не требуется  
early\_stopping = EarlyStopping*(*monitor='val\_loss', restore\_best\_weights=True, patience=5, mode='min'*)*#%%  
  
hist = model.fit*(*X\_train, Y\_train, epochs=20, validation\_data=*(*X\_test, Y\_test*)*, batch\_size=32,  
 callbacks=*[*early\_stopping, checkpoint, reduce\_LR*])*#%%  
  
plt.plot*(*hist.history*[*'accuracy'*]*, label='Обучающая выборка'*)*plt.plot*(*hist.history*[*'val\_accuracy'*]*, label='Тестовая выборка'*)*plt.rcParams*[*'figure.figsize'*]* = plt.rcParamsDefault*[*'figure.figsize'*]*plt.xlabel*(*'Эпоха'*)*plt.ylabel*(*'Точность'*)*plt.legend*([*'Обучающая выборка', 'Тестовая выборка'*])*plt.savefig*(*'plt.png'*)*#%%  
# Попробуем предсказать одно случайное изображение из датасета   
image\_index = 5555  
x\_input = X\_test*[*image\_index:image\_index + 1*]*plt.rcParams*[*'figure.figsize'*]* = *(*2.5, 2.5*)*plt.imshow*(*x\_input*[*0, :, :, :*])*plt.axis*(*'off'*)*scores = model.predict*(*x\_input*)*print*(*scores*[*0*]*.shape*)*prediction = np.argmax*(*scores*)*print*(*'Class id:', labels\_dict*[*prediction*])*#%%  
  
# Задаем размер рисунка  
plt.figure*(*figsize=*(*15, 15*))*for i in range*(*25*)*:  
 # Случайный индекс изображения  
 start\_index = random.randint*(*0, 10000*)* plt.subplot*(*5, 5, i + 1*)* plt.xticks*([])* plt.yticks*([])* # Берем изображение на вход  
 x\_input = X\_test*[*start\_index:start\_index+1*]* # Составляем предсказание  
 prediction = labels\_dict*[*np.argmax*(*model.predict*(*x\_input*))]* # Получаем класс знака на самом деле  
 actual = labels\_dict*[*Y\_test*[*start\_index*]]* col = 'g' if prediction == actual else 'r'  
 # Подписываем с определенным цветом текста в зависимости от правильности предсказания  
 plt.xlabel*(*f'На деле: *{*actual*}*\nМодель: *{*prediction*}*', color=col*)* plt.imshow*(*X\_test*[*start\_index*])*# Выводим рисунок и сохраняем  
plt.show*()*plt.savefig*(*'../training\_examples.png'*)*

Листинг Б.3 – вспомогательный модуль – «processing.py»:

from datetime import datetime  
import time  
  
import numpy as np  
import pandas as pd  
import cv2  
import pickle  
  
from tensorflow.keras.models import load\_model  
  
# Считываем все названия классов  
labels = pd.read\_csv*(*'data/label\_names\_eng.csv'*)*# Загружаем готовую модель-детектор знаков с конфигурационным файлом  
network = cv2.dnn.readNetFromDarknet*(*'data/yolov3\_ts\_test.cfg', 'data/znaki\_rtx\_final.weights'*)*# Включаем дополнительные настройки графического ускорителя для увеличения скорости рабоыт программы  
network.setPreferableBackend*(*cv2.dnn.DNN\_BACKEND\_CUDA*)*network.setPreferableTarget*(*cv2.dnn.DNN\_TARGET\_CUDA\_FP16*)*layers\_names\_all = network.getLayerNames*()*# print(layers\_names\_all)  
  
# Загружаем модель-классификатор  
model = load\_model*(*'models/model-new.h5'*)*with open*(*'data/mean\_image\_rgb.pickle', 'rb'*)* as f:  
 mean = pickle.load*(*f, encoding='latin1'*)*# print(mean['mean\_image\_rgb'].shape)  
  
layers\_names\_output = *[*layers\_names\_all*[*i*[*0*]* - 1*]* for i in network.getUnconnectedOutLayers*()]*# print(layers\_names\_output)  
  
# Минимальная вероятность чтобы избавиться от слабых обнаружений  
probability\_minimum = 0.5  
# Устанавливаем минимальный порог для фильтрации слабых граничных рамок методом подавления немаксимумов  
threshold = 0.3  
# Генерируем массив цветов рамок для каждого из классов  
colours = np.random.randint*(*0, 255, size=*(*len*(*labels*)*, 3*)*, dtype='uint8'*)*def process\_image*(*path*)*:  
 *""" Метод в котором обрабатывается изображение """* # Считываем изображение  
 image\_BGR = cv2.imread*(*path*)* # Получаем высоту и ширину изображения  
 h, w = image\_BGR.shape*[*:2*]* # Получаем объект с необработанными данными из изображения  
 blob = cv2.dnn.blobFromImage*(*image\_BGR, 1 / 255.0, *(*416, 416*)*, swapRB=True, crop=False*)* # print(blob.shape)  
  
 # Проходимся по изображению через слои нашей модели и подсчитываем время обработки  
 network.setInput*(*blob*)* start = time.time*()* output\_from\_network = network.forward*(*layers\_names\_output*)* end = time.time*()* print*(*f'Objects Detection took *{*end - start:.5f*}* seconds'*)* # Определяем границы найденых знаков и отсекаем их в отдельный массив,  
 # чтобы в дальнейшем использовать на модели-классификаторе  
 bounding\_boxes = *[]* confidences = *[]* class\_numbers = *[]* for result in output\_from\_network:  
 for detected\_objects in result:  
 scores = detected\_objects*[*5:*]* class\_current = np.argmax*(*scores*)* confidence\_current = scores*[*class\_current*]* if confidence\_current > probability\_minimum:  
 box\_current = detected\_objects*[*0:4*]* \* np.array*([*w, h, w, h*])* x\_center, y\_center, box\_width, box\_height = box\_current  
 x\_min = int*(*x\_center - *(*box\_width / 2*))* y\_min = int*(*y\_center - *(*box\_height / 2*))* bounding\_boxes.append*([*x\_min, y\_min, int*(*box\_width*)*, int*(*box\_height*)])* confidences.append*(*float*(*confidence\_current*))* class\_numbers.append*(*class\_current*)* # Применяем алгоритм подавления немаксимумов для отсечения ошибочно найденных объектов  
 # и уточнения границ рамок  
 results = cv2.dnn.NMSBoxes*(*bounding\_boxes, confidences, probability\_minimum, threshold*)* # Смотрим осталось ли что-то после подавления  
 if len*(*results*)* > 0:  
 # Проходимся по все результатам  
 for i in results.flatten*()*:  
 x\_min, y\_min = bounding\_boxes*[*i*][*0*]*, bounding\_boxes*[*i*][*1*]* box\_width, box\_height = bounding\_boxes*[*i*][*2*]*, bounding\_boxes*[*i*][*3*]* # Получаем фрагмент изображения на основе предсказания предыдущей модели  
 c\_ts = image\_BGR*[*y\_min: y\_min + int*(*box\_height*)*, x\_min: x\_min + int*(*box\_width*)*, :*]* if c\_ts.shape*[*:1*]* == *(*0,*)* or c\_ts.shape*[*1:2*]* == *(*0,*)*:  
 pass  
 # Если все хорошо - получаем blob этого фрагмента  
 else:  
 blob\_ts = cv2.dnn.blobFromImage*(*c\_ts, 1 / 255.0, size=*(*32, 32*)*, swapRB=True, crop=False*)* blob\_ts*[*0*]* = blob\_ts*[*0, :, :, :*]* - mean*[*'mean\_image\_rgb'*]* blob\_ts = blob\_ts.transpose*(*0, 2, 3, 1*)* # Получаем все предсказания классов  
 scores = model.predict*(*blob\_ts*)* # Выбираем максимальное из них  
 prediction = np.argmax*(*scores*)* print*(*labels*[*'SignName'*][*prediction*])* colour\_box\_current = colours*[*class\_numbers*[*i*]]*.tolist*()* cv2.rectangle*(*image\_BGR, *(*x\_min, y\_min*)*, *(*x\_min + box\_width, y\_min + box\_height*)*,  
 colour\_box\_current, 2*)* text\_box\_current = '{}: {:.4f}'.format*(*labels*[*'SignName'*][*prediction*]*, confidences*[*i*])* # text\_box\_current = '{}: {:.4f}'.format('Кирилический-Тест 123', confidences[i])  
 cv2.putText*(*image\_BGR, text\_box\_current, *(*x\_min - 10, y\_min - 5*)*,  
 cv2.FONT\_HERSHEY\_COMPLEX, 0.5, colour\_box\_current, 2*)* cv2.imwrite*(*'result.png', image\_BGR*)* return True, end - start  
  
  
def process\_camera*()*:  
 cap = cv2.VideoCapture*(*1*)* # cap.set(cv2.CAP\_PROP\_FRAME\_WIDTH, 1280)  
 # cap.set(cv2.CAP\_PROP\_FRAME\_HEIGHT, 720)  
 h, w = None, None  
 t, f = 0, 0  
 while True:  
 ret, frame = cap.read*()* if not ret:  
 break  
  
 if w is None or h is None:  
 h, w = frame.shape*[*:2*]* blob = cv2.dnn.blobFromImage*(*frame, 1 / 255.0, *(*416, 416*)*, swapRB=True, crop=False*)* network.setInput*(*blob*)* start = time.time*()* output\_from\_network = network.forward*(*layers\_names\_output*)* end = time.time*()* f += 1  
 t += end - start  
 print*(*'Frame number {0} took {1:.5f} seconds'.format*(*f, end - start*))* bounding\_boxes = *[]* confidences = *[]* class\_numbers = *[]* for result in output\_from\_network:  
 for detected\_objects in result:  
 scores = detected\_objects*[*5:*]* class\_current = np.argmax*(*scores*)* confidence\_current = scores*[*class\_current*]* if confidence\_current > probability\_minimum:  
 box\_current = detected\_objects*[*0:4*]* \* np.array*([*w, h, w, h*])* x\_center, y\_center, box\_width, box\_height = box\_current  
 x\_min = int*(*x\_center - *(*box\_width / 2*))* y\_min = int*(*y\_center - *(*box\_height / 2*))* bounding\_boxes.append*([*x\_min, y\_min, int*(*box\_width*)*, int*(*box\_height*)])* confidences.append*(*float*(*confidence\_current*))* class\_numbers.append*(*class\_current*)* results = cv2.dnn.NMSBoxes*(*bounding\_boxes, confidences, probability\_minimum, threshold*)* if len*(*results*)* > 0:  
 for i in results.flatten*()*:  
 x\_min, y\_min = bounding\_boxes*[*i*][*0*]*, bounding\_boxes*[*i*][*1*]* box\_width, box\_height = bounding\_boxes*[*i*][*2*]*, bounding\_boxes*[*i*][*3*]* c\_ts = frame*[*y\_min:y\_min+int*(*box\_height*)*, x\_min:x\_min+int*(*box\_width*)*, :*]* if c\_ts.shape*[*:1*]* == *(*0,*)* or c\_ts.shape*[*1:2*]* == *(*0,*)*:  
 pass  
 else:  
 blob\_ts = cv2.dnn.blobFromImage*(*c\_ts, 1 / 255.0, size=*(*32, 32*)*, swapRB=True, crop=False*)* blob\_ts*[*0*]* = blob\_ts*[*0, :, :, :*]* - mean*[*'mean\_image\_rgb'*]* blob\_ts = blob\_ts.transpose*(*0, 2, 3, 1*)* scores = model.predict*(*blob\_ts*)* prediction = np.argmax*(*scores*)* colour\_box\_current = colours*[*class\_numbers*[*i*]]*.tolist*()* cv2.rectangle*(*frame, *(*x\_min, y\_min*)*, *(*x\_min + box\_width, y\_min + box\_height*)*,  
 colour\_box\_current, 2*)* text\_box\_current = '{}: {:.4f}'.format*(*labels*[*'SignName'*][*prediction*]*, confidences*[*i*])* cv2.putText*(*frame, text\_box\_current, *(*x\_min - 10, y\_min - 5*)*,  
 cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, colour\_box\_current, 2*)* cv2.namedWindow*(*'Image from camera', cv2.WINDOW\_AUTOSIZE*)* cv2.imshow*(*'Camera view', frame*)* if cv2.waitKey*(*1*)* & 0xFF == ord*(*'q'*)* or ord*(*'й'*)*:  
 break  
  
 cap.release*()* cv2.destroyAllWindows*()* return f, t  
  
  
def process\_video*(*path*)*:  
 video = cv2.VideoCapture*(*path*)* writer = None  
 h, w = None, None  
 frames, t = 0, 0  
 while True:  
 ret, frame = video.read*()* if not ret:  
 break  
  
 if w is None or h is None:  
 h, w = frame.shape*[*:2*]* blob = cv2.dnn.blobFromImage*(*frame, 1 / 255.0, *(*416, 416*)*, swapRB=True, crop=False*)* network.setInput*(*blob*)* start = time.time*()* output\_from\_network = network.forward*(*layers\_names\_output*)* end = time.time*()* frames += 1  
 t += end - start  
 print*(*'Frame number {0} took {1:.5f} seconds'.format*(*f, end - start*))* bounding\_boxes = *[]* confidences = *[]* class\_numbers = *[]* for result in output\_from\_network:  
 for detected\_objects in result:  
 scores = detected\_objects*[*5:*]* class\_current = np.argmax*(*scores*)* confidence\_current = scores*[*class\_current*]* if confidence\_current > probability\_minimum:  
 box\_current = detected\_objects*[*0:4*]* \* np.array*([*w, h, w, h*])* x\_center, y\_center, box\_width, box\_height = box\_current  
 x\_min = int*(*x\_center - *(*box\_width / 2*))* y\_min = int*(*y\_center - *(*box\_height / 2*))* bounding\_boxes.append*([*x\_min, y\_min, int*(*box\_width*)*, int*(*box\_height*)])* confidences.append*(*float*(*confidence\_current*))* class\_numbers.append*(*class\_current*)* results = cv2.dnn.NMSBoxes*(*bounding\_boxes, confidences, probability\_minimum, threshold*)* if len*(*results*)* > 0:  
 for i in results.flatten*()*:  
 x\_min, y\_min = bounding\_boxes*[*i*][*0*]*, bounding\_boxes*[*i*][*1*]* box\_width, box\_height = bounding\_boxes*[*i*][*2*]*, bounding\_boxes*[*i*][*3*]* c\_ts = frame*[*y\_min:y\_min+int*(*box\_height*)*, x\_min:x\_min+int*(*box\_width*)*, :*]* if c\_ts.shape*[*:1*]* == *(*0,*)* or c\_ts.shape*[*1:2*]* == *(*0,*)*:  
 pass  
 else:  
 blob\_ts = cv2.dnn.blobFromImage*(*c\_ts, 1 / 255.0, size=*(*32, 32*)*, swapRB=True, crop=False*)* blob\_ts*[*0*]* = blob\_ts*[*0, :, :, :*]* - mean*[*'mean\_image\_rgb'*]* blob\_ts = blob\_ts.transpose*(*0, 2, 3, 1*)* scores = model.predict*(*blob\_ts*)* prediction = np.argmax*(*scores*)* colour\_box\_current = colours*[*class\_numbers*[*i*]]*.tolist*()* cv2.rectangle*(*frame, *(*x\_min, y\_min*)*, *(*x\_min + box\_width, y\_min + box\_height*)*,  
 colour\_box\_current, 2*)* text\_box\_current = '{}: {:.4f}'.format*(*labels*[*'SignName'*][*prediction*]*, confidences*[*i*])* cv2.putText*(*frame, text\_box\_current, *(*x\_min - 10, y\_min - 5*)*,  
 cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, colour\_box\_current, 2*)* if writer is None:  
 fourcc = cv2.VideoWriter\_fourcc*(*\*'mp4v'*)* writer = cv2.VideoWriter*(*'result.mp4', fourcc, 25,  
 *(*frame.shape*[*1*]*, frame.shape*[*0*])*, True*)* writer.write*(*frame*)* video.release*()* writer.release*()* return True, frames, t

Листинг Б.4 – основной модуль программы «main.py»:

import os  
import sys  
  
import cv2  
from PyQt5 import QtWidgets  
from PyQt5.QtGui import QPixmap  
import design  
import processing  
  
  
# Основной класс главного окна  
class MainApp*(*QtWidgets.QMainWindow, design.Ui\_MainWindow*)*:  
  
 # Инициализирующая функция  
 def \_\_init\_\_*(*self*)*:  
 # Ключевая точка для старта  
 super*()*.\_\_init\_\_*()* # Установка дизайна  
 self.setupUi*(*self*)* # Установка начальной вкладки после запуска приложения  
 self.tabWidget.setCurrentIndex*(*0*)* # Установка фукнции обработки изображения на сигнал нажатия кнопки  
 self.pushButton.clicked.connect*(*self.process\_image*)* # Установка фукнции обработки видео на сигнал нажатия кнопки  
 self.pushButton\_2.clicked.connect*(*self.process\_video*)* # Установка функции обработки видео с камеры на сигнал переключения соотвествующей вкладки  
 self.tabWidget.currentChanged.connect*(*self.selector*)* # Установка начальных значений информационных полей в программе  
 self.label.setText*(*''*)* self.label\_2.setText*(*''*)* def selector*(*self*)*:  
 if self.tab\_3 is self.tabWidget.currentWidget*()*:  
 self.tabWidget.setCurrentIndex*(*2*)* self.process\_camera*()* def process\_image*(*self*)*:  
 self.label.setText*(*'Выполняю...'*)* image\_path = QtWidgets.QFileDialog.getOpenFileName*(*self, 'Выберите изображение', '.', '\*.png \*.jpg \*.bmp'*)* if image\_path*[*0*]* != '':  
 image\_path = image\_path*[*0*]* else:  
 self.label.setText*(*'Вы не выбрали изображение. \nПопробуйте еще раз.'*)* return  
  
 flag, t = processing.process\_image*(*image\_path*)* if flag:  
 pixmap\_image = QPixmap*(*'result.png'*)* self.label.setPixmap*(*pixmap\_image*)* self.label.resize*(*pixmap\_image.width*()*, pixmap\_image.height*())* self.window*()*.resize*(*pixmap\_image.width*()*, pixmap\_image.height*())* msg = QtWidgets.QMessageBox*()* msg.setIcon*(*QtWidgets.QMessageBox.Information*)* msg.setText*(*f'Обработка изображения заняла *{*t:.5f*}* с'*)* msg.setWindowTitle*(*'Дополнительная информация'*)* msg.exec\_*()* def process\_video*(*self*)*:  
 self.window*()*.resize*(*window.baseSize*())* self.label\_2.setText*(*'Выполняю...'*)* video\_path = QtWidgets.QFileDialog.getOpenFileName*(*self, 'Выберите видео', '.', '\*.mp4'*)* if video\_path != '':  
 video\_path = video\_path*[*0*]* else:  
 self.label\_2.setText*(*'Вы не выбрали видео.\nПопробуйте еще раз.'*)* return  
  
 flag, f, t = processing.process\_video*(*video\_path*)* if flag:  
 msg = QtWidgets.QMessageBox*()* msg.setIcon*(*QtWidgets.QMessageBox.Information*)* msg.setText*(*f'Всего обработано кадров: *{*f*}*\nЗаняло *{*t:.5f*}* секунд\nFPS: *{*round*((*f / t*)*, 1*)}*'*)* msg.setWindowTitle*(*'Дополнительная информация'*)* msg.exec\_*()* self.label\_2.setText*(*'Видео обработано! \nМожете выбрать еще одно.'*)* os.startfile*(*'result.mp4'*)* def process\_camera*(*self*)*:  
 self.window*()*.resize*(*window.baseSize*())* self.label\_3.setText*(*'Посмотрите в окошко с камерой.'*)* f, t = processing.process\_camera*()* self.window*()*.resize*(*window.baseSize*())* msg = QtWidgets.QMessageBox*()* msg.setIcon*(*QtWidgets.QMessageBox.Information*)* msg.setText*(*f'Всего обработано кадров: *{*f*}*\nЗаняло *{*t:.5f*}* секунд\nFPS: *{*round*((*f / t*)*, 1*)}*'*)* msg.setWindowTitle*(*'Дополнительная информация'*)* msg.exec\_*()* self.label\_3.setText*(*''*)*if \_\_name\_\_ == '\_\_main\_\_':  
 app = QtWidgets.QApplication*(*sys.argv*)* window = MainApp*()* window.show*()* sys.exit*(*app.exec*())*

Листинг Б.5 – класс с дизайном программы «design.py»:

# -\*- coding: utf-8 -\*-

# Form implementation generated from reading ui file '.\design.ui'

#

# Created by: PyQt5 UI code generator 5.9.2

#

# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui\_MainWindow(object):

def setupUi(self, MainWindow):

MainWindow.setObjectName("MainWindow")

MainWindow.resize(800, 600)

MainWindow.setMinimumSize(QtCore.QSize(228, 200))

MainWindow.setMaximumSize(QtCore.QSize(1920, 1080))

self.centralwidget = QtWidgets.QWidget(MainWindow)

self.centralwidget.setObjectName("centralwidget")

self.verticalLayout = QtWidgets.QVBoxLayout(self.centralwidget)

self.verticalLayout.setObjectName("verticalLayout")

self.tabWidget = QtWidgets.QTabWidget(self.centralwidget)

self.tabWidget.setObjectName("tabWidget")

self.tab = QtWidgets.QWidget()

self.tab.setObjectName("tab")

self.verticalLayout\_2 = QtWidgets.QVBoxLayout(self.tab)

self.verticalLayout\_2.setObjectName("verticalLayout\_2")

self.label = QtWidgets.QLabel(self.tab)

self.label.setAlignment(QtCore.Qt.AlignCenter)

self.label.setObjectName("label")

self.verticalLayout\_2.addWidget(self.label)

self.pushButton = QtWidgets.QPushButton(self.tab)

self.pushButton.setMinimumSize(QtCore.QSize(241, 41))

self.pushButton.setMaximumSize(QtCore.QSize(16777215, 41))

font = QtGui.QFont()

font.setPointSize(14)

font.setBold(True)

font.setWeight(75)

self.pushButton.setFont(font)

self.pushButton.setObjectName("pushButton")

self.verticalLayout\_2.addWidget(self.pushButton)

self.tabWidget.addTab(self.tab, "")

self.tab\_2 = QtWidgets.QWidget()

self.tab\_2.setObjectName("tab\_2")

self.verticalLayout\_3 = QtWidgets.QVBoxLayout(self.tab\_2)

self.verticalLayout\_3.setObjectName("verticalLayout\_3")

self.label\_2 = QtWidgets.QLabel(self.tab\_2)

self.label\_2.setAlignment(QtCore.Qt.AlignCenter)

self.label\_2.setObjectName("label\_2")

self.verticalLayout\_3.addWidget(self.label\_2)

self.pushButton\_2 = QtWidgets.QPushButton(self.tab\_2)

self.pushButton\_2.setMinimumSize(QtCore.QSize(241, 41))

self.pushButton\_2.setMaximumSize(QtCore.QSize(16777215, 41))

font = QtGui.QFont()

font.setPointSize(14)

font.setBold(True)

font.setWeight(75)

self.pushButton\_2.setFont(font)

self.pushButton\_2.setObjectName("pushButton\_2")

self.verticalLayout\_3.addWidget(self.pushButton\_2)

self.tabWidget.addTab(self.tab\_2, "")

self.tab\_3 = QtWidgets.QWidget()

self.tab\_3.setObjectName("tab\_3")

self.verticalLayout\_4 = QtWidgets.QVBoxLayout(self.tab\_3)

self.verticalLayout\_4.setObjectName("label\_3")

self.label\_3 = QtWidgets.QLabel(self.tab\_3)

self.label\_3.setAlignment(QtCore.Qt.AlignCenter)

self.label\_3.setObjectName("label\_3")

self.verticalLayout\_4.addWidget(self.label\_3)

self.tabWidget.addTab(self.tab\_3, "")

self.verticalLayout.addWidget(self.tabWidget)

MainWindow.setCentralWidget(self.centralwidget)

self.retranslateUi(MainWindow)

self.tabWidget.setCurrentIndex(2)

QtCore.QMetaObject.connectSlotsByName(MainWindow)

def retranslateUi(self, MainWindow):

\_translate = QtCore.QCoreApplication.translate

MainWindow.setWindowTitle(\_translate("MainWindow", "Обнаружение знаков дорожного движения"))

self.label.setText(\_translate("MainWindow", ""))

self.pushButton.setText(\_translate("MainWindow", "Выберите изображение"))

self.tabWidget.setTabText(self.tabWidget.indexOf(self.tab), \_translate("MainWindow", "Изображение"))

self.label\_2.setText(\_translate("MainWindow", ""))

self.pushButton\_2.setText(\_translate("MainWindow", "Выберите видео"))

self.label\_3.setText(\_translate("MainWindow", ""))

self.tabWidget.setTabText(self.tabWidget.indexOf(self.tab\_2), \_translate("MainWindow", "Видео"))

self.tabWidget.setTabText(self.tabWidget.indexOf(self.tab\_3), \_translate("MainWindow", "Камера"))

Листинг Б.6 – разбиение на обучающую и тестовую выборки «make\_train\_test.py»:

import os  
  
  
path\_to\_images = r'C:\Users\Lolimpo\Google Drive\SibSUTIS\Labs\4 course\Diploma\Program\data\yolo\_formated'  
  
os.chdir*(*path\_to\_images*)*p = *[]*for cur\_dir, dirs, files in os.walk*(*'..'*)*:  
 for f in files:  
 if f.endswith*(*'.jpg'*)*:  
 path\_to\_save = path\_to\_images + '\\' + f  
 p.append*(*path\_to\_save + '\n'*)*p\_test = p*[*:int*(*len*(*p*)* \* 0.15*)]*p = p*[*int*(*len*(*p*)* \* 0.15*)*:*]*with open*(*'train.txt', 'w'*)* as t\_file:  
 for e in p:  
 t\_file.write*(*e*)*with open*(*'test.txt', 'w'*)* as t\_file:  
 for e in p\_test:  
 t\_file.write*(*e*)*c = 0  
with open*(*path\_to\_images + '\\' + 'classes.names', 'w'*)* as names, open*(*path\_to\_images + '\\' + 'classes.txt', 'r'*)* as txt:  
 for line in txt:  
 names.write*(*line*)* c += 1  
  
with open*(*path\_to\_images + '\\' + 'ts\_data.data', 'w'*)* as data:  
 data.write*(*'classes = ' + str*(*c*)* + '\n'*)* data.write*(*'train = ' + path\_to\_images + '/' + 'train.txt' + '\n'*)* data.write*(*'valid = ' + path\_to\_images + '/' + 'test.txt' + '\n'*)* data.write*(*'names = ' + path\_to\_images + '/' + 'classes.names' + '\n'*)* data.write*(*'backup = backup'*)*

Листинг Б.7 – в нем тестировались компоненты системы, модели обученных нейронных сетей, обработка видео и изображения до внесения в основную часть программы «detection.ipynb»:

import time  
  
import numpy as np  
import pandas as pd  
import cv2  
import pickle  
  
from matplotlib import pyplot as plt  
from tensorflow.keras.models import load\_model  
  
#%%  
  
labels = pd.read\_csv*(*'../data/label\_names\_eng.csv'*)*print*(*labels.head*())*#%%  
  
model = load\_model*(*'../models/model-new.h5'*)*with open*(*'../data/mean\_image\_rgb.pickle', 'rb'*)* as f:  
 mean = pickle.load*(*f, encoding='latin1'*)*print*(*mean*[*'mean\_image\_rgb'*]*.shape*)*#%%  
  
path\_to\_weights: str = '../data/znaki.weights'  
path\_to\_cfg: str = '../data/yolov3\_ts\_test.cfg'  
network = cv2.dnn.readNetFromDarknet*(*path\_to\_cfg, path\_to\_weights*)*network.setPreferableBackend*(*cv2.dnn.DNN\_BACKEND\_CUDA*)*network.setPreferableTarget*(*cv2.dnn.DNN\_TARGET\_CUDA\_FP16*)*# network.setPreferableBackend(cv2.dnn.DNN\_BACKEND\_OPENCV)  
# network.setPreferableTarget(cv2.dnn.DNN\_TARGET\_OPENCL\_FP16)  
  
#%%  
  
layers\_all = network.getLayerNames*()*print*(*layers\_all*)*layers\_names\_output = *[*layers\_all*[*i*[*0*]* - 1*]* for i in network.getUnconnectedOutLayers*()]*print*(*layers\_names\_output*)*#%%  
  
probability\_minimum = 0.2  
threshold = 0.2  
colors = np.random.randint*(*0, 255, size=*(*len*(*labels*)*, 3*)*, dtype='uint8'*)*print*(*colors.shape*)*print*(*colors*[*0*])*#%%  
  
image\_BGR = cv2.imread*(*'../data/yolo\_formated/autosave21\_01\_2013\_12\_35\_10\_2.jpg'*)*print*(*f'Input shape: *{*image\_BGR.shape*}*'*)*h, w = image\_BGR.shape*[*:2*]*blob = cv2.dnn.blobFromImage*(*image\_BGR, 1 / 255.0, *(*416, 416*)*, swapRB=True, crop=False*)*network.setInput*(*blob*)*start = time.time*()*output\_from\_network = network.forward*(*layers\_names\_output*)*end = time.time*()*print*(*f'Total time: *{*end - start:.5f*}*'*)*bounding\_boxes = *[]*confidences = *[]*class\_numbers = *[]*for result in output\_from\_network:  
 for detected\_objects in result:  
 scores = detected\_objects*[*5:*]* class\_current = np.argmax*(*scores*)* confidence\_current = scores*[*class\_current*]* if confidence\_current > probability\_minimum:  
 box\_current = detected\_objects*[*0:4*]* \* np.array*([*w, h, w, h*])* x\_center, y\_center, box\_width, box\_height = box\_current  
 x\_min = int*(*x\_center - *(*box\_width / 2*))* y\_min = int*(*y\_center - *(*box\_height / 2*))* bounding\_boxes.append*([*x\_min, y\_min, int*(*box\_width*)*, int*(*box\_height*)])* confidences.append*(*float*(*confidence\_current*))* class\_numbers.append*(*class\_current*)*results = cv2.dnn.NMSBoxes*(*bounding\_boxes, confidences, probability\_minimum, threshold*)*if len*(*results*)* > 0:  
 for i in results.flatten*()*:  
 x\_min, y\_min = bounding\_boxes*[*i*][*0*]*, bounding\_boxes*[*i*][*1*]* box\_width, box\_height = bounding\_boxes*[*i*][*2*]*, bounding\_boxes*[*i*][*3*]* c\_ts = image\_BGR*[*y\_min:y\_min+int*(*box\_height*)*, x\_min:x\_min+int*(*box\_width*)*, :*]* if c\_ts.shape*[*:1*]* == *(*0,*)* or c\_ts.shape*[*1:2*]* == *(*0,*)*:  
 pass  
 else:  
 blob\_ts = cv2.dnn.blobFromImage*(*c\_ts, 1 / 255.0, size=*(*32, 32*)*, swapRB=True, crop=False*)* blob\_ts*[*0*]* = blob\_ts*[*0, :, :, :*]* - mean*[*'mean\_image\_rgb'*]* blob\_ts = blob\_ts.transpose*(*0, 2, 3, 1*)* scores = model.predict*(*blob\_ts*)* prediction = np.argmax*(*scores*)* print*(*labels*[*'SignName'*][*prediction*])* colour\_box\_current = colors*[*class\_numbers*[*i*]]*.tolist*()* cv2.rectangle*(*image\_BGR, *(*x\_min, y\_min*)*, *(*x\_min + box\_width, y\_min + box\_height*)*,  
 colour\_box\_current, 2*)* text\_box\_current = '{}: {:.4f}'.format*(*labels*[*'SignName'*][*prediction*]*, confidences*[*i*])* cv2.putText*(*image\_BGR, text\_box\_current, *(*x\_min, y\_min - 5*)*,  
 cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, colour\_box\_current, 2*)*cv2.imwrite*(*'../result.png', image\_BGR*)*#%%  
  
%matplotlib inline  
  
plt.rcParams*[*'figure.figsize'*]* = *(*35.0, 35.0*)* # Setting default size of plots  
image\_BGR = cv2.imread*(*'../result.png'*)*print*(*'Image shape:', image\_BGR.shape*)*h, w = image\_BGR.shape*[*:2*]*print*(*f'Высота=*{*h*}* ширина=*{*w*}*'*)*plt.imshow*(*cv2.cvtColor*(*image\_BGR, cv2.COLOR\_BGR2RGB*))*plt.axis*(*'off'*)*plt.show*()*plt.close*()*#%%  
  
video = cv2.VideoCapture*(*0*)*writer = None  
h, w = None, None  
  
  
#%%  
  
%matplotlib inline  
  
plt.rcParams*[*'figure.figsize'*]* = *(*3, 3*)*f = 0  
t = 0  
  
while True:  
 ret, frame = video.read*()* if not ret:  
 break  
  
 if w is None or h is None:  
 h, w = frame.shape*[*:2*]* blob = cv2.dnn.blobFromImage*(*frame, 1 / 255.0, *(*416, 416*)*, swapRB=True, crop=False*)* network.setInput*(*blob*)* start = time.time*()* output\_from\_network = network.forward*(*layers\_names\_output*)* end = time.time*()* f += 1  
 t += end - start  
 print*(*'Frame number {0} took {1:.5f} seconds'.format*(*f, end - start*))* bounding\_boxes = *[]* confidences = *[]* class\_numbers = *[]* for result in output\_from\_network:  
 for detected\_objects in result:  
 scores = detected\_objects*[*5:*]* class\_current = np.argmax*(*scores*)* confidence\_current = scores*[*class\_current*]* if confidence\_current > probability\_minimum:  
 box\_current = detected\_objects*[*0:4*]* \* np.array*([*w, h, w, h*])* x\_center, y\_center, box\_width, box\_height = box\_current  
 x\_min = int*(*x\_center - *(*box\_width / 2*))* y\_min = int*(*y\_center - *(*box\_height / 2*))* bounding\_boxes.append*([*x\_min, y\_min, int*(*box\_width*)*, int*(*box\_height*)])* confidences.append*(*float*(*confidence\_current*))* class\_numbers.append*(*class\_current*)* results = cv2.dnn.NMSBoxes*(*bounding\_boxes, confidences, probability\_minimum, threshold*)* if len*(*results*)* > 0:  
 for i in results.flatten*()*:  
 x\_min, y\_min = bounding\_boxes*[*i*][*0*]*, bounding\_boxes*[*i*][*1*]* box\_width, box\_height = bounding\_boxes*[*i*][*2*]*, bounding\_boxes*[*i*][*3*]* c\_ts = frame*[*y\_min:y\_min+int*(*box\_height*)*, x\_min:x\_min+int*(*box\_width*)*, :*]* if c\_ts.shape*[*:1*]* == *(*0,*)* or c\_ts.shape*[*1:2*]* == *(*0,*)*:  
 pass  
 else:  
 blob\_ts = cv2.dnn.blobFromImage*(*c\_ts, 1 / 255.0, size=*(*32, 32*)*, swapRB=True, crop=False*)* blob\_ts*[*0*]* = blob\_ts*[*0, :, :, :*]* - mean*[*'mean\_image\_rgb'*]* blob\_ts = blob\_ts.transpose*(*0, 2, 3, 1*)* scores = model.predict*(*blob\_ts*)* prediction = np.argmax*(*scores*)* colour\_box\_current = colors*[*class\_numbers*[*i*]]*.tolist*()* cv2.rectangle*(*frame, *(*x\_min, y\_min*)*, *(*x\_min + box\_width, y\_min + box\_height*)*,  
 colour\_box\_current, 2*)* text\_box\_current = '{}: {:.4f}'.format*(*labels*[*'SignName'*][*prediction*]*, confidences*[*i*])* cv2.putText*(*frame, text\_box\_current, *(*x\_min, y\_min - 5*)*,  
 cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, colour\_box\_current, 2*)* cv2.namedWindow*(*'Camera view', cv2.WINDOW\_NORMAL*)* cv2.imshow*(*'Camera view', frame*)* if cv2.waitKey*(*1*)* & 0xFF == ord*(*'q'*)*:  
 break  
  
video.release*()*cv2.destroyAllWindows*()*#%%  
  
print*(*'Всего обработано кадров', f*)*print*(*f'Заняло *{*t:.5f*}* секунд'*)*print*(*'FPS:', round*((*f / t*)*, 1*))*