МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. Р.Е. АЛЕКСЕЕВА»
(НГТУ)

Институт ИРИТ
Направление подготовки (специальность)
09.03.01 Информатика и вычислительная техника
(код и наименование)
Направленность (профиль) образовательной программы
Вычислительные машины, комплексы, системы и сети
(наименование)
Кафедра Вычислительные системы и технологии

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

бакалавра
(бакалавра, магистра, специалиста)

Студента Новичкова Владислава Сергеевича группы 15-В-2
(Ф.И.О.)

на тему Программная система распознавания знаков дорожного движения
(наименование темы работы)

СТУДЕНТ

Новичков В.С.
(подпись) (фамилия, и., о.)
(дата)

РУКОВОДИТЕЛЬ

Гай В.Е.
(подпись) (фамилия, и., о.)
(дата)

РЕЦЕНЗЕНТ

(подпись) (фамилия, и., о.)
(дата)

ЗАВЕДУЮЩИЙ КАФЕДРОЙ

Мякиньков А.В
(подпись) (фамилия, и., о.)
(дата)

КОНСУЛЬТАНТЫ

1. По
(подпись) (фамилия, и., о.)
(дата)

2. По
(подпись) (фамилия, и., о.)
(дата)

3. По
(подпись) (фамилия, и., о.)
(дата)

ВКР защищена
протокол №
с оценкой

МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. Р.Е. АЛЕКСЕЕВА»
(НГТУ)

АННОТАЦИЯ

к выпускной квалификационной работе

по направлению подготовки (специальности) 09.03.01 Информатика и вычислительная техника
(код и наименование)

студента Новичкова Владислав Сергеевича группы 15-В-2
(Ф.И.О.)

по теме Программная система распознавания знаков дорожного движения

Выпускная квалификационная работа выполнена на 40 страницах, содержит 12 рисунков, библиографический список из 7 источников.

Актуальность: в настоящее время всё больше и больше компаний продолжают присоединяться к разработке полностью автономно управляемых транспортных средств. Одной из важнейших частей такого автомобиля является система распознавания знаков дорожного движения. В связи с этим было принято решение о создании её программной реализации.

Объект исследования: видеопоток, изображения.

Предмет исследования: алгоритмы распознавания объектов, алгоритмы обработки видеопотока.

Цель исследования: разработка программной системы распознавания знаков дорожного движения с использованием персонального компьютера.

Задачи исследования: исследовать существующие реализации систем распознавания объектов; разработать собственную программную систему распознавания знаков дорожного движения; выполнить тестирование разработанной системы, с целью проверки её работоспособности.

Методы исследования: классификация данных на основе полученного изображения.

Структура работы: выпускная квалификационная работа состоит из введения, пяти глав, заключения, списка литературы и приложения.

Во введении даётся описание проблемы, лежащей в основе данной работы.

В 1 разделе «Техническое задание» составлены технические требования к разрабатываемому продукту.

Во 2 разделе «Анализ поставленной задачи» производится выбор программных средств, обзор существующих подходов, даётся краткое описание алгоритма решения поставленной задачи.

В 3 разделе «Разработка системы на структурном уровне» разрабатывается структурная схема, алгоритмы решения каждого из этапов обработки данных, описываются характеристики передающей и принимающей сторон.

В 4 разделе «Разработка системы на программном уровне» разрабатываются программные средства для решения поставленной задачи.

В 5 разделе «Тестирование системы» описываются методы тестирования системы и оцениваются полученные результаты.

В заключении производятся основные выводы о проделанной работе.

Выводы:

1. Разработана программная система распознавания знаков дорожного движения.
2. Тестирование системы подтвердило её работоспособность и возможность использования для поставленной задачи.

Рекомендации:

1. Дальнейшее развитие проекта.
2. Оптимизация существующего кода.

_____/ Новичков В.С.
подпись студента /расшифровка подписи

«____» _____ 20_____ г.

МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. Р.Е. АЛЕКСЕЕВА»
(НГТУ)

Кафедра Вычислительные системы и технологии

УТВЕРЖДАЮ
Зав. кафедрой
_____ А.В. Мякиньков
«___» _____ 20___ г.

ЗАДАНИЕ

на выполнение выпускной квалификационной работы

по направлению подготовки (специальности) 09.03.01 Информатика и вычислительная техника
(код и наименование)

студенту Новичкову Владиславу Сергеевичу группы 15-В-2
(Ф.И.О.)

1. Тема ВКР Программная система дополненной реальности с многопользовательским режимом

(утверждена приказом по вузу от 28.03.2019 № 79715)

2. Срок сдачи студентом законченной работы

3. Исходные данные к работе Данные с камеры (видеопоток), набор изображений; язык разработки – Python; система включает в себя приложение для персонального компьютера и обученную модель

4. Содержание расчетно-пояснительной записки (перечень вопросов, подлежащих разработке) 1. Техническое задание
2. Анализ поставленной задачи
3. Разработка системы на структурном уровне
4. Разработка системы на программном уровне
5. Тестирование системы

5. Перечень графического материала (с точным указанием обязательных чертежей)
1. Общая структурная схема системы
2. Архитектура клиентского приложения
3. Примеры работы различных структур
4. Изображения работы приложения

6. Консультанты по ВКР (с указанием относящихся к ним разделов)

Нормоконтроль Гай В.Е.

7. Дата выдачи задания_____

Код и содержание Компетенции	Задание	Проектируемый результат	Отметка о выполнении
ПК-1, Способность разрабатывать модели компонентов информационных систем, включая модели баз данных и модели интерфейсов «человек - электронно-вычислительная машина»	Разработать структурную схему системы	Структурная схема системы	
ПК-2, Способность разрабатывать компоненты аппаратно-программных комплексов и баз данных, используя современные инструментальные средства и технологии программирования	Разработать алгоритмы работы системы, реализовать их на одном из языков программирования с помощью современной среды разработки	Схема алгоритма работы реализована на языке Python в среде разработки PyCharm Community	
ПК-3, Способность обосновывать принимаемые проектные решения, осуществлять постановку и выполнять эксперименты по проверке их корректности и эффективности	Осуществить отбор лучших результатов, оценить их качество по нескольким критериям	Осуществлён отбор лучших результатов, оценено их качество	

Руководитель _____ В.Е. Гай
(подпись) (И.О. Фамилия)

Задание принял к исполнению _____
(дата)

Студент _____ В.С. Новичков
(подпись) (И.О. Фамилия)

Примечания:
1. Это задание прилагается к законченной работе и в составе пояснительной записки предоставляется в ГЭК.
2. До начала консультаций студент должен составить и утвердить у руководителя календарный график работы на весь период выполнения ВКР (с указанием сроков выполнения и трудоемкости отдельных этапов).

Оглавление

Введение

В последнее время всё больше и больше компаний продолжают присоединяться к разработке полностью автономно управляемых транспортных средств. Сравнительно недавний доклад свидетельствует, что к 2030 году технология автономного вождения станет обширной индустрией стоимостью 87 миллиарда долларов, другие исследования говорят о 7 триллионах долларов к 2050 году. Это не удивительно, что многие компании сейчас активно инвестируют в создание подобных технологий.

Тем не менее, много проблем и препятствий нужно рассмотреть и решить до того, как полностью автономно управляемые автомобили смогут продаваться и использоваться на общественных улицах. Пожалуй, наиболее фундаментальная задача — это дать автономно управляемым автомобилям способность водить действительно аккуратно, безопасно и в соответствии со всеми законами.

Одним из конкретных препятствий для транспортного средства будет распознавать дорожные знаки так же, как это может делать человек. Например, обнаруживать знаки остановки, светофоры, знаки ограничений скорости, предупреждающие знаки и так далее.

В центре внимания данного исследования будет улучшить существующие методы распознавания за счёт повышения точности (меньше ложных срабатываний, меньше пропущенных обнаружений) и сокращения времени анализа изображения.

Эта работа будет попыткой разработать надёжный детектор знаков дорожного движения путём построения, обучения и настройки различных свёрточных нейронных сетей.

1. Техническое задание

1.1. Назначение разработки и область применения

Разрабатываемая система предназначена для распознавания знаков дорожного движения при помощи использования свёрточных нейронных сетей. Для работы с данной системой должна существовать передающая сторона и принимающая сторона.

Области применения разрабатываемой системы:

- Автономно управляемые транспортные средства;

- Система помощи водителю;

- Создание дорожных карт.

1.2. Технические требования

Рассмотрим требования, предъявляемые разрабатываемой системой к ЭВМ:

- Операционная система с графическим интерфейсом;

- Требования к аппаратному обеспечению определяются операционной системой;

- Мышь, дисплей, Wi-Fi модуль.

Разрабатываемая система распознавания знаков дорожного движения должна обладать следующим функционалом:

Передающая сторона:

- Передача изображения со встроенной камеры;

- Обладает Wi-Fi модулем.

Принимающая сторона:

- Система должна предоставить возможность выбирать изображение, на котором будет искать объект;

- Позволяет указать обучающую выборку или выбрать уже обученную модель;

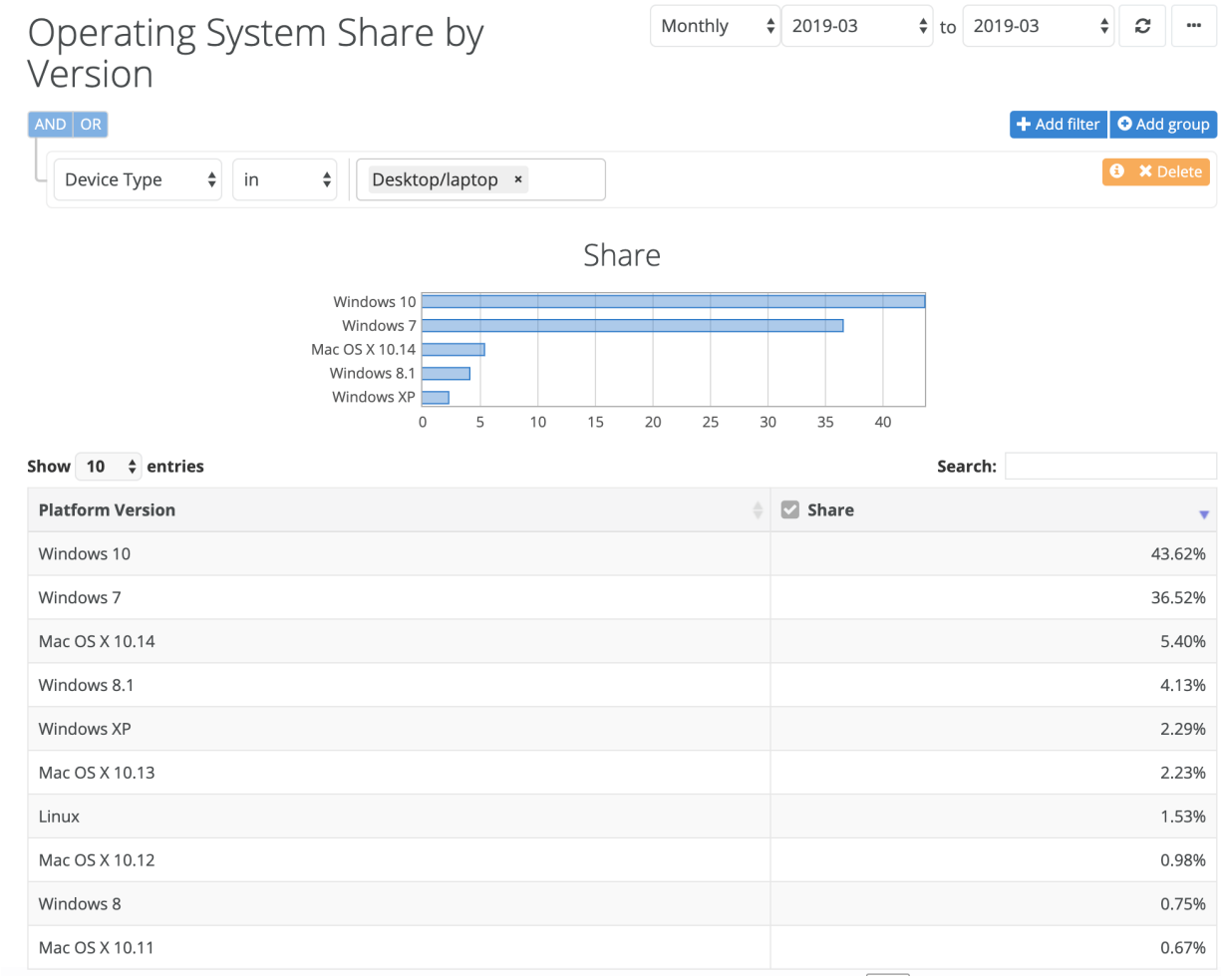
- Реализовывает поиск и распознавание дорожного знака;

- Выводит результирующее изображение.

2. Анализ поставленной задачи

2.1. Выбор операционной системы для разработки

На этапе начала разработки важным является выбор операционной системы, так как каждая из них имеет свои особенности, влияющие на разработку. Как показывает статистика (рис.1), наиболее распространёнными операционными системами являются Windows, Linux и Mac OS, их мы и рассмотрим.



Platform Version	Share
Windows 10	43.62%
Windows 7	36.52%
Mac OS X 10.14	5.40%
Windows 8.1	4.13%
Windows XP	2.29%
Mac OS X 10.13	2.23%
Linux	1.53%
Mac OS X 10.12	0.98%
Windows 8	0.75%
Mac OS X 10.11	0.67%

Рисунок 1 – Распространённость операционных систем

Windows. Самая распространенная операционная система. По данным статистики, она установлена на 85% устройств: планшеты, ноутбуки, компьютеры. Используются как дома, так и на предприятиях. Для данной операционной системы имеется огромное количество различного программного обеспечения.

Самые главные плюсы – отличная совместимость и распространенность.

Linux. Общее название для всех Unix-подобных систем. Существует множество дистрибутивов на основе ядра Linux Kernel. Отличаются они включёнными в себя компонентами. Более 80% серверов в Интернете работают на базе одного из дистрибутивов Linux, FreeBSD или другой Unix-подобной системы. Ядро Linux создается и распространяется в соответствии с моделью разработки свободного и открытого программного обеспечения. Это является главной особенностью данной операционной системы и делает её очень привлекательной для разработчиков.

Достоинство – оперативная доработка недочетов и неточностей благодаря открытому исходному коду.

Mac OS. Система, которая была разработана компанией Apple на основе Unix. Это сопутствующее ПО для выпускаемых этой корпорацией устройств. Является второй наиболее часто используемой операционной системой. Исходный код закрыт. В настоящий момент занимает менее 20% рынка и считается второй по популярности.

Преимущества – стабильность и производительность.

Для разработки системы распознавания знаков дорожного движения мной была выбрана ОС Windows, так как она отвечает всем предъявляемым требованиям и имеется ранее полученный опыт разработки на данной платформе.

2.2. Выбор языка программирования

Python. Один из наиболее часто используемых языков программирования сегодня. У него высокая удобочитаемость, поэтому он хорош для обучения. Это бесплатный язык программирования с открытым исходным кодом, обширной поддержкой модулей и большим сообществом разработчиков. Можно выделить лёгкую интеграцию с веб сервисами, интуитивно понятные для пользователя структуры данных. Язык очень популярен для реализации машинного обучения и искусственного интеллекта. Python может использоваться для разработки пакетов для таких графических программ как Blender, Inkscape и Autodesk. Популярен и для технического изучения, и для коммерческого использования.

Достоинства – гибкость, интуитивная читаемость, количество документации, динамическая типизация.

C. Вероятно, старейший из часто используемых языков программирования. C++ является расширенной версией C. Многие разработчики в наши дни пропускают обучение C, другие же считают, что его изучение даёт значительный фундамент для дальнейшего изучения. Оба языка широко используются в компьютерной науке и программировании. Разработчики могут использовать компиляторы для большого разнообразия платформ, делая приложения хорошо портируемыми. Оба считаются высокопроизводительными языками. Таким образом, они широко используются в разработке приложений, где важна производительность.

Преимущества: хорошая фундаментальная база для дальнейшего изучения, более высокий уровень контроля, чем в других языках.

C#. Изначально, Microsoft собиралась выпустить свою версию языка Java, однако им пришлось судиться с правообладателями из-за некоторых спорных моментов. Поэтому руководство приняло решение о необходимости создания собственного языка, который бы отвечал их требованиям и развитие которого они могли бы контролировать. Так и появился C#. Обучение на C# не представляет большой сложности. Особенно для тех, кто знаком с синтаксисом из C или C++. В любом случае, это не сложнее, чем Java.

Плюсы: средний порог вхождения, много конструкций, созданных для облегчения написания и понимания кода, большое сообщество.

Java**.** Это язык программирования общего назначения, который следует парадигме объектно-ориентированного программирования и подходу «Написать один раз и использовать везде». Java используется для настольных, сетевых, мобильных и корпоративных приложений. Это не только язык программирования, но и экосистема инструментов, охватывающая почти все, что может понадобиться при программировании на Java. В нее входят:

Java Development Kit (JDK)—комплект разработчика Java. С помощью JDK и стандартного блокнота можно писать и запускать/ компилировать код на Java;

Java Runtime Environment (JRE)—исполняющая система Java. Механизм распространения программного обеспечения, состоит из автономной виртуальной машины Java, стандартной библиотеки Java (Java Class Library) и инструментов настройки.

Integrated Development Environment (IDE)—интегрированная среда разработки. Инструменты, которые помогают запускать, редактировать и компилировать код. Самые популярные из них—IntelliJ IDEA, Eclipse и NetBeans.

Достоинства: язык высокого уровня с простым синтаксисом и плавной кривой обучения, автоматическое управление памятью.

Для создания системы был выбран язык Python. Он проще для обучения, чем C и C++, а написанный код выглядит намного более лаконично, чем на Java.

Решающим факторам выбора языка стало наличие библиотек (пакетов) для работы с нейронными сетями и изображением, которые впоследствии понадобятся при разработке и тестировании системы распознавания знаков дорожного движения.

2.3. Выбор среды разработки

Eclipse + PyDev.

Eclipse де-факто является open-source IDE для разработки на Java. Существует множество расширений и аддонов, которые делают Eclipse полезным для разного рода задач. Одним из таких расширений является PyDev, предоставляющий интерактивную консоль Python и возможности для отладки и авто дополнения кода.

Sublime Text.

Sublime Text, написанный инженером из Google с мечтой о лучшем текстовом редакторе, является весьма популярным редактором кода. Доступный на всех платформах, Sublime Text имеет встроенную поддержку редактирования Python-кода, а также богатый набор расширений, называемых пакетами, которые расширяют возможности синтаксиса и редактирования.

Установить дополнительный Python-пакет может быть непросто - все пакеты Sublime Text написаны на Python, поэтому для установки пакетов сообщества зачастую может потребоваться выполнить Python-скрипт непосредственно в редакторе.

Visual Studio.

Python-расширение: Python Tools for Visual Studio (PTVS).

Visual Studio — полнофункциональная IDE от Microsoft, которая во многом сопоставима с Eclipse. Доступная на Windows и Mac OS, Visual Studio представлена как в бесплатном (Community), так и в платном (Professional и Enterprise) вариантах. Visual Studio позволяет разрабатывать приложения для разных платформ и предоставляет свой собственный набор расширений.

Преимущества: как и в случае с Eclipse, если у вас уже установлена Visual Studio для других задач, установка PTVS пройдёт без проблем.

Недостатки: как и в случае с Eclipse, Visual Studio будет многовато, если вам нужен только Python.

Thonny.

Thonny называют IDE для новичков. Написанный и поддерживаемый Институтом информатики Тартуского университета в Эстонии, Thonny доступен на всех основных платформах. По умолчанию Thonny устанавливается с версией Python, идущей в комплекте.

Spyder.

Spyder — open-source IDE для Python, оптимизированная для data science. Spyder идёт в комплекте с менеджером пакетов Anaconda, поэтому вполне возможно, что он у вас уже установлен. Что в Spyder интересно, так это то, что его целевой аудиторией являются data scientist'ы, использующие Python. Например, Spyder хорошо взаимодействует с такими библиотеками для data science, как SciPy, NumPy и Matplotlib.

Spyder обладает той функциональностью, которую вы можете ожидать от стандартной IDE, вроде редактора кода с подсветкой синтаксиса, автодополнения кода и даже встроенного обозревателя документации. Отличительной особенностью Spyder является наличие проводника переменных. Он позволяет просмотреть значения переменных в форме таблицы прямо внутри IDE. Также хорошо работает интеграция с IPython/Jupyter.

Про Spyder можно сказать, что он более «приземлённый», чем другие IDE. Его можно рассматривать как инструмент для определённой цели, а не как основную среду разработки. Что в нём хорошо, так это, что он бесплатный, open-source и доступный на Windows, macOS и Linux.

PyCharm.

Одной из лучших полнофункциональных IDE, предназначенных именно для Python, является PyCharm. Существует как бесплатный open-source (Community), так и платный (Professional) вариант IDE. PyCharm доступен на Windows, Mac OS X и Linux. Кроме того, в IDE есть поддержка проектов и системы управления версиями.

Это среда разработки для Python с поддержкой всего, и вся с хорошим сообществом. В ней «из коробки» можно редактировать, запускать и отлаживать Python-код. PyCharm может медленно загружаться, а настройки по умолчанию, возможно, придётся подкорректировать для существующих проектов.

В итоге, мной была выбрана среда разработки PyCharm, как наиболее знакомая.

2.4. Подбор необходимых пакетов

OpenCV.

OpenCV (англ. Open Source Computer Vision Library, библиотека компьютерного зрения с открытым исходным кодом) — библиотека алгоритмов компьютерного зрения, обработки изображений и численных алгоритмов общего назначения с открытым кодом. Реализована на C/C++, также разрабатывается для Python, Java, Ruby, Matlab, Lua и других языков. Может свободно использоваться в академических и коммерческих целях — распространяется в условиях лицензии BSD.

TensorFlow.

TensorFlow — открытая программная библиотека для машинного обучения, разработанная компанией Google для решения задач построения и тренировки нейронной сети с целью автоматического нахождения и классификации образов, достигая качества человеческого восприятия. Применяется как для исследований, так и для разработки собственных продуктов Google. Основной API для работы с библиотекой реализован для Python, также существуют реализации для C++, Haskell, Java, Go и Swift.

NumPy.

NumPy — это библиотека с открытым исходным кодом, добавляющая поддержку больших многомерных массивов и матриц, вместе с большой библиотекой высокоуровневых (и очень быстрых) математических функций для операций с этими массивами.

Tkinter.

Tkinter — это графическая библиотека, позволяющая создавать программы с оконным интерфейсом. Получаемые изображения могут быть использованы в качестве иллюстраций в публикациях.

2.5. Обзор подходов к решению задачи распознавания и классификации

Процесс распознавания дорожных знаков разделяется на два основных этапа, и первым из них является локализация. На этапе локализации дорожных знаков используются методы, основанные на особенностях цвета и формы дорожных знаков. Данные детекторы определяют и сегментируют области входного кадра видеопоследовательности, цвета которых входят в заранее заданный диапазон. Основной проблемой подобных методов является вариативность результатов поиска искомых областей кадра к наличию различных помех, вызванных изменением освещенности, воздействия погодных условий и т. д. Для решения данной проблемы исследователи используют различные цветовые пространства, такие как:

- RGB - используется нормализованный фиксированный диапазон цвета;

Это самая распространенная модель представления цвета. В ней любой цвет рассматривается как оттенки трех основных (или базовых) цветов: красный (Red), зеленый (Green) и синий (Blue). При этом существует два вида этой модели: восьмибитное представление, где цвет задается числами от 0 до 255 (например, цвет [0,0,255] будет соответствовать синему, а [255, 255, 0] - желтому), и шестнадцатибитное, которое чаще всего используется в графических редакторах и html, где цвет задается числами от 0 до ff (зеленый - #00ff00, синий - #0000ff, желтый - #ffff00).

Разница представлений в том, что в восьмибитном виде для каждого базового цвета используется отдельная шкала, а в шестнадцатибитном уже сразу вводится цвет. Иными словами, восьмибитное представление - три шкалы с каждым основным цветов, шестнадцатибитное - одна шкала с тремя цветами.

Особенность этой модели в том, что здесь новый цвет получается путем добавления оттенков основных цветов, т.е. "смешивания".

Цвета смешиваются друг с другом, образуя новые цвета (желтый - [255,255,0], пурпурный - [255,0,255], голубой - [0,255,255] и белый [255,255,255]).

При этом эта модель чаще всего используется именно в численном виде, а не в визуальном (когда цвет задается вводом его значения в соотв. поля). Для визуальной настройки цвета используются другие модели.

- HSV - используется для получения информации с кадра, с меньшим воздействием погодных условий и изменением освещенности;

В цветовом пространстве модели HSV (Hue, Saturation, Value), иногда называемой HSB (Hue, Saturation, Brightness), используется цилиндрическая система координат, а множество допустимых цветов представляет собой шестигранный конус, поставленный на вершину.

Основание конуса представляет яркие цвета и соответствует V = 1. Однако цвета основания V = 1 не имеют одинаковой воспринимаемой интенсивности. Тон (H) измеряется углом, отсчитываемым вокруг вертикальной оси OV. При этом красному цвету соответствует угол 0°, зелёному – угол 120° и так далее до 360. Цвета, взаимно дополняющие друг друга до белого, находятся напротив один другого, т. е. их тона отличаются на 180°. Величина S изменяется от 0 на оси OV до 1 на гранях конуса.

Конус имеет единичную высоту (V = 1) и основание, расположенное в начале координат. В основании конуса величины H и S смысла не имеют. Белому цвету соответствует пара S = 1, V = 1. Ось OV (S = 0) соответствует ахроматическим цветам (серым тонам).

Для решения задачи локализации дорожных знаков на входном кадре с помощью детекторов геометрических признаков, используют такие методы, как:

- преобразование Хафа;

Классическое преобразование Хафа было первоначально разработано для выделения на бинарном изображении не кругов, а прямых линий. Оно основывается на использовании пространства параметров, в котором и производится поиск прямых. Наиболее распространены следующие параметрические уравнения прямых:

$Y=kX+b$;

$X\cos\theta+Y\sin\theta=\rho$.

Преобразование Хафа позволяет находить на монохромном изображении плоские кривые, заданные параметрически, например, прямые, окружности, эллипсы, и т.д. Монохромным изображением считается изображение, состоящее из точек двух типов: фоновых точек и точек интереса. Задача преобразования Хафа состоит в выделении кривых, образованных точками интереса.

Идея преобразования Хафа состоит в поиске кривых, которые проходят через достаточное количество точек интереса. Параметры семейства кривых образуют фазовое пространство, каждая точка которого соответствует некоторой кривой. Ввиду дискретности машинного представления и входных данных (изображения), требуется перевести непрерывное фазовое пространство в дискретное. Для этого в фазовом пространстве вводится сетка, разбивающая его на ячейки, каждая из которых соответствует набору кривых с близкими значениями параметров. Каждой ячейке фазового пространства можно поставить в соответствие число (счётчик), указывающее количество точек интереса на изображении, принадлежащих хотя бы одной из кривых, соответствующих данной ячейке. Анализ счётчиков ячеек позволяет найти на изображении кривые, на которых лежит наиболее количество точек интереса.

- построение карты расстояний;

Карта расстояний (Distance Map) — это объект, позволяющий быстро получить расстояние от заданной точки до определенной поверхности. Обычно представляет собой матрицу значений расстояний для узлов с фиксированным шагом. Часто используется в играх для определения «попадания» в игрока или предмет, и для оптимизационных задач по совмещению объектов: расположить объекты максимально близко друг к другу, но так, чтобы они не пересекались.

- построение гистограммы направленных градиентов.

Гистограмма направленных градиентов (HOG) используется достаточно распространена в сфере компьютерного зрения. Метод был впервые предложен в 2005 году для обнаружения пешеходов в статических изображениях. В последствии он был расширен для обнаружения людей на видео, также, как и для обнаружения различных животных и объектов.

Перед стабильной работой, классифицирующий алгоритм нужно натренировать, показывая тысячи изображений на которых есть интересующий нас предмет и на которых его нет. Каждый алгоритм обучения учится по-разному, но главный принцип заключается в том, что они рассматривают характерические векторы как точки в пространстве большей размерности и пытаются найти плоскости/поверхности, которые разбивают это пространство таким образом, что все эти объекты, принадлежащие к одному классу, находятся на одной стороны плоскости/поверхности.

Основным недостатком по сравнению с методами, основанными на особенностях цвета, является высокая вычислительная сложность алгоритмов.

Также для локализации дорожных знаков используются максимально стабильные области экстремума, при которых исходное изображение обрабатывается пороговой функцией с изменяющимся значением порога. В результате получается новая последовательность изображений, размер которой соответствует количеству различных значений порога (например, для монохромного изображения со значениями пикселей от 0 до 255 получим 256 изображений). Первое изображение в последовательности будет абсолютно белым. Дальше появятся черные области и самое последнее изображение в последовательности будет полностью черным. Данный метод является устойчивым к изменению контраста и световых условий.

Работа представленных детекторов может быть скомбинирована для увеличения точности локализации и сегментирования дорожных знаков.

После получения локализованного дорожного знака требуется его классифицировать. При решении задач классификации необходимо отнести имеющиеся статические образцы к определенным классам. Возможно несколько способов представления данных.

Наиболее распространенным является способ, при котором образец представляется вектором. Компоненты этого вектора представляют собой различные характеристики образца, которые влияют на принятие решения о том, к какому классу можно отнести данный образец. Например, для медицинских задач в качестве компонентов этого вектора могут быть данные из медицинской карты больного. Таким образом, на основании некоторой информации о примере, необходимо определить, к какому классу его можно отнести.

Классификатор относит объект к одному из классов в соответствии с определенным разбиением N-мерного пространства, которое называется пространством входов, и размерность этого пространства является количеством компонент вектора.

Для решения данной задачи используются методы, разделяющиеся на две категории:

- шаблонные методы;

Поиск объектов на основании некоторого шаблона предполагает, что имеется изображение объекта с выделенными признаками – шаблон – и тестовое изображение, которое сопоставляется с этим шаблоном. В простейшем случае в качестве шаблона может выступать матрица интенсивностей цветов, наиболее характерных для объекта. Более сложные методы рассматриваемой группы в качестве шаблона используют наборы векторов признаков (дескрипторы), геометрическое представление объекта или вероятностные модели объектов, которые содержат информацию о распределениях интенсивностей пикселей.

Сопоставление с шаблоном подразумевает сравнение описаний тестового и шаблонного изображений по некоторой выбранной метрике, как правило, выбирается евклидово расстояние, норма L1, взвешенная свертка квадратичных ошибок либо корреляция. Отметим, что методы поиска по заданному шаблону эффективно работают при поиске одиночных объектов, т.к. при возникновении перекрытий исчезают некоторые признаки в описании.

Шаблонные методы выполняют попиксельно сравнение между сегментированной областью и заранее созданным шаблоном. Данная техника достаточно проста и дает высокие показатели точности и быстродействия при классификации на предварительно выровненных изображениях. Но для изображений, снятых под определенным углом потребуется создать шаблоны под различными углами наклона и поворота, либо же в процессе сравнения поворачивать и масштабировать сравниваемый шаблон в соответствии с различными ориентирами (направление прямых линий и т. п). Такие методы называются деформируемыми моделями.

- классификаторы на основе нейронных сетей.

Классификаторы основываются на технологиях машинного обучения, что позволяет при должном обучении получать приемлемые результаты классификации вне зависимости от каких-либо внешних помех. Основными алгоритмам в этой категории являются:

- метод опорных векторов;

SVM один из наиболее популярных алгоритмов двоичной классификации. Для удобства возьмём характеристический вектор размерностью 2. Точки на изображении 2 представляют два класса (присутствие объекта и его отсутствие). На изображении, расположенном ниже, два класса представлены двумя типами точек. В процессе тренировки алгоритму предоставляется много примеров изображений двух классов. Другими словами, алгоритм собирает информацию о координатах точек, а также, являются точки черными или белыми.

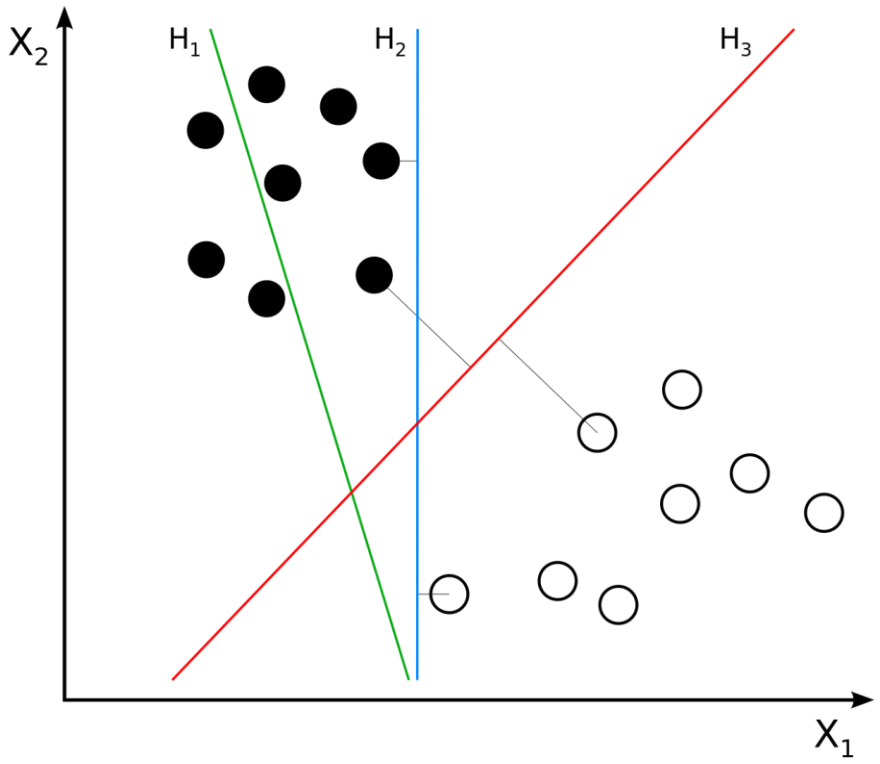


Рисунок 2 – Метод опорных векторов.

SVM пытается найти наиболее подходящую линию, которая разделяет два класса. На рисунке есть три линии H1, H2 и H3. H1 не разделяет два класса и не является хорошим классификатором. H2 и H3 разделяют два класса, но H3 делает это лучшим образом. Таким образом, в процессе тренировки SVM найдет линию H3. Если вектор является трехмерным, SVM построит соответствующую плоскость, которая максимально разделяет два класса. Для дескриптора HOG будет построена гиперплоскость.

- нейронные сети;

Нейронная сеть представляет собой последовательность связанных нейронов. Нейроны — единицы, получающие и передающие информацию. Сами по себе они не играют важной роли: нейроны имеют значение только в выстроенной из них цепи. К нейрону поступают входящие сигналы, каждому из которых присвоен определенный вес. Сигнал умножается на свой вес, значения суммируются, и получается единое число, которое получает активационная функция.

Простая нейронная сеть может состоять из трех уровней и передавать данные только вперед. Она включает в себя входящие нейроны, скрытый (промежуточный) слой нейронов, недоступный внешнему обозревателю, и нейрон на выходе. (рис.3)

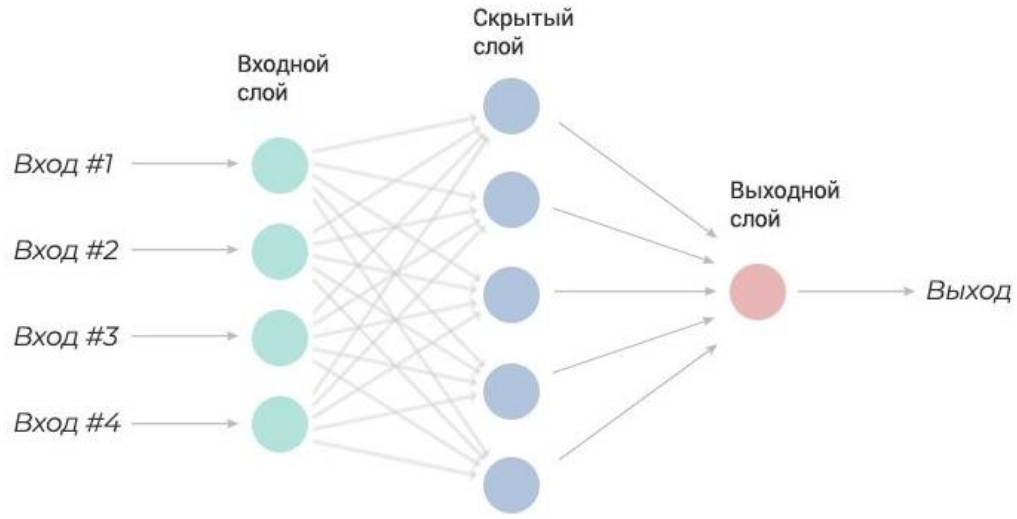


Рисунок 3 – Компоненты нейронной сети.

Добавляя скрытые слои, можно делать усложнения и обобщать: это то, что свойственно человеческому мозгу и считалось несвойственным обычному алгоритму.

Обобщение для нейросети — это способность учитывать сочетание условий, так как некоторые пары условий приводят к новому качественному показателю. Это происходит, когда в разных комбинациях условия имеют различный вес. Чтобы принимать решения в таких ситуациях, и используют скрытый слой нейронов.

Самая сложная задача в работе с нейросетью — грамотно подобрать коэффициенты к нейронам. Для этого используется обучение — процесс нахождения корректных весов для нейросети. От того, как именно обучат нейросеть, будут зависеть ее решения.

- регрессионные деревья решений;

На первой итерации мы строим все возможные (в дискретном смысле) гиперплоскости, которые разбивали бы наше пространство на два. Для каждого такого разбиения пространства считается количество наблюдений в каждом из подпространств разных классов. В результате выбирается такое разбиение, которое максимально выделило в одном из подпространств наблюдения одного из классов. Соответственно, это разбиение будет нашим корнем дерева принятия решений, а листами на данной итерации будет два разбиения.

На следующих итерациях мы берем один худший (в смысле отношения количества наблюдений разных классов) лист и проводим ту же операцию по разбиению его. В результате этот лист становится узлом с каким-то разбиением, и двумя листами.

Продолжаем так делать, пока не достигнем ограничения по количеству узлов, либо от одной итерации к другой перестанет улучшаться общая ошибка (количество неправильно классифицированных наблюдений всем деревом). Однако, полученное дерево будет “переобучено” (будет подогнано под обучающую выборку) и, соответственно, не будет давать нормальные результаты на других данных. Для того, чтобы избежать “переобучения”, используют тестовые выборки (либо кросс-валидацию) и, соответственно, проводится обратный анализ, когда дерево уменьшают в зависимости от результата на тестовой выборке.

Относительно простой алгоритм, в результате которого получается одно дерево принятия решений. За счет этого, он удобен для первичного анализа данных, к примеру, чтобы проверить на наличие связей между переменными и другим.

Основным недостатком данных методов является требование правильно выбирать модель обучения, после чего проводить длительное обучение классификатора.

Подводя итоги, можно сделать предположение, что наиболее эффективным будет использовать в качестве локализаторов цветовой детектор вместе с одним из геометрических детекторов, предварительно выбрать цветовое пространство RGB. Для классификатора лучше всего подойдет классификатор на основе нейронных сетей, так как вычислительные мощности на распознавание малы, что приемлемо для устройств, не обладающих высокими техническими характеристиками.

3. Разработка системы на структурном уровне

3.1. Архитектура разрабатываемой системы

Разрабатываемую систему обмена сообщениями можно представить в виде передающей и принимающей стороны, которые включают в себя основные модули работы системы.

Реализация распознавания дорожных знаков состоит из двух частей:

- Создание классификатора и обучение модели.
- Классификация входных данных на основе обученной модели.

В процесс обучения классификатора и создание модели входят несколько этапов:

- Предварительный поиск базы изображений, изначально классифицированных как позитивные (должны распознаваться) и негативные (не должны).
- Систематизация данных в наборы (тренировочный и тестовый).
- Сборка и тестирование модели.
- Оценка точности модели.

Классификация входных данных подразумевает в себе:

- Получение входных данных – получение видеоряда из источника.
- Формирование системы признаков – формирование набора признаков, на основании которых будет принято решение
- Принятие решения – идентификация дорожного знака.

Таким образом, можно сформировать систему распознавания дорожных знаков, состоящую из следующих компонентов: база данных изображений, обученная модель, модуль получения изображения, модуль предварительной обработки изображения, модуль принятия решения (рис.4).

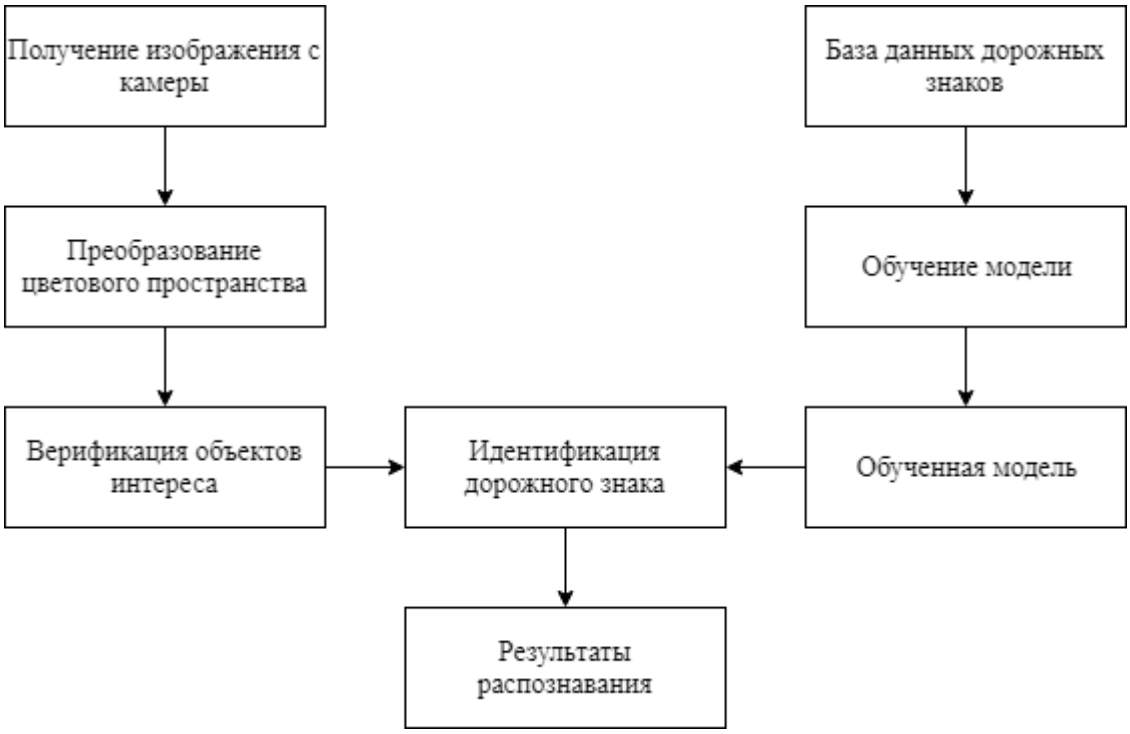


Рисунок 4 – Архитектура системы распознавания дорожных знаков.

3.2. Передающая сторона

В нашем случае в качестве имитации автомобиля было решено использовать танк на радиоуправлении Brookstone Rover 2.0 (рис.5). Для простейшего тестирования может быть использована обычная USB веб-камера, подключаемая к основной системе.

Характеристики:

Регулируемая камера с широким углом обзора с инфракрасным ночным видением.

Wi-Fi модуль, по которому идёт передача сигнала на расстоянии до 60 метров.

Рисунок 5 – Brookstone Rover 2.0.

3.3. Принимающая сторона

В качестве принимающей стороны выступает персональный компьютер. На нём будет проходить обучение модели, принятие и обработка изображения, а также его классификация и вывод решения.

От характеристик компьютера зависит скорость принятия решения и обучения модели на заданном датасете. В нашем случае обучение модели происходило на видеокарте (GPU) и заняло 3 часа. Процессор (CPU) потратил бы намного больше времени.

Так как запуск происходит из среды разработки, то характеристики персонального компьютера должны соответствовать её требованиям.

Системные требования для PyCharm:

64-bit versions of Microsoft Windows 10, 8, 7 (SP1);

4 GB RAM minimum, 8 GB RAM recommended;

1.5 GB hard disk space + at least 1 GB for caches;

1024x768 minimum screen resolution;

Python 2: versions 2.6 and 2.7 or Python 3: from the version 3.4 up to the version 3.7.

3.4. Свёрточные нейронные сети

CNN показали себя невероятно мощным инструментов в области компьютерного зрения для классификации изображений и обнаружения объектов.

Выбор именно CNN не так тривиален. Есть много альтернатив, которые можно рассмотреть, такие как глубокие сети доверия (deep belief network), глубокие нейронные сети (deep neural network), машина Больцмана (Boltzmann machine), рекуррентные нейронные сети и так далее. Тем не менее, CNN, как правило, является наиболее распространённым выбором для задач компьютерного зрения, таких как распознавание образов.

CNN, свёрточные нейронные сети, представляют собой тип искусственной нейронной сети. Искусственные нейронные сети (Artificial neural network) это математическая модель, построенная по принципу организации и функционирования биологических нейронных сетей. По своей сути ANN представляет собой сеть узлов (нейронов), где каждый узел соединён с некоторыми другими узлами, и всем этим узлам присваиваются веса на основе важности этих соединений (так же, как и в реальном мозге). Все эти веса оптимизируются через обучение на обучающем наборе. Кроме того, в процессе обучения, валидационный набор используется для оптимизации и изучения новых параметров, а также позволяет судить о том, как хорошо вы оптимизировали весы с помощью обучающих данных. После использования, обучающего и валидационного набора для построения весов мы можем оценить качество распознавания. Для дальнейшего изменения модели должны использовать только результаты, полученные после проверки на тестовом наборе. CNN это ANN, которая состоит из нескольких типов слоёв, они включают в себя:

- Convolution layers (CONV). Это слои, от которого CNN и получили своё имя. По сути, это процесс, в котором входные данные «сворачиваются» с набором фильтров, а в результаты передаются в качестве входных данных в следующий слой. Фильтр можно рассматривать как скользящее окно с весами (матрица чисел), которое скользит по всему изображению (большей матрице чисел), и в каждой позиции оно берёт скалярное произведение с той частью изображения, что оно покрывает. Это будет частью выходных данных. Веса на этих фильтрах изучаются в процессе обучения.

Объект в CONV слое четырёхмерных: количество фильтров, количество каналов (входных матриц), высота фильтра и ширина фильтра. Как правило, для человеческого глаза очень трудно получить представление о том, что эти фильтры делают, особенно если это фильтры не из первого свёрточного слоя. Лучше не стоит пытаться их визуализировать в таком виде. Но вот небольшой тривиальный пример, чтобы создать некоторое базовое представление:

Одной интересной свёрткой является оператор Собеля, делает края на изображении сильно выделенными. Это особенно полезно в обнаружении краёв, и, следовательно, может быть использовано в обнаружении края дорог, дорожных знаков и многих других ситуациях.

Пример оператора Собеля:

SO =

-1	0	+1
-2	0	+2
-1	0	+1

Рисунок 6 – Ядро Собеля

Как можем увидеть, интуитивно, оператор Собеля должен вычислять очень большое скалярное произведение по абсолютной величине, когда он «свёрнут» в матрицу 3x3, можно видеть явное вертикальное разделение в центре этой матрицы. Так как в левой её стороне и в правой значения сильно отличаются. Если нужно обнаружить горизонтальные рёбра, то можно повернуть эту матрицу на 90 градусов. Можно также представить детектор для изогнутой или восьмиугольной форму, которые могут быть разработаны для обнаружения знаков. Но есть куда более полезные, но сложные операторы, которые хорошо работают, но их сложно представить так интуитивно.

- Down-sampling / pooling layers (POOL). Слои подвыборки в значительной степени ответственны за прореживание информации (изображение, когда оно только приходит), протекающей через CNN, что хорошо ускоряет вычисление и удаляет ненужные детали или шумы, однако если убрать слишком много, тогда может ухудшиться обнаружение объекта. Таким образом, необходимо найти правильный баланс. Суть подвыборки заключается в том, чтобы после того, как фильтр определил признак (получая более высокую точечную ценность), точное местоположение признака не так важно, а важно его расположение относительно других признаков

Таким образом, можно разделить текущую матрицу ввода на 2x2 или 3x3 плитки, взять максимальное или среднее значение из них и на выходе получить всего одно значение вместо этих четырёх или девяти. Пример можно увидеть на рисунке 7. Самым распространённым вариантом является размер пула 2x2 и нахождение его максимального значения, так как это наиболее хорошо работает на практике.

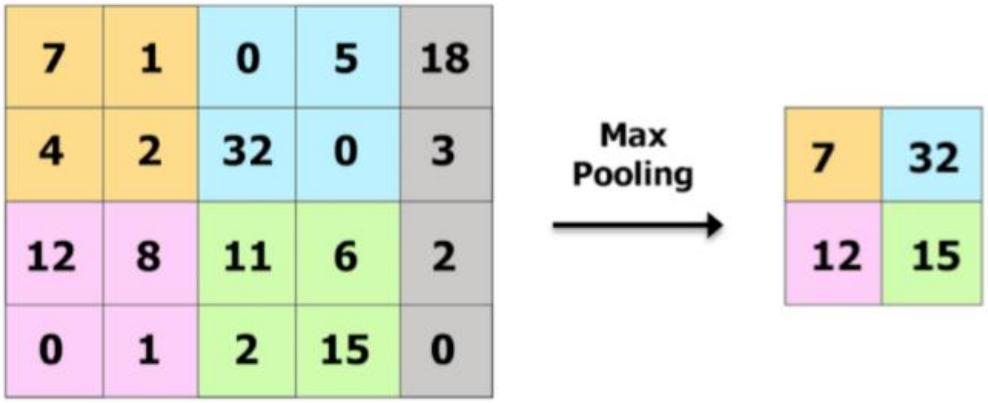


Рисунок 7 – Техника Макс-пула.

- Rectified Linear Unit Layers(ReLU). Слои активации отвечают за добавление дополнительной нелинейности к CNN, он должен действовать в качестве функции активации, чтобы позволить CNN обучаться быстрее. Существует спор о том, какую функцию использовать в этом слое, но используются три основные функции (рис. 8):

$$f(x) = \max(0,x) \qquad \text{range: } [0, \infty]$$

$$f(x) = \tanh(x) \qquad \text{range: } [-1, 1]$$

$$f(x) = (1 + e^{-x})^{-1} \qquad \text{range: } [0, 1]$$

Рисунок 8 – Три основные функции активации.

Первая функция часто является более предпочтительной, поскольку она является вычислительно простой, но имеет недостатки, связанные с большими значениями градиентного спуска. Вторая функция превосходит третью в том, что центрирована на нуль, что даёт дополнительные полезные свойства.

- Fully Connected Layer(FC). Как правило, это последний слой в CNN после некоторой комбинации любого количества упомянутых выше типов слоёв до него. Он уникален тем, что это единственный раздел CNN, где каждый узел в слое n имеет свои собственные связи для всех узлов в слое n+1 до тех пор, пока слои n и n+1 являются частью секции FC. Как можно понять, такое большое количество весов требует большое количество вычислительных мощностей. Так что комбинации CONV, POOL и ReLU слоёв до этого значительно уменьшают размер входных данных в FC слой.

Если FC слой имеет скрытые слои внутри себя, то он может рассматриваться как многослойный персептрон (MLP). В таком случае его обучение происходит с помощью метода обратного распространения ошибки в качестве стохастического градиентного спуска, чтобы минимизировать ошибку в результатах, которые являются выходом из конечного слоя FC.

Каждый слой в FC слое имеет весовую матрицу W и вектор смещения b, эти значения были получены после обучения. Последним слоем CNN является слой логической регрессии (LR), это самый распространённый метод для повышения производительности. Скрытые слои (HL) преобразуют входные данные в линейно разделённое пространство, а затем LR слой классифицирует полученные данные. Ниже приведена полезная визуализация FC слоя с тремя HL.

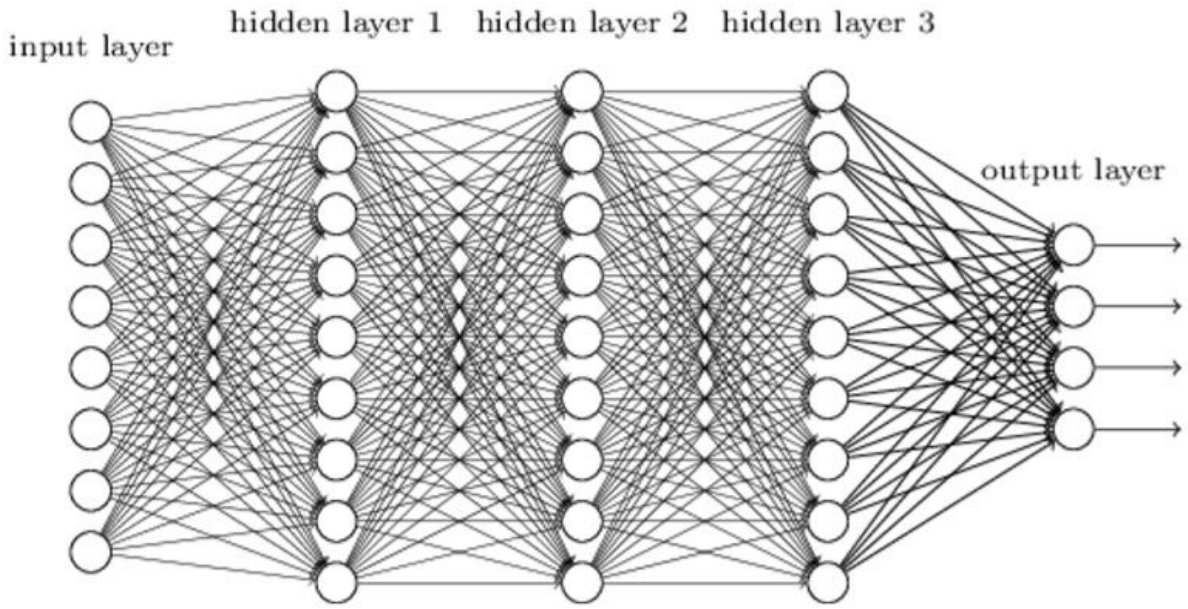


Рисунок 9 – Пример FC слоя.

3.5. Конфигурация сети

Для начала нужно определиться с топологией свёрточной сети (количество сверточных, подвыборочных, полносвязанных слоев). Наша нейронная сеть будет состоять из нескольких слоёв.

Первый из них – свёрточный. Для него нужно определить входные и выходные данные. Если размер будет слишком велик, то вычислительная сложность повысится, соответственно ограничения на скорость ответа будут нарушены, определение размера в данной задаче решается методом подбора. Если выбрать размер слишком маленький, то сеть не сможет выявить ключевые признаки лиц. Каждое изображение разбивается на 3 канала: красный, синий, зеленый. Таким образом получается 3 изображения размера 64x64 пикселей.

Входные: тип – изображение, размер – 64x64, формат – RGB.

Выходные: количество классов – 62.

Второй слой – подвыборки. Цель слоя – уменьшение размерности карт предыдущего слоя. Если на предыдущей операции свертки уже были выявлены некоторые признаки, то для дальнейшей обработки настолько подробное изображение уже не нужно, и оно уплотняется до менее подробного. К тому же фильтрация уже ненужных деталей помогает не переобучаться. Определим ядро размером 2x2.

Для большей производительности повторим эти слои ещё раз.

Следующий слой выравнивания. Он переводит полученный двумерный массив в вектор, подготавливая данные для последнего слоя.

Последний слой – полносвязанный. На этом этапе происходит классификация изображения, в зависимости от обнаруженных признаков.

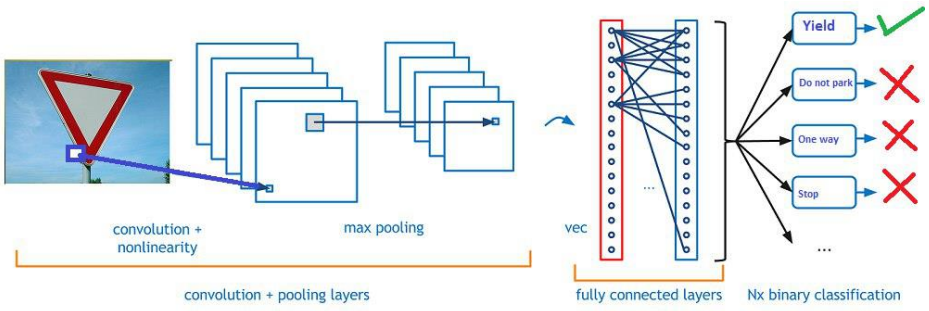


Рисунок 10 – Структура CNN.

4. Разработка системы на программном уровне

4.1. Поиск подходящего датасета

Для CNN требуется большое количество обучающих и валидационных изображений, чтобы получить оптимально настроенную модель с точки зрения производительности, впоследствии измеренной на валидационном и тестовом наборе. Реальный глобальный максимум, скорее всего, найти не представляется возможным и вычислительно непрактично пытаться получить его. Обычно ставится целью получить более подходящую модель, которая хорошо показывает себя на практике.

Важный момент дать нейросети как можно больше данных (изображений), но также они должны быть достаточно разнообразны для высокого качества обучения. Например, если подавать для обучения знак «стоп», и все изображения показывают его в одном метре от камеры на передней части автомобиля, знак занимает большую часть изображения, ярко освещён, без бликов или теней и снят под прямым углом, то обучения точного детектора для обнаружения такого знака будет являться гораздо более лёгкой задачей. Но тогда детектор будет также хорошо работать только в этой точной и простой ситуации. Если же изображения содержат знак под углом, частично затенённый, на разном расстоянии и тому подобные реалистичные ситуации, то детектору будет значительно сложнее распознать знак. И такой плохой детектор, очевидно, не может быть использоваться как центр автономной системы вождения, которая призвана держать водителя в безопасности.

Обучение CNN предполагает создание сетевой архитектуры путём настройки параметров (количество слоёв, типы слоёв, порядок слоёв, число ядер и так далее), а потом позволить оптимизатору обработать тысячи изображений в обучающем наборе, тысячи раз, чтобы оценить оптимальные значения для тысяч параметров в слоях с помощью метода обратного распространения ошибки (Основная идея этого метода состоит в распространении сигналов ошибки от выходов сети к её входам, в направлении, обратном прямому распространению сигналов в обычном режиме работы) и стохастическому градиентному спуску (При стохастическом градиентном спуске значение градиента аппроксимируются градиентом функции стоимости, вычисленном только на одном элементе обучения. Затем параметры изменяются пропорционально приближенному градиенту).

Количество параметров огромно и результат в значительной степени зависит от хорошо подобранной выборки для обучающего датасета. Важно так же отметить, что слово «тысячи» могут быть заменены на «десятки тысяч», «сотни тысяч», «миллионы» и так далее. Всё это зависит от многих факторов, таких как сложность задачи обнаружения, количество имеющихся данных для обучения, лёгкость создания новых данных и вычислительных мощностей (аппаратное обеспечение), доступных при исследовании.

- LISA-TS – Laboratory for Intelligent & Safe Automobiles – Traffic Sign

Этот датасет представляет собой набор видео и аннотированных кадров, содержащих дорожные знаки США. Он есть в двух вариантах, один только с фотографиями, а во втором добавлены и видеоролики. Несколько исходных видео были разбиты на дорожки. Они названы по типу дорожного знака, который они содержат, но могут содержать и другие знаки. Из каждой дорожки извлекается до 30 кадров, и всех знаки в этих кадрах помечаются позицией, типом и некоторыми дополнительными метаданными. Датасет содержит 47 типов дорожных знаков США, 7855 объектов на 6610 кадрах, где сами знаки занимают от 6x6 до 167x168 пикселей. Изображения собраны с использованием нескольких различных камер и варьируются от 640x480 до 1024x522 пикселей. Файл с аннотацией содержит тип знака и его местоположение на изображениях. Полный датасет находится в стадии разработке и в данный момент недоступен для загрузки.

http://cvrr.ucsd.edu/LISA/lisa-traffic-sign-dataset.html

- The German Traffic Sign Detection Benchmark

Изначально был создан для IJCNN 2013 (International Joint Conference on Neural Networks). Обучающий набор стал публично доступен 1 декабря 2012 года, а тестовый только 18 февраля 2013 года. Основной архив включает в себя 900 изображений 1360x800 пикселей в PPM (portable pixmap format) формате и разбит на 600 (обучающий) и 300 (тестирующий). Весит чуть меньше 200 Мб.

Авторы: Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing, Christian Igel.

http://benchmark.ini.rub.de/?section=gtsdb

- The German Traffic Sign Recognition Benchmark

Использовался на Международной объединённой конференции по нейронным сетям (IJCNN) 2011. Содержит более 40 классов, структурированных в архиве. Большая и реалистичная база данных, всего более 50000 изображений. В каждой папке хранятся изображения и файл с правильной классификацией формата CSV. Для каждого дорожного знака представлено более 30 изображений. Изображение содержит только один знак, размер варьируется от 15x15 до 250x250 пикселей. В CVS файле указаны: название соответствующего изображения, его высота и ширина, а также координаты левого верхнего и правого нижнего углов для локализации знаков (рис. 11). Для обучающего набора указан ещё и класс знака.

Авторы: Johannes Stallkamp, Marc Schlipsing, Jan Salmen, Christian Igel.

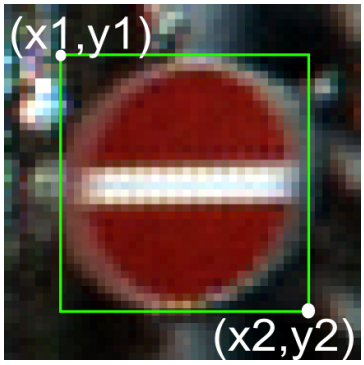


Рисунок 11 – Пример локализации.

http://benchmark.ini.rub.de/?section=gtsrb

- BelgiumTS Dataset

Поддерживается Radu Timofte. Состоит из нескольких наборов данных. Наиболее часто используется сокращённый набор из 62 дорожных знаков. Так же есть 8 наборов, снятых с 8 разных камер, 6 наборов, содержащих фоновые изображения для проверки на ложные отрицательные срабатывания, а также 48 наборов для тестирования. Суммарный их вес превышает 50 Гб, но есть два заранее разделённых на обучающих и тестирующих датасеты, содержащие только обрезанные изображения самих знаков дорожного движения. Их вес немного превышает 250 Мб.

https://btsd.ethz.ch/shareddata/

Так как в аннотации указаны только номера дорожных знаков, пришлось составить именную матрицу с помощью википедии:

https://wiki.openstreetmap.org/wiki/Road_signs_in_Belgium

4.2. Программная реализация модулей системы

Модуль управления танком. (Приложение 1)

Скрипт позволяет использовать весь функционал танка с персонального компьютера: управлять им (для правой и левой гусениц передаются отдельные коды сигналы), получать видеопоток, регулировать камеру, включать свет и микрофон.

Конкретно для наших целей используются эти части кода (рис. 12 и 13):

```
class Rover:
    def __init__(self):
        ''' Creates a Rover object that you can communicate with.'''

        self.HOST = '192.168.1.100'
        self.PORT = 80
        TARGET_ID = 'Rover_00E04C0DABA2'
        TARGET_PASSWORD = ''
        self.KEEPALIVE_PERIOD_SEC = 60

        # Create command socket connection to Rover
        self.commandsock = self._newSocket()
        # Send login request with four arbitrary numbers
        self._sendCommandIntRequest(0, [0, 0, 0, 0])
        # Get login reply
        reply = self._receiveCommandReply(82)
        # Send encrypted reply to Rover
        self._sendCommandIntRequest(2, [-14696588, 292529175, 1044086044, -1771855375])
        # Ignore reply from Rover
        self._receiveCommandReply(26)
        # Set up vertical camera controller
        self.cameraVertical = _RoverCamera(self, 1)
        # Send video-start request
        self._sendCommandIntRequest(4, [1])
        # Get reply from Rover
        reply = self._receiveCommandReply(29)
        # Create media socket connection to Rover
        self.mediasock = self._newSocket()
        # Send video-start request based on last four bytes of reply
        self._sendRequest(self.mediasock, 'V', 0, 4, map(ord, reply[25:]))
```

Рисунок 12 – Инициализация и подключение к танку.

```
def process_video_from_rover(self, jpegbytes, timestamp_10msec, lower_color=None, upper_color=None):
    array_of_bytes = np.fromstring(jpegbytes, np.uint8)
    self.image = cv2.imdecode(array_of_bytes, flags=-3)

    if self.image is None:
        print("[RoverExtended] self.image is empty.")
        return np.zeros((4, 3)), None
    else:
        self.image = cv2.resize(self.image, (640, 480))
```

Рисунок 13 – Получение видеопотока.

Модуль создания обученной модели. (Приложение 2)

Для классификации знаков дорожного движения необходимо создать обученную модель. Для этого нужно сконфигурировать нейронную сеть (рис.14) и подать ей набор данных по классам для обучения (рис.15).

```
# PREPARE A CONVOLUTIONAL NEURAL NETWORKS TAKES 64X64X3 (3 FOR RGB) AS INPUT
def prepare_classifier():
    # initialize
    classifier = Sequential()
    # STEP 1 - Convolution
    # 64,64,3 = 64X64 IMAGE DIMENSIONS, 3 FOR RGB
    # 32,3 = 32 SUB SAMPLES, 3X3 CONVOLUTON MATRIX DIMENSIONS, 1X1 STRIDE DIMENSIONS, PADDING IS VALID
    classifier.add(Convolution2D(32, 3, strides=(1, 1), input_shape=(64, 64, 3), padding='valid', activation='relu'))
    # STEP 2 - Pooling
    # 2,2 = POOL MATRIX DIMENSIONS
    classifier.add(MaxPooling2D(pool_size=(2, 2)))
    # STEP 3 - 2. Convolution
    # 64,64,3 = 64X64 IMAGE DIMENSIONS, 3 FOR RGB
    # 32,3 = 32 SUB SAMPLES, 3X3 CONVOLUTON MATRIX DIMENSIONS, 1X1 STRIDE DIMENSIONS, PADDING IS VALID
    classifier.add(Convolution2D(32, 3, strides=(1, 1), activation='relu', padding='valid'))
    # STEP 4 - Convolution
    # 2,2 = POOL MATRIX DIMENSIONS
    classifier.add(MaxPooling2D(pool_size=(2, 2)))
    # STEP 5 - Flattening
    # FLATTENING FOR THE FULLY CONNECTED LAYER
    classifier.add(Flatten())
    # STEP 6 - NEURAL NETWORK
    # 128 = HIDDEN LAYER BITS
    classifier.add(Dense(output_dim=128, activation='relu'))
    # 62 - OUTPUT BITS FOR 62 CLASSES. SOFTMAX IS SUGGESTED FOR OUTPUT
    classifier.add(Dense(output_dim=62, activation='softmax'))
    # CNN
    # OPTIMIZE ACCURACY WITH ADAM OPTIMIZER AND CATEGORICAL CROSS ENTROPY LOSS FUNCTION
    classifier.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    return classifier
```

Рисунок 14 – Реализация конфигурации свёрточной нейронной сети.

```
# OUR CURRENT MODEL
classifier = prepare_classifier()

# CNN AND IMAGES WITH IMAGE DATA GENERATOR WITH DATA AUGMENTATION FOR INCREASING THE SUCCESS
train_datagen = ImageDataGenerator(rescale=1./150, shear_range=0.2, zoom_range=0.2)
test_datagen = ImageDataGenerator(rescale=1./150, shear_range=0.2, zoom_range=0.2)
training_set = train_datagen.flow_from_directory("TrafficSigns/Training",
                                                 target_size=(64, 64), batch_size=8, class_mode='categorical')
test_set = test_datagen.flow_from_directory("TrafficSigns/Testing",
                                            target_size=(64, 64), batch_size=8, class_mode='categorical')
# 4552 = TRAINING IMAGES, 2520 = TEST IMAGES
classifier.fit_generator(training_set, samples_per_epoch=4552, nb_epoch=50,
                         validation_steps=2520, validation_data=test_set)
classifier.save('C:\\Users\\vnovichk\\PycharmProjects\\savedmodel\\my_model.h5')  # creates a HDF5 file 'my_model.h5'
# returns a compiled model
# identical to the previous one
# model = load_model('./savedmodel/my_model.h5')
```

Рисунок 15 – Реализация обучения свёрточной нейронной сети.

Модуль тестирования CNN. (приложение 2)

После того как была получена обученная модель на её основе происходит классификация знаков дорожного движения с подаваемого изображения. Тестирование проводится сразу после обучения модели для её оценки. Показательное тестирование отображает результат классификации 10 случайных изображений из тестового набора данных (рис.16), а статистическое тестирование классифицирует весь датасет (рит.16).

```
# PRINT 10 RANDOM IMAGES PREDICTIONS AND TRUE VALUES WITH THEIR IMAGE
print("----10 RANDOM TEST SAMPLES---")
# Display the predictions and the ground truth visually.
fig = plt.figure(figsize=(10, 10))
for i in range(len(sample_images)):
    truth = sample_labels[i]
    prediction = np.argmax(predicted[i])
    plt.subplot(5, 2, 1+i)
    plt.axis('off')
    color = 'green' if truth == prediction else 'red'
    plt.text(70, 20, "Truth:        {0}\nPrediction: {1}".format(truth, prediction),
             fontsize=12, color=color)
    plt.imshow(sample_images[i], cmap="gray")
```

Рисунок 16 – Реализация тестирования по 10 случайным изображениям.

```
# TEST WHOLE TESTING DATASET
array = classifier.predict(images28)

# COUNT THE TRUE PREDICTIONS
result = []
for i in range(0, len(array)):
    result.append(np.argmax(array[i]))
result2 = []
true = 0
for i in range(0, len(labelsTest)):
    if labelsTest[i] == int(result[i]):
        true = true+1
        result2.append(1)
    else:
        result2.append(0)

print("----ALL TEST SAMPLES---")
print("True predicted:"+str(true)+" / Total: "+str(len(labelsTest))+" = "+str(true/len(labelsTest)))

print("done")
```

Рисунок 17 – Реализация тестирования по всему тестовому датасету.

Модуль классификации изображения. (приложение 3)

Данный модуль позволяет, используя пользовательский интерфейс, передать модели исследуемое изображение, и на основе результата классификации выводит один из 62 изученных классов дорожных знаков.

```
#tkinter app
root = Tk()
root.geometry("550x300+300+150")
root.resizable(width=True, height=True)
model = load_model('C:\\Users\\vnovichk\\PycharmProjects\\TSR\\savemodel\\my_model.h5')


# open a image folder
def openfn():
    filename = filedialog.askopenfilename(title='open')
    return filename


#load image to screen and make prediction with saved CNN model
def open_img():
    x = openfn()
    img = Image.open(x)
    img = img.resize((64, 64), Image.ANTIALIAS)
    img = ImageTk.PhotoImage(img)
    panel = Label(root, image=img)
    panel.image = img
    panel.pack()
    # you must specify the path according to your own model path.

    testimage = load_img(x,target_size=(64,64))
    testimage = img_to_array(testimage)
    testimage = np.expand_dims(testimage, axis=0)
    prediction = model.predict(testimage)
    #print the prediction
    print(np.argmax(prediction[0]))
    var = StringVar()
    label = Label(root, textvariable=var, relief=RAISED )
    # give the output to screen
    var.set("This photo means: "+str(names[np.argmax(prediction[0])]))
    label.pack()


btn = Button(root, text='open image', command=open_img).pack()

root.mainloop()
```

Рисунок 18 – Реализация классификации дорожного знака по изображению.

5. Тестирование системы

5.1. Работоспособность программы

Для начала необходимо проверить работоспособность программы. Для этого сначала проверим обучение модели, а затем её работу на валидационном датасете.

Как видно на рисунке 19, обучение проходит успешно, а точность классификации на валидационном наборе данных составляет 99.58%.

```
567/569 [============================>.] - ETA: 0s - loss: 0.0164 - acc: 0.9958
568/569 [============================>.] - ETA: 0s - loss: 0.0164 - acc: 0.9958
569/569 [==============================] - 92s 162ms/step - loss: 0.0164 - acc: 0.9958 - val_loss: 0.3516 - val_acc: 0.9523
Epoch 49/50
```

Рисунок 19 – Обучение модели и валидация.

Далее проверим, как хорошо работает наш классификатор на выборке из 10 случайных изображениях из тестового набора данных. Как видим, все изображения классифицированы правильно (рис. 20).

Рисунок 20 – Тестирование на 10 изображениях.

5.2. Точность классификатора

Для тестирования системы на точность классификации были взяты все 2520 изображений из тестового набора данных.

```
----ALL TEST SAMPLES---
True predicted:2390 / Total: 2520 = 0.9484126984126984
done

Process finished with exit code 0
```

Рисунок 21 – Полное тестирование.

В ходе тестирования системы, сбоев в работе пользовательского интерфейса обнаружено не было, а итоговая точность классификатора составила 94.84%. Исходя из полученных результатов, можно сделать вывод, что разработанная система работает корректно.

Заключение

В результате выполнения выпускной квалификационной работы была спроектирована и программно реализована система распознавания знаков дорожного движения. Для этой цели были изучены и использованы свёрточные нейронные сети.

Созданная программная система предназначена для распознавания знаков дорожного движения. Она может быть использована в качестве системы помощи водителю. Тестирование системы подтвердило её работоспособность и возможность использования для решения поставленной задачи.

В дальнейшем можно будет собрать набор данных со знаками с российских дорог и переобучить модель на них.

Список использованных материалов

1. Ruta, A. «In-vehicle camera traffic sign detection and recognition» // A.Ruta, F.Porikli, S.Watanabe, Y.Li // Machine Vision and Applications. - 2011.
2. Брюхомицкий Ю.А. «Нейросетевые модели для систем информационной безопасности» // Таганрог: Изд-во ТРТУ - 2005.
3. В.Вьюгин. «Математические основы теории машинного обучения и прогнозирования». // МЦМНО – 2013.
4. ГОСТ Р 52290-2004 «Технические средства организации дорожного движения. Правила применения дорожных знаков, разметки, светофоров, дорожных ограждений и направляющих устройств».
5. Keras Documentation [Электронный ресурс]. https://keras.io/
6. Хабрахабр [Электронный ресурс]. https://habr.com
7. Википедия [Электронный ресурс]. https://www.wikipedia.org/

Приложение 1

```
import struct
import threading
import socket
import time
from blowfish import Blowfish
from adpcm import decodeADPCMToPCM
from byteutils import *

class Rover:
    def __init__(self):
        ''' Creates a Rover object that you can communicate with.'''
        self.HOST = '192.168.1.100'
        self.PORT = 80
        TARGET_ID = 'Rover_00E04C0DABA2'
        TARGET_PASSWORD = ''
        self.TREAD_DELAY_SEC = 1.0
        self.KEEPALIVE_PERIOD_SEC = 60
        # Create command socket connection to Rover
        self.commandsock = self._newSocket()
        # Send login request with four arbitrary numbers
        self._sendCommandIntRequest(0, [0, 0, 0, 0])
        # Get login reply
        reply = self._receiveCommandReply(82)
        # Extract Blowfish key from camera ID in reply
        cameraID = reply[25:37].decode('utf-8')
        key = TARGET_ID + ':' + cameraID + '-save-private:' + TARGET_PASSWORD
        # Extract Blowfish inputs from rest of reply
        L1 = bytes_to_int(reply, 66)
        R1 = bytes_to_int(reply, 70)
        L2 = bytes_to_int(reply, 74)
        R2 = bytes_to_int(reply, 78)
        # Make Blowfish cipher from key
        bf = _RoverBlowfish(key)
        # Encrypt inputs from reply
        L1,R1 = bf.encrypt(L1, R1)
        L2,R2 = bf.encrypt(L2, R2)
        # Send encrypted reply to Rover
        self._sendCommandIntRequest(2, [-14696588, 292529175, 1044086044, -1771855375])
        # Ignore reply from Rover
        self._receiveCommandReply(26)
        # Start timer task for keep-alive message every 60 seconds
        self._startKeepaliveTask()
        # Set up vertical camera controller
        self.cameraVertical = _RoverCamera(self, 1)
        # Send video-start request
        self._sendCommandIntRequest(4, [1])
        # Get reply from Rover
        reply = self._receiveCommandReply(29)
        # Create media socket connection to Rover
        self.mediasock = self._newSocket()
        # Send video-start request based on last four bytes of reply
        self._sendRequest(self.mediasock, 'V', 0, 4, map(ord, reply[25:]))
        # Send audio-start request
        self._sendCommandByteRequest(8, [1])
        # Ignore audio-start reply
        self._receiveCommandReply(25)
        # Receive images on another thread until closed
```

```
        self.is_active = True
        self.reader_thread = _MediaThread(self)
        self.reader_thread.start()

    def close(self):
        ''' Closes off commuincation with Rover.'''
        self.keepalive_timer.cancel()
        self.is_active = False
        self.commandsock.close()
        if self.mediasock:
            self.mediasock.close()

    def turnStealthOn(self):
        ''' Turns on stealth mode (infrared).'''
        self._sendCameraRequest(94)

    def turnStealthOff(self):
        ''' Turns off stealth mode (infrared).'''
        self._sendCameraRequest(95)

    def moveCameraVertical(self, where):
        ''' Moves the camera up or down, or stops moving it.  A nonzero value for the
            where parameter causes the camera to move up (+) or down (-).  A
            zero value stops the camera from moving.'''
        self.cameraVertical.move(where)

    def _startKeepaliveTask(self,):
        self._sendCommandByteRequest(255)
        self.keepalive_timer = \
            threading.Timer(self.KEEPALIVE_PERIOD_SEC, self._startKeepaliveTask, [])
        self.keepalive_timer.start()

    def _sendCommandByteRequest(self, id, bytes=[]):
        self._sendCommandRequest(id, len(bytes), bytes)

    def _sendCommandIntRequest(self, id, intvals):
        bytevals = []
        for val in intvals:
            for c in struct.pack('i', val):
                bytevals.append(ord(c))
        self._sendCommandRequest(id, 4*len(intvals), bytevals)

    def _sendCommandRequest(self, id, n, contents):
        self._sendRequest(self.commandsock, 'O', id, n, contents)

    def _sendRequest(self, sock, c, id, n, contents):
        bytes = [ord('M'), ord('O'), ord('_'), ord(c), id, \
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, n, 0, 0, 0, 0, 0, 0, 0]
        bytes.extend(contents)
        request = ''.join(map(chr, bytes))
        sock.send(request)

    def _receiveCommandReply(self, count):
        reply = self.commandsock.recv(count)
        return reply

    def _newSocket(self):
        sock = socket.socket()
        sock.connect((self.HOST, self.PORT))
        return sock
```

```
    def _sendDeviceControlRequest(self, a, b) :
        self._sendCommandByteRequest(250, [a,b])

    def _sendCameraRequest(self, request):
        self._sendCommandByteRequest(14, [request])

class Rover20(Rover):
    def __init__(self):
        Rover.__init__(self)
        # Set up treads
        self.leftTread = _RoverTread(self, 4)
        self.rightTread = _RoverTread(self, 1)

    def close(self):
        ''' Closes off commuincation with Rover.'''
        Rover.close(self)
        # Stop moving treads
        self.setTreads(0, 0)

    def getBatteryPercentage(self):
        ''' Returns percentage of battery remaining'''
        self._sendCommandByteRequest(251)
        reply = self._receiveCommandReply(32)
        return 15 * ord(reply[23])

    def setTreads(self, left, right):
        ''' Sets the speed of the left and right treads (wheels).  + = forward;
        - = backward; 0 = stop. Values should be in [-1..+1].'''
        currTime = time.time()
        self.leftTread.update(left)
        self.rightTread.update(right)

    def turnLightsOn(self):
        ''' Turns the headlights and taillights on.'''
        self._setLights(8)

    def turnLightsOff(self):
        ''' Turns the headlights and taillights off.'''
        self._setLights(9)

    def _setLights(self, onoff):
        self._sendDeviceControlRequest(onoff, 0)

    def processVideo(self, jpegbytes, timestamp_10msec):
        ''' Proccesses bytes from a JPEG image streamed from Rover.
            Default method is a no-op; subclass and override to do something interesting.'''
        pass

    def processAudio(self, pcmsamples, timestamp_10msec):
        ''' Proccesses a block of 320 PCM audio samples streamed from Rover.
            Audio is sampled at 8192 Hz and quantized to +/- 2^15.
            Default method is a no-op; subclass and override to do somethinginteresting.'''
        pass

    def _spinWheels(self, wheeldir, speed):
        # 1: Right, forward
        # 2: Right, backward
        # 4: Left, forward
        # 5: Left, backward
        self._sendDeviceControlRequest(wheeldir, speed)
```

```
class Revolution(Rover):
    def __init__(self):
        Rover.__init__(self)
        self.steerdir_prev = 0
        self.command_prev = 0
        self.goslow_prev = 0
        self.using_turret = False
        # Set up vertical camera controller
        self.cameraHorizontal = _RoverCamera(self, 5)

    def drive(self, wheeldir, steerdir, goslow):
        goslow = 1 if goslow else 0
        command = 0
        if wheeldir == +1 and steerdir ==  0:
            command = 1
        if wheeldir == -1 and steerdir ==  0:
            command = 2
        if wheeldir ==  0 and steerdir == +1:
            command = 4
        if wheeldir ==  0 and steerdir == -1:
            command = 5
        if wheeldir == +1 and steerdir == -1:
            command = 6
        if wheeldir == +1 and steerdir == +1:
            command = 7
        if wheeldir == -1 and steerdir == -1:
            command = 8
        if wheeldir == -1 and steerdir == +1:
            command = 9
        if steerdir == 0 and self.steerdir_prev != 0:
            command = 3
        if command != self.command_prev or goslow != self.goslow_prev:
            self._sendDeviceControlRequest(command, goslow)
        self.steerdir_prev = steerdir
        self.command_prev = command
        self.goslow_prev = goslow

    def processVideo(self, imgbytes, timestamp_msec):
        ''' Proccesses bytes from an image streamed from Rover.
            Default method is a no-op; subclass and override to do something interesting.'''
        pass

    def processAudio(self, audiobytes, timestamp_msec):
        ''' Proccesses a block of 1024 PCM audio samples streamed from Rover.
            Audio is sampled at 8192 Hz and quantized to +/- 2^15.
            Default method is a no-op; subclass and override to do something interesting.'''
        pass

    def useTurretCamera(self):
        '''  Switches to turret camera.'''
        self._sendUseCameraRequest(1)

    def useDrivingCamera(self):
        '''  Switches to driving camera.'''
        self._sendUseCameraRequest(2)

    def moveCameraHorizontal(self, where):
        ''' Moves the camera up or down, or stops moving it.  A nonzero value for the where parameter causes
        The camera to move up (+) or down (-).  A zero value stops the camera from moving.'''
        self.cameraHorizontal.move(where)
```

```
    def _sendUseCameraRequest(self, camera):
        self._sendCommandByteRequest(19, [6, camera])

# "Private" classes ============================================================
# A special Blowfish variant with P-arrays set to zero instead of digits of Pi
class _RoverBlowfish(Blowfish):
    def __init__(self, key):
        ORIG_P = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
        self._keygen(key, ORIG_P)

# A thread for reading streaming media from the Rover
class _MediaThread(threading.Thread):
    def __init__(self, rover):
        threading.Thread.__init__(self)
        self.rover = rover
        self.BUFSIZE = 1024

    def run(self):
        # Accumulates media bytes
        mediabytes = ''
        # Starts True; set to False by Rover.close()
        while self.rover.is_active:
            # Grab bytes from rover, halting on failure
            try:
                buf = self.rover.mediasock.recv(self.BUFSIZE)
            except:
                break
            # Do we have a media frame start?
            k = buf.find('MO_V')
            # Yes
            if k  >= 0:
                # Already have media bytes?
                if len(mediabytes) > 0:
                    # Yes: add to media bytes up through start of new
                    mediabytes += buf[0:k]
                    # Both video and audio messages are time-stamped in 10msec units
                    timestamp = bytes_to_uint(mediabytes, 23)
                    # Video bytes: call processing routine
                    if ord(mediabytes[4]) == 1:
                        self.rover.processVideo(mediabytes[36:], timestamp)
                    # Audio bytes: call processing routine
                    else:
                        audsize = bytes_to_uint(mediabytes, 36)
                        sampend = 40 + audsize
                        offset = bytes_to_short(mediabytes, sampend)
                        index  = ord(mediabytes[sampend+2])
                        pcmsamples = decodeADPCMToPCM(mediabytes[40:sampend], offset, index)
                        self.rover.processAudio(pcmsamples, timestamp)
                    # Start over with new bytes
                    mediabytes = buf[k:]
                # No media bytes yet: start with new bytes
                else:
                    mediabytes = buf[k:]
            # No: accumulate media bytes
            else:
                mediabytes += buf
```

```
class _RoverTread(object):

    def __init__(self, rover, index):
        self.rover = rover
        self.index = index
        self.isMoving = False
        self.startTime = 0

    def update(self, value):
        if value == 0:
            if self.isMoving:
                self.rover._spinWheels(self.index, 0)
                self.isMoving = False
        else:
            if value > 0:
                wheel = self.index
            else:
                wheel = self.index + 1
            currTime = time.time()
            if (currTime - self.startTime) > self.rover.TREAD_DELAY_SEC:
                self.startTime = currTime
                self.rover._spinWheels(wheel, int(round(abs(value)*10)))
                self.isMoving = True

class _RoverCamera(object):
    def __init__(self, rover, stopcmd):
        self.rover = rover
        self.stopcmd = stopcmd
        self.isMoving = False

    def move(self, where):
        if where == 0:
            if self.isMoving:
                self.rover._sendCameraRequest(self.stopcmd)
                self.isMoving = False
        elif not self.isMoving:
            if where == 1:
                self.rover._sendCameraRequest(self.stopcmd-1)
            else:
                self.rover._sendCameraRequest(self.stopcmd+1)
            self.isMoving = True
```

Приложение 2.

```
import random
import keras
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import skimage as sm
from skimage import transform
from keras.models import Sequential
from keras.layers import Convolution2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense
from keras.preprocessing.image import ImageDataGenerator

#WORK WITH GPU
gpu_options = tf.GPUOptions(per_process_gpu_memory_fraction=0.5)
config = tf.ConfigProto(gpu_options=gpu_options, log_device_placement=True)
config.gpu_options.allow_growth = True
sess = tf.Session(config=tf.ConfigProto(gpu_options=gpu_options))
keras.backend.set_session(sess)

# LOADING IMAGE DATA'S AND THEIR LABELS
def load_data(data_directory):
    directories = [d for d in os.listdir(data_directory)
                   if os.path.isdir(os.path.join(data_directory, d))]
    labels = []
    images = []
    for d in directories:
        label_directory = os.path.join(data_directory, d)
        file_names = [os.path.join(label_directory, f)
                      for f in os.listdir(label_directory)
                      if f.endswith(".ppm")]
        for f in file_names:
            images.append(sm.data.imread(f))
            labels.append(int(d))
    return images, labels

# PREPARE A CNN TAKES 64X64X3 (3 FOR RGB) AS INPUT
def prepare_classifier():
    # initialize
    classifier = Sequential()
    # STEP 1 - Convolution
    # 64,64,3 = 64X64 IMAGE DIMENSIONS, 3 FOR RGB
    # 32,3 = 32 SUB SAMPLES, 3X3 CONVOLUTON MATRIX DIMENSIONS, 1X1 STRIDE
DIMENSIONS, PADDING IS VALID
    classifier.add(Convolution2D(32, 3, strides=(1, 1), input_shape=(64, 64, 3), padding='valid',
activation='relu'))
    # STEP 2 - Pooling
    # 2,2 = POOL MATRIX DIMENSIONS
    classifier.add(MaxPooling2D(pool_size=(2, 2)))
    # STEP 3 - Convolution
    # 64,64,3 = 64X64 IMAGE DIMENSIONS, 3 FOR RGB
    # 32,3 = 32 SUB SAMPLES, 3X3 CONVOLUTON MATRIX DIMENSIONS, 1X1 STRIDE
DIMENSIONS, PADDING IS VALID
    classifier.add(Convolution2D(32, 3, strides=(1, 1), activation='relu', padding='valid'))
```

```
            # STEP 4 - Pooling
            # 2,2 = POOL MATRIX DIMENSIONS
            classifier.add(MaxPooling2D(pool_size=(2, 2)))
            # STEP 5 - Flattening
            # FLATTENING FOR THE FULLY CONNECTED LAYER
            classifier.add(Flatten())
            # STEP 5 - NEURAL NETWORK
            # 128 = HIDDEN LAYER BITS
            classifier.add(Dense(output_dim=128, activation='relu'))
            # 62 - OUTPUT BITS FOR 62 CLASSES. SOFTMAX IS SUGGESTED FOR OUTPUT
            classifier.add(Dense(output_dim=62, activation='softmax'))

            # CNN
            # OPTIMIZE ACCURACY WITH ADAM OPTIMIZER AND CATEGORICAL CROSS ENTROPY
LOSS FUNCTION
            classifier.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
            return classifier

        # OUR CURRENT MODEL
        classifier = prepare_classifier()

        # CNN AND IMAGES WITH IMAGE DATA GENERATOR WITH DATA AUGMENTATION FOR
INCREASING THE SUCCESS
        train_datagen = ImageDataGenerator(rescale=1./150, shear_range=0.2, zoom_range=0.2)
        test_datagen = ImageDataGenerator(rescale=1./150, shear_range=0.2, zoom_range=0.2)
        training_set = train_datagen.flow_from_directory("TrafficSigns/Training",
                                                         target_size=(64, 64), batch_size=8, class_mode='categorical')
        test_set = test_datagen.flow_from_directory("TrafficSigns/Testing",
                                                    target_size=(64, 64), batch_size=8, class_mode='categorical')
        # 4552 = TRAINING IMAGES, 2520 = TEST IMAGES
        classifier.fit_generator(training_set, samples_per_epoch=4552, nb_epoch=50,
                                 validation_steps=2520, validation_data=test_set)

        classifier.save('C:\\Users\\vnovichk\\PycharmProjects\\savedmodel\\my_model.h5')  # creates a HDF5 file
'my_model.h5'
        # returns a compiled model
        # identical to the previous one
        # model = load_model('./savedmodel/my_model.h5')

        ROOT_PATH = "C:\\Users\\vnovichk\\PycharmProjects\\TSR"

        test_data_directory = os.path.join(ROOT_PATH, "TrafficSigns/Testing")

        # LOAD IMAGES AND LABELS
        imagesTest, labelsTest = load_data(test_data_directory)
        unique_labels = set(labelsTest)

        # RESIZE ALL IMAGES TO 64X64
        images28 = [transform.resize(image, (64, 64)) for image in imagesTest]

        # CONVERT IT TO ARRAY
        images28 = np.array(images28)
```

```
# Pick 10 random images
sample_indexes = random.sample(range(len(images28)), 10)
# 10 RANDOM IMAGE
sample_images = [images28[i] for i in sample_indexes]
# 10 RANDOM IMAGES'S LABELS
sample_labels = [labelsTest[i] for i in sample_indexes]

# CONVERT THEM TO ARRAY
sample_images_array = np.array(sample_images)

predicted = classifier.predict(sample_images_array)

# PRINT 10 RANDOM IMAGES PREDICTIONS AND TRUE VALUES WITH THEIR IMAGE
print("----10 RANDOM TEST SAMPLES---")
# Display the predictions and the ground truth visually.
fig = plt.figure(figsize=(10, 10))
for i in range(len(sample_images)):
   truth = sample_labels[i]
   prediction = np.argmax(predicted[i])
   plt.subplot(5, 2, 1+i)
   plt.axis('off')
   color = 'green' if truth == prediction else 'red'
   plt.text(70, 20, "Truth:       {0}\nPrediction: {1}".format(truth, prediction),
            fontsize=12, color=color)
   plt.imshow(sample_images[i], cmap="gray")

plt.show()
# TEST WHOLE TESTING DATASET
array = classifier.predict(images28)

# COUNT THE TRUE PREDICTIONS
result = []
for i in range(0, len(array)):
   result.append(np.argmax(array[i]))
result2 = []
true = 0
for i in range(0, len(labelsTest)):
   if labelsTest[i] == int(result[i]):
      true = true+1
      result2.append(1)
   else:
      result2.append(0)

print("----ALL TEST SAMPLES---")
print("True predicted:"+str(true)+" / Total: "+str(len(labelsTest))+" = "+str(true/len(labelsTest)))

print("done")
```

Приложение 2.

```
import keras
import os
from tkinter import *
from PIL import ImageTk, Image
from tkinter import filedialog
import os
from keras.models import load_model
import h5py
import tensorflow as tf
from PIL import Image
from skimage import transform
from skimage import data
import skimage as sm
from skimage import color
from skimage.color import rgb2gray
from keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array
import numpy as np

names = ["Bumpy road", "Speed bump", "Slippery road", "Dangerous left curve", "Dangerous right curve",
"Left curve followed by right curve", "Right curve followed by left curve", "Place where a lot of children come",
"Bicycle crossing", "Cattle crossing", "Construction", "Traffic lights", "Railway crossing with gates", "Caution!",
"Both road narrows", "Left road narrows", "Right road narrows", "Priority at next intersection", "Intersection with
priority to the right", "Give way", "Narrow passage w/o priority", "Stop", "No entry", "No entry for bicycles", "No
entry for vehicles with more mass than indicated", "No entry for vehicles used for goods transport, "No entry for
vehicles which are wider than indicated", "No entry for vehicles which are higher than indicated", "No entry in both
directions", "No turn left", "No turn right", "No overtaking", "Maximum speed", "Road reserved for cyclists and
pedestrians", "Mandatory to follow left", "Mandatory to follow right", "Mandatory to follow forward and right",
"Roundabout", "Mandatory cycleway", "Road reserved for pedestrians, cyclists and mofas", "No parking allowed",
"No parking or standing still allowed", "No parking allowed on this side of the road from 1st day of the month until
the 15th", "No parking allowed on this side of the road from the 16th day of the month until the last", "Narrow
passage w/ priority", "Parking allowed", "  ??????  ", "Parking exclusively for passenger car", "Parking exclusively
for trunks", "Parking exclusively for buses", "Parking mandatory on pavement or verge", "Start of residential zone",
"End of residential zone", "Road with one-way traffic", "No exit", "End of road works", "Pedestrian crossing",
"Bicycle and moped crossing", "Indicating parking","Speed bump", "End of priority road", "Priority road"]

#tkinter app
root = Tk()
root.geometry("550x300+300+150")
root.resizable(width=True, height=True)
model = load_model('C:\\Users\\vnovichk\\PycharmProjects\\TSR\\savemodel\\my_model.h5')

# open a image folder
def openfn():
    filename = filedialog.askopenfilename(title='open')
    return filename

#load image to screen and make prediction with saved CNN model
def open_img():
    x = openfn()
    img = Image.open(x)
    img = img.resize((64, 64), Image.ANTIALIAS)
    img = ImageTk.PhotoImage(img)
    panel = Label(root, image=img)
    panel.image = img
    panel.pack()
    # you must specify the path according to your own model path.
```

```
    testimage = load_img(x,target_size=(64,64))
    testimage = img_to_array(testimage)
    testimage = np.expand_dims(testimage, axis=0)
    prediction = model.predict(testimage)
    #print the prediction
    print(np.argmax(prediction[0]))
    var = StringVar()
    label = Label(root, textvariable=var, relief=RAISED )
    # give the output to screen
    var.set("This photo means: "+str(names[np.argmax(prediction[0])]))
    label.pack()

btn = Button(root, text='open image', command=open_img).pack()

root.mainloop()
```