



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

ФАКУЛЬТЕТ

Информатика и системы управления

КАФЕДРА Программное обеспечение ЭВМ и информационные технологии

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

НА ТЕМУ:

*Распознавание дорожных знаков с использованием
капсульных нейронных сетей*

Студент _____
(Группа)

(Подпись, дата)

(И.О.Фамилия)

Руководитель ВКР

(Подпись, дата)

(И.О.Фамилия)

Консультант

(Подпись, дата)

(И.О.Фамилия)

Консультант

(Подпись, дата)

(И.О.Фамилия)

Нормоконтролер

(Подпись, дата)

(И.О.Фамилия)

2020 г.

Реферат

Расчетно-пояснительная записка:

73 с., 49 рис., 5 табл., 15 источников, 1 прил.

Ключевые слова: нейронные сети, система распознавания, детектирование знаков дорожного движения, капсулльная нейронная сеть, сети глубокого обучения.

Объектом разработки является система распознавания дорожных знаков.

Цель работы — предложить способ реализации системы распознавания автодорожных знаков с использованием архитектуры капсулльной нейросети.

Поставленная цель достигается за счет применения нейросетевых систем распознавания дорожных знаков. Рассмотрен альтернативный метод классификации дорожных знаков с использованием капсулльных нейронных сетей, основанный на действии нейронных сетей, но имеющий свои особенности. Предложен метод локализации знаков с применением глубокой нейросети YOLO.

Содержание

Термины и условные обозначения.....	5
Введение.....	6
1. Аналитический раздел.....	8
1.1 Постановка задачи.....	8
1.1 Обзор существующих разработок	9
1.1.1 Коммерческие решения	9
1.1.2 Научно-исследовательские работы.....	10
1.2 Обзор методов обнаружения.....	11
1.2.1 Эвристические алгоритмы.....	12
1.2.2 Метод Виолы-Джонса.....	12
1.2.3 Гистограммы ориентированных градиентов (HOG)	12
1.2.4 Нейронные сети.....	13
1.2 Принцип работы нейронных сетей.....	14
1.2.1 Модель нейрона.....	14
1.2.2 Функция активации.....	15
1.2.3 Полносвязные слои	16
1.2.4 Обучение сети.....	17
1.3 Свёрточная нейронная сеть.....	18
1.3.1 Обзор архитектуры.....	18
1.3.2 Свёрточный слой	18
1.3.3 Подвыборочный слой	21
1.3.4 Полносвязный слой.....	21
1.4 Капсулльная нейронная сеть.....	22
1.4.1 Идея	22
1.4.2 Обзор архитектуры.....	23
1.4.3 Сверточный слой	24
1.4.4 Слой первичных капсул.....	24
1.4.5 Функция сжатия	25
1.4.6 Капсулльный слой. Алгоритм динамической маршрутизации	25
1.4.7 Функция потерь	29
1.4.8 Реконструкция	30
1.4.9 Особенности	30
1.5 Методы детектирования с помощью глубинного обучения	31
1.5.1 Подход на основе классификации	31
1.5.2 Подход на основе регрессии	31
1.5.3 Выбор метода.....	32
1.5.4 Описание модели YOLOv3.....	33
1.6 Выводы	36

2.	Конструкторский раздел	37
1.1	Подготовка данных для обучения	37
1.2	Валидация данных	38
1.3	Используемые метрики	40
1.4	Предлагаемое решение	42
1.5	Выводы	43
3.	Технологический раздел	44
3.1	Средства разработки	44
3.2	Предварительная обработка данных для задачи классификации	45
3.3	Построение и обучение капсулной нейронной сети	49
3.4	Модификация и обучение YOLOv3	53
3.5	Демонстрация работы	56
3.6	Выводы и особенности работы системы обнаружения	62
4.	Экспериментальный раздел	63
4.1	Предварительная обработка	63
4.2	Влияние параметров на обучение	64
4.2.1	Скорость обучения	64
4.2.2	Размер пакета	66
4.3	Сравнение с другими реализациями	68
4.1	Выводы	68
	Заключение	69
	Список источников и литературы	71
	Приложение А	72

Термины и условные обозначения

Искусственный интеллект (ИИ) (Artificial Intelligence, AI)

— свойство интеллектуальной системы выполнять творческие функции, которые считаются прерогативой человека. Способность компьютера совершать абстрагированные умозаключения или, по крайней мере, имитировать этот процесс, тем самым приближая его к мышлению человека

Машинное обучение (Machine Learning, ML)

— подраздел искусственного интеллекта, изучающий различные способы построения обучающихся алгоритмов. Большая часть машинного обучения — это наука о том, как решать задачу восстановления функции по точкам. Среди множества парадигм и подходов в машинном обучении выделяются нейросети.

Искусственная нейронная сеть (ИНС) (Artificial Neural Network, ANN)

— упрощенная модель биологической нейронной сети.

Компьютерное зрение (Computer Vision, CV)

— научное направление в области искусственного интеллекта, которое занимается задачами, связанными с анализом изображений и видео.

Датасет (Dataset) — набор данных, выборка.

Сокращения названий архитектур нейронных сетей:

- **многослойный персепtron (MultiLayer Perceptron, MLP);**
- **сверточная нейронная сеть (Convolutional Neural Network, CNN);**
- **капсульная нейронная сеть (Capsule Network, CapsNet).**

Введение

В настоящие времена усилия многих разработчиков направлены на создание недорогих и доступных широкому кругу потребителей систем автоматического предупреждения водителя о дорожной ситуации. Для этой цели разрабатываются продвинутые системы помощи водителю “Advanced Driver Assistance Systems” (ADAS), которые постепенно встраиваются в некоторые современные высокотехнологичные серии автомобилей. Система призвана уведомлять о наличии проблематичной ситуации, которая возникла, либо может возникнуть на дороге, то есть автоматизирует и повышает уровень безопасности транспортного средства. Активно развиваются технологии для автономного движения автомобилей, при котором процесс передвижения частично или полностью ложится на искусственный интеллект автопилота.

Одна из важнейших задач подобных систем — распознавание дорожных знаков — имеет несколько важных применений:

- 1) в системах помощи водителю (ADAS), чтобы уведомлять его о наличии дорожных знаков, помочь придерживаться установленного скоростного ограничения на участке, соблюдать ограничения на проезд, обгон и т.д.;
- 2) при автоматическом составлении и пополнении навигационных карт;
- 3) в системах мониторинга знаков на дорогах для служб дорожного хозяйства.

На сегодняшний день разработано несколько алгоритмов и способов, позволяющих распознавать дорожные знаки, однако, показатели точности и скорости распознавания дорожных знаков не всегда позволяют использовать эти подходы в реальных автономных системах управления. Таким образом, задача создания качественного алгоритма распознавания дорожных знаков является актуальной.

Следует отметить, что задачи распознавания являются абсолютно нетривиальными. Постоянные изменения условий, таких как изменение освещения, местоположения и различное окружение объектов усложняют обнаружение и, на данный момент, не существует какого-либо универсального, канонического решения. В то же время существует множество различных подходов к решению подобных задач.

Современные алгоритмы распознавания объектов на изображении используют машинное обучение. Текущий уровень развития вычислительной техники и применение методов обработки изображений с помощью нейронных сетей позволяют создавать системы технического зрения, способные решать задачи анализа изображений не хуже, чем человек, а иногда данные системы могут оказаться эффективнее человека. Структура сетей позволяет выполнять операции нелинейного преобразования, что в результате позволяет решить задачу распознавания объектов. Наиболее распространенным подходом для решения данного типа задач является использование свёрточных нейронных сетей.

Цель данной работы — предложить способ реализации системы распознавания автодорожных знаков с использованием принципиально новой архитектуры нейронной сети — капсульной нейронной сети.

Для достижения поставленной цели необходимо решить следующие задачи:

- 1) исследовать научную литературу и основные подходы по рассматриваемой теме;
- 2) изучить необходимые инструменты и технологии;
- 3) подготовить базу дорожных знаков для обучения модели распознавания;
- 4) создать ПО, реализующее систему распознавания дорожных знаков;
- 5) оценить и проанализировать результаты работы.

1. Аналитический раздел

1.1 Постановка задачи.

Рассмотрим задачу распознавания автодорожных знаков на изображении.

На вход алгоритму распознавания подаётся видеокадр. Выходом алгоритма являются обрамляющие прямоугольники, содержащие автодорожные знаки на кадрах и классы найденных знаков. Таким образом, задача распознавания дорожных знаков является частным случаем задачи распознавания образов, которую можно разделить на два этапа: выделение (локализация, детектирование) и классификация. На первом этапе выделяются все знаки на кадре, на втором этапе найденные знаки классифицируются. Следует отметить, что применение капсулльных нейронных сетей имеет место именно в задаче классификации. Постановка задачи представлена на рис. 1.1.



Рис. 1.1. Постановка задачи. Диаграмма верхнего уровня.

Основные требования к системе:

- Работа в пределах некоторой области видимости (30-100 метров).
- Работоспособность и устойчивость при различном освещении и погодных условиях.
- Устойчивость к изменению угла, под которым наблюдается дорожный знак.

1.1 Обзор существующих разработок

Существующие решения для систем распознавания дорожных знаков можно разбить на две категории:

- коммерческие продукты,
- научные работы и статьи.

1.1.1 Коммерческие решения

Системы распознавания дорожных знаков впервые появились в 2008 году в компаниях BMW и MercedesBenz. Тогда они были со значительными ограничениями и умели распознавать только знаки ограничения скорости. В настоящее время ведущие мировые производители автомобилей стараются встроить подобные технологии в свои продукты.

Применяемые на автомобилях системы распознавания дорожных знаков имеют типовую конструкцию, которая включает видеокамеру, блок управления и средство вывода информации.

Видеокамера, как правило, располагается на ветровом стекле за зеркалом заднего вида. Камера снимает пространство перед автомобилем в зоне расположения дорожных знаков и передает изображение в электронный блок управления. Видеокамера также используется другими системами активной безопасности.

Электронный блок управления решает задачи распознавания формы, цвета знака, надписи (величина скорости), информационных таблиц (вид транспорта, время действия, зона действия). Блок анализирует фактическую скорость автомобиля, сравнивает её с максимально допустимой скоростью и осуществляет визуальное и звуковое предупреждение водителя при отклонении.

Изображение в виде знака ограничения скорости выводится на дисплей приборов или дисплей информационной системы и остается видимым, пока ограничение не закончится или будет изменено.

В ряде конструкций система распознавания дорожных знаков взаимодействует с навигационной системой и использует сведения о знаках ограничения скорости из навигационных карт. Даже если знак не будет определен видеокамерой, информация о нем будет выведена на панель приборов.

Названия решений от разных производителей:

- **Speed Limit Assist** от Mercedes-Benz.
- **Road Sign Information** от Volvo.
- **Traffic Sign Recognition (TSR)** встречается в таких авто, как Audi, Ford, BMW, Volkswagen.
- **Opel Eye** от одноименного производителя.

Отличие решений заключается в качестве используемого оборудования и логики работы алгоритмов распознавания объектов на дороге.

К особенностям приведённых систем можно отнести то, что они распознают только знаки ограничения скорости и иногда некоторые предупреждающие и запрещающие знаки.

Системы распознавания значительно повышают цену автомобилей. Так как механизмы распознавания — обычно коммерческая тайна, то сложно оценить и делать выводы по качеству работы и надежности этих систем.

1.1.2 Научно-исследовательские работы

Из последних работ можно выделить статью 2016 года «Российская база изображений автодорожных знаков» [8], выполненная авторами Шахуро В.И. и Конушин А.С. В их работе представлена новая открытая российская база изображений дорожных знаков, а также применены современные алгоритмы детектирования и классификации.

Выборка, представленная в статье, получена с широкоформатных видеорегистраторов, установленных за лобовым стеклом машины. Кадры получены в различные времена года, при разном освещении и погодных условиях. В статье показано, что российская база превосходит другие публичные базы по количеству изображений и классов знаков.

Для детектирования знаков использовалась реализация каскадного метода на интегральных признаках из библиотеки Петра Доллара [15]. Для подсчёта точности и полноты использовалась мера IoU (отношение площади пересечения к площади объединения прямоугольников). Обнаружение считалось верным, если $\text{IoU} > 0.5$. В качестве метрики качества использовалась площадь под ROC-кривой, AUC (Area Under Curve). Результаты тестирования детектора на нескольких выборках показывают значения качества AUC, не превосходящие 0.90.

Для классификации знаков использовалась сверточная нейронная сеть, состоящая из 8 слоев. Результаты тестирования классификатора на вырезанных знаках показывают значения точности на выборках, не превосходящие 92.90 %.

Таким образом, описанные решения по детектированию и классификации дорожных знаков работают недостаточно качественно для практических приложений и требует улучшений.

1.2 Обзор методов обнаружения

Задача обнаружения объектов на изображении — одна из фундаментальных в области компьютерного зрения и обработки изображений.

Структуру системы обнаружения объектов на изображении можно представить в виде совокупности трёх блоков: предварительная обработка изображения, формирование признакового описания, принятие решения.

Предварительная обработка изображения обычно заключается в применении к изображению фильтра, подавляющего шум. Часто реализация данного этапа не выполняется, в этом случае ответственность за устойчивость к шуму перекладывается на метод формирования признакового описания.

При формировании признакового описания объекта используется гистограмма ориентированных градиентов, каскады Хаара и т. д.

Блок принятия решения заключается в формировании модели на основе признакового описания с использованием нейронных сетей, метода опорных векторов, k-ближайших соседей

Однако существуют методы глубокого обучения способны выполнять обнаружение объектов «целиком» без конкретного определения признаков и обычно основаны на сверточных нейронных сетях (CNN).

Алгоритмы распознавания объектов на изображении можно условно разделить на три группы: эвристические алгоритмы, подходы классического машинного обучения и глубокое обучение.

1.2.1 Эвристические алгоритмы

Исторически эвристические алгоритмы являются самыми ранними. Такие методы для поиска дорожных знаков на изображении используют то, что знаки имеют фиксированный цвет и форму. Такие системы достаточно просты в реализации и работают с высокой скоростью. К недостаткам таких алгоритмов можно отнести неустойчивость при размытых изображениях и сложность их построения в случае большого количества знаков разных цветов и формы. Жёстко запрограммированные правила лишают систему гибкости и устойчивости. Далее рассмотрим основные подходы с использованием машинного обучения.

1.2.2 Метод Виолы-Джонса

Подход на основе каскада слабых классификаторов начинается с работы Виолы и Джонса 2001 года [3], в которой впервые была решена задача выделения лиц на изображении в реальном времени. Это способ построения «сильного» классификатора, на основе «слабых». Под термином сила понимается качество работы классификатора в решении поставленной задачи.

В алгоритме применяется принцип сканирующего окна. Метод использует быстро вычисляемые интегральные признаки и неглубокие деревья решений (слабые классификаторы), которые объединяются с помощью бустинга в каскад (сильный классификатор). Сильный классификатор последовательно применяет слабые классификаторы. После каждого слабого классификатора часть окон отбрасывается. Таким образом, целиком каскад проходят только окна с объектами и наиболее сложными примерами фона. Модифицированные варианты каскадного подхода показывают высокое качество и скорость на задачах выделения объектов с небольшой внутриклассовой изменчивостью.

1.2.3 Гистограммы ориентированных градиентов (HOG)

Другой подход основан на гистограммах ориентированных градиентов (HOG) и машине опорных векторов (SVM). В работе [4] 2005 г. была показана эффективность метода для задачи выделения пешеходов на изображении. Описание изображения с помощью HOG оказалось эффективным и для задач многоклассовой классификации. Это один из первых алгоритмов, которыйправлялся с задачей распознавания с достаточной скоростью и надёжностью.

Основная идея заключается в том, что форма и вид объектов на изображении могут хорошо описываться распределением относительных величин градиентов функции интенсивности, характеризующих направление границ объектов. Как правило, построение этих дескрипторов происходит путем разбиения изображения на ячейки, и присвоения каждой ячейке гистограммы направлений градиентов для пикселей внутри ячейки, их комбинация и является дескриптором. С целью увеличения точности изображение делают чёрно-белым, а локальные гистограммы нормализуют по контрасту относительно меры интенсивности, вычисляемой на большем фрагменте изображения. Нормализация по контрасту позволяет добиться большей инвариантности к освещению.

Конечным этапом применения метода в оригинальной работе является классификация дескрипторов с помощью системы обучения с учителем, в частности, использовался метод опорных векторов.

1.2.4 Нейронные сети

Этот подход переживает бурный рост в последние годы. Метод становится одним из самых популярных в машинном обучении. Глубинное обучение стало активно применяться для решения различных задач компьютерного зрения после работы [5] 2012 г., в которой свёрточная нейронная сеть успешно используется для классификации изображений базы ImageNet на 1000 классов. В настоящее время подход, основанный на глубинном обучении, является наиболее перспективным. Нейронные сети показывают лучшие результаты по сравнению с альтернативными методами в таких областях, как распознавание речи, обработка естественного языка, компьютерное зрение, медицинская информатика, сложные прогнозы и др. [1]. Одна из причин успешного применения нейронных сетей заключается в том, что сеть автоматически выделяет из данных важные признаки, необходимые для решения задачи. В альтернативных алгоритмах машинного обучения признаки должны выделяться людьми, существует специализированное направление исследований — инженерия признаков (feature engineering). Однако при обработке больших объемов данных нейронная сеть справляется с выделением признаков гораздо лучше, чем человек.

1.2 Принцип работы нейронных сетей

Чтобы описать применение нейронных сетей в компьютерном зрении необходимо понять, как работает классическая базовая архитектура нейронных сетей под названием многослойный персептрон (multilayer perceptron, MLP), используемая в других, более сложных архитектурах.

1.2.1 Модель нейрона

У биологического нейрона есть **тело**, которое накапливает и некоторым образом преобразует сигнал, приходящий к нему через **дendриты** — короткие отростки, функция которых заключается в приеме сигналов от других нервных клеток. Накопив сигнал, нейрон передает его по цепочке другим нейронам по длинному отростку — **аксону**, который связан с дендритами других нейронов, и так далее. Дендриты с аксонами связаны не напрямую, а через так называемые **синапсы**. Нейрон выдаст сигнал, если суммарное возбуждение превысит некоторое предельное значение, которое в общем случае меняется в некоторых границах. В противном случае на аксон сигнал выдан не будет. Большинство нейронных сетей моделируют именно эти простые свойства (рис. 1.2).

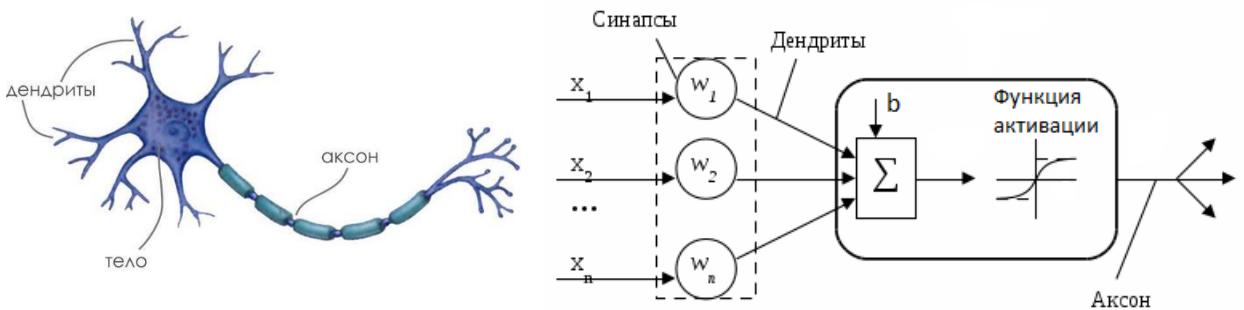


Рис. 1.2. Упрощенная модель нейрона.

В искусственном нейроне входные данные x_i — это сигналы, поступающие к нейрону. Это могут быть сигналы с других нейронов. Веса w_i — это эквиваленты синаптической связи. Вес представлен действительным числом, на которое будет умножено значение входящего в нейрон сигнала. Вес показывает, насколько сильно между собой связаны те или иные нейроны — это коэффициент связи между ними.

В теле нейрона накапливается взвешенная сумма z от перемножения значений сигналов и весов. Порог, или сдвиг b (англ. bias), определяет, насколько большой должна быть взвешенная сумма, чтобы активировать нейрон.

$$z = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b.$$

Данная запись не всегда удобна, её можно представить в виде:

$$z = w_1x_1 + w_2x_2 + \cdots + w_nx_n + w_0x_0, \quad \text{где } x_0 = 1, w_0 = b; \quad \text{то есть } z = \sum_{i=0}^n w_i x_i.$$

1.2.2 Функция активации

Вычислив взвешенную сумму, можно получить любое число в широком диапазоне значений. Разумно использовать функцию, которая «сжимает» весь диапазон. Она называется функцией активации нейрона. Активация нейрона — мера того, насколько положительна взвешенная сумма.

Одной из наиболее распространенных является нелинейная функция с насыщением, так называемая логистическая функция (сигмоида) (рис.1.3, слева). Сильно отрицательные значения оказываются близки к нулю, сильно положительные — к единице. Следует отметить, что во всех точках функция имеет производную, которая может быть выражена через эту же функцию.

Однако в современных нейросетях чаще используется ReLU-функция (Rectified Linear Unit), облегчающая обучение нейросети (рис.1.3, справа). Функция ReLU соответствует биологической аналогии того, что нейроны могут находиться в активном состоянии либо нет. Если пройден определенный порог, то срабатывает функция, а если не пройден, то нейрон просто остается неактивным, с активацией 0.

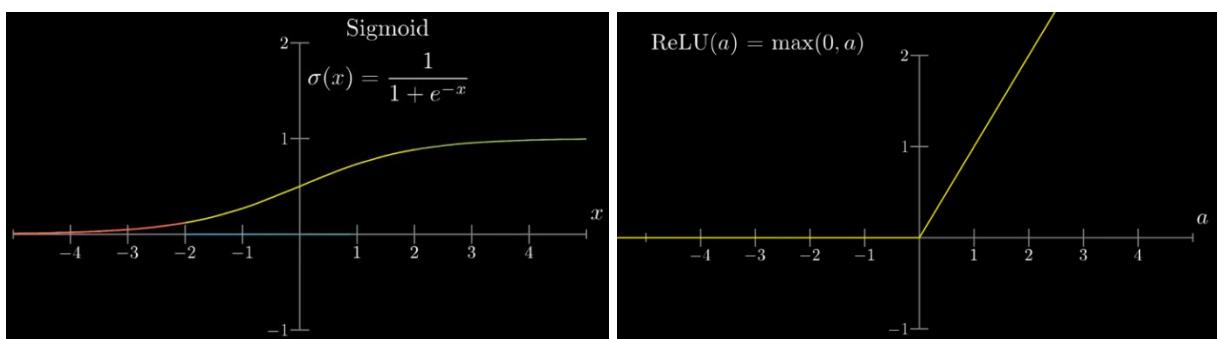


Рис. 1.3. Логистическая функция (слева) и ReLU-функция (справа).

1.2.3 Полносвязные слои

Рассмотрим сеть нейронов на примере задачи распознавания цифр на изображении. Пусть есть число, изображенное на черно-белой картинке в разрешении 28×28 пикселей. Тогда есть $28 \times 28 = 784$ входных сигнала (нейрона), содержащие различные числа от 0 до 1 (рис. 1.4). Описанные 784 нейрона образуют первый слой нейросети. Последний слой содержит 10 нейронов, каждый из которых соответствует одной из десяти цифр. В этих числах активация — это также число от 0 до 1, отражающее насколько система уверена, что входное изображение содержит соответствующую цифру. Также есть пара средних слоев, называемых скрытыми. Выбор количества скрытых слоев и содержащихся в них нейронов произволен, обычно они выбираются из определенных представлений о задаче, решаемой нейронной сетью (на рисунке 2 скрытых слоя по 16 нейронов). Каждый нейрон текущего слоя имеет связи с каждым нейроном предыдущего слоя. Часто входные сигналы не считают за самостоятельный слой, т.к. он не участвует ни в каких операциях. Так образуется полносвязная нейронная сеть. Её можно называть *глубокой*, если она имеет более одного скрытого слоя. Каждый такой слой увеличивает нелинейность вычислений нейронной сети.

В данном примере можно надеяться, что каждый нейрон из первого скрытого слоя нейросети осуществляет операцию распознавания граней и кривых на изображении. Второй же скрытый слой распознает более высокоуровневые абстракции, составленные из шаблонов предыдущего слоя. Активации в одном слое определяют активации в следующем.

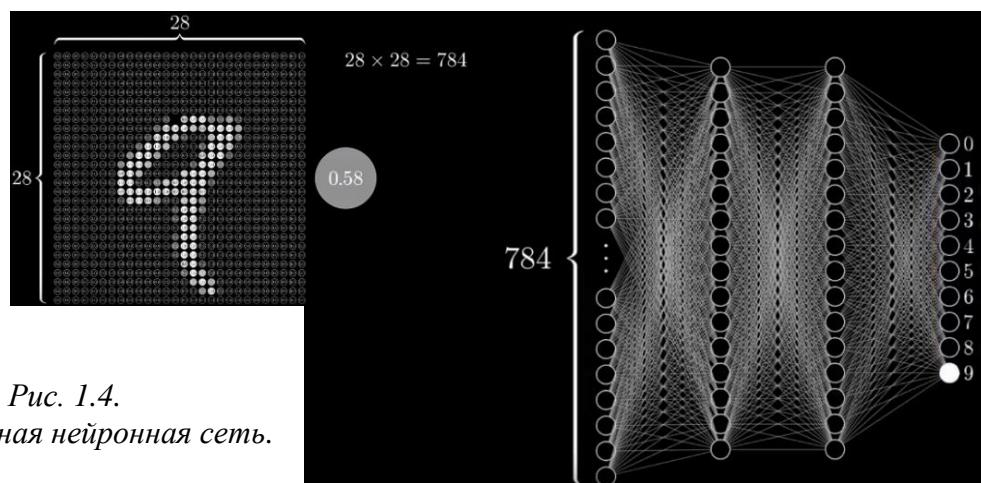


Рис. 1.4.
Полносвязная нейронная сеть.

1.2.4 Обучение сети

Обучение нейронной сети — это процесс изменения весов, т.е. коэффициентов связи между нейронами. Для примера выше (рис. 1.4) для первого скрытого слоя имеется 784×16 весов + 16 сдвигов (всего ~ 13 тыс.).

Чтобы оценить, насколько плохо работает сеть вводят **функцию потерь**, или стоимости C (cost function), которую необходимо минимизировать. Она может быть разной, например, сумма квадратов разностей между реальными значениями активации выходного слоя и их идеальными значениями. Чтобы понять, насколько хорошо произошло обучение нейросети, необходимо определить среднее значение потерь для *всех* изображений обучающей выборки. Функция потерь зависит от всех коэффициентов в нейронной сети.

Ключевым алгоритмом обучения нейронной сети является **обратное распространение ошибки**, в основе которого лежит **идея градиентного спуска**. Градиент ∇ — это вектор производных, смысл которого заключается в том, что он указывает направление самого крутого подъема. Тогда направление наиболее быстрого спуска представляет собой отрицательный градиент $-\nabla$. Рассчитывая градиент функции потерь, делается шаг в направлении $-\nabla C$, процедура повторяется до тех пор, пока мы не окажемся в минимуме (рис. 1.5).

Каждый шаг **классического градиентного спуска** предполагает использование всех примеров обучающей выборки. Это замедляет процесс спуска, поэтому применяют **стохастический градиентный спуск**. Случайным образом данные обучающей выборки перемешиваются и разделяются на подгруппы. Далее рассчитывается шаг градиентного спуска для одной подгруппы. Кривая обучения будет похожа не на равномерный спуск с холма, а на петляющую траекторию, делающую более быстрые шаги, но приходящую так же к минимуму.

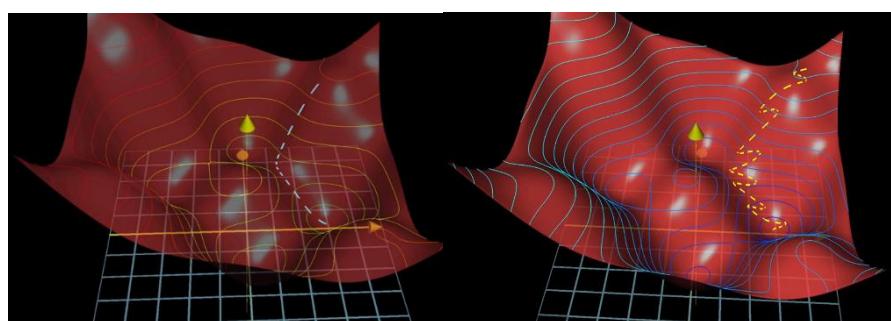


Рис. 1.5. Градиентный спуск функции двух переменных: классический (слева) и стохастический (справа).

1.3 Свёрточная нейронная сеть

Свёрточные нейронные сети изначально разрабатывались для работы с изображениями. Когда-то данный тип сетей произвел революцию в компьютерном зрении и распознавании образов. Также его используют для распознавания речи, обработки аудиосигналов, обработки временных рядов, для анализа смысла текстов и даже для игры в Го.

1.3.1 Обзор архитектуры

В свёрточных сетях используют 3 типа слоев: свёрточный, подвыборочный и полносвязный. Типовая структура таких сетей включает в себя чередование свёрточных и подвыборочных (субдискретизирующих) слоев в начале и наличие нескольких полносвязных слоев на выходе (рис 1.6). Первые два типа слоёв формируют входной вектор признаков для многослойного персептрана.

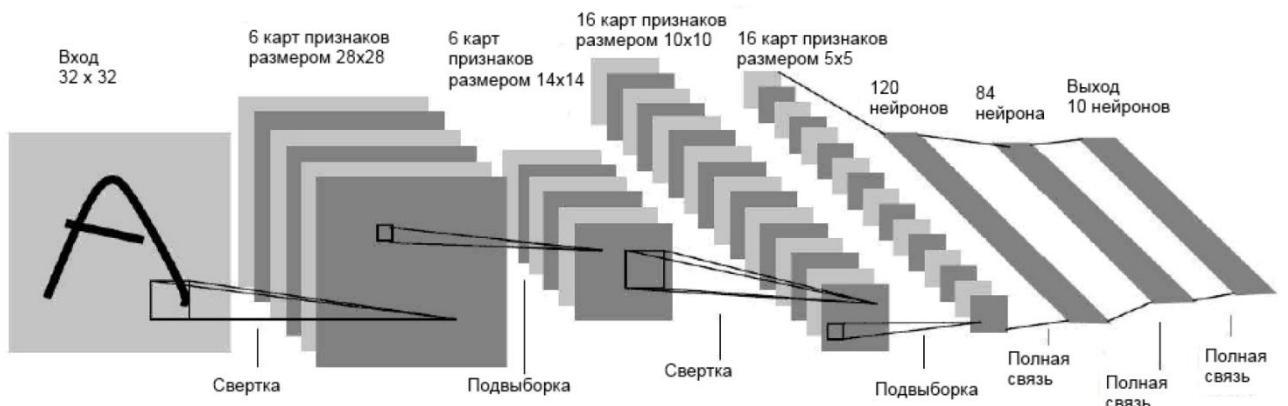


Рис. 1.6. Пример архитектуры свёрточных нейронных сетей.

1.3.2 Свёрточный слой

Свое название сеть получила по названию операции — свертка. Основная идея в том, что на вход каждого нейрона подается не все изображение (или выходы предыдущего слоя), а только некоторая область, причем на каждый нейрон своя. То есть каждый нейрон последующего слоя "смотрит" на небольшой кусочек (например, 3 на 3 или 5 на 5 пикселей) предыдущего слоя и обобщает с него информацию для передачи дальше. Это значительно сокращает объем вычислений и позволяет сохранить топологию изображения от слоя к слою.

Ядро, или фильтр, представляет из себя матрицу весов. Ядро «скользит» над двумерным изображением, поэлементно выполняя операцию умножения с той частью входных данных, над которой оно сейчас находится, и затем суммирует все полученные значения в один выходной пиксель (рис.1.7).

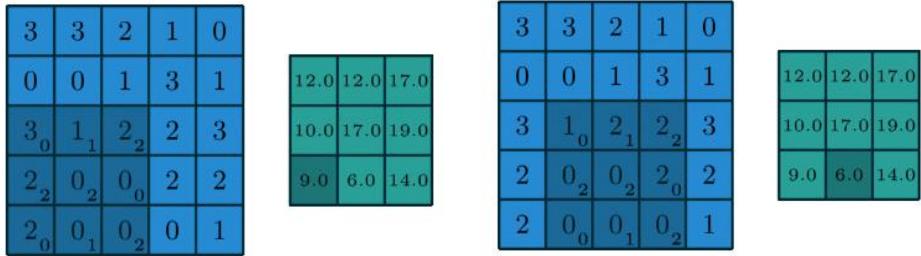


Рис. 1.7. Пример свертки. Нижние индексы – значения весов в ядре 3x3.

Ядро повторяет эту процедуру с каждой локацией, над которой оно находится, преобразуя двумерную матрицу в другую все еще двумерную матрицу (**карту признаков**). Признаки на выходе являются взвешенными суммами признаков на входе, расположенных примерно в том же месте, что и выходной пиксель. При этом размерность выхода уменьшается:

$$S_{out} = S_{in} - S_{kernel} + 1, \quad \text{где}$$

S_{out} – размер стороны выходной карты;

S_{in} – размер стороны входной карты;

S_{kernel} – размер стороны ядра.

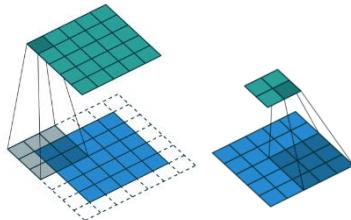
На рисунке размер входного изображения равен 5x5, а размер ядра — 3x3. После применения ядра на выходе получаем карту признаков размера 3x3.

Матрица весов, или ядро свёртки, кодирует какой-либо признак, например, наличие угла или линии. Каждое ядро формирует отдельный экземпляр карты признаков. Нейроны одной карты имеют одинаковые весовые коэффициенты. Это позволяет находить один и тот же признак по всей области изображения. Обычно используется несколько ядер, и в результате получается стек независимых карт признаков одного уровня.

При обучении сети для целой карты происходит корректировка только матрицы весов. В результате обучение сети проходит намного быстрее, чем у обычного персептрона, у которого настраивается каждая связь между нейронами соседних слоев.

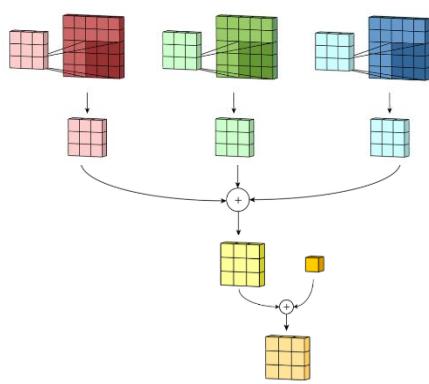
Следует отметить, что крайние пиксели никогда не оказываются в центре ядра, потому что тогда ядру не над чем будет скользить за краем. Иногда применяется техника под названием **padding**, в которой к краям входной карты добавляются поддельные (fake) пиксели (например, нулевого значения или значения, близкого к крайним пикселям). Таким образом, в результате свертки получится выходная матрица того же размера, что и входная (рис. 1.8, слева).

Кроме того, часто применяется техника **striding**, идея которого в том, чтобы пропустить некоторые области, над которыми скользит ядро. Шаг 1 означает, что берутся пролеты через пиксель, шаг 2 — через каждые два пикселя, уменьшая матрицу примерно в 2 раза (рис. 1.8, справа).



*Рис. 1.8.
Техники padding (слева) и striding с шагом 2 (справа).
Зелёная матрица — результат свёртки.*

Большинство изображений имеют 3 цветовых канала (RGB). Для каждого отдельного входного канала есть одно ядро. Некоторые ядра могут иметь больший вес, для того чтобы уделять больше внимания определенным входным каналам. Каждая из обработанных ядром версий суммируется вместе для формирования одного канала (рис. 1.9). Затем добавляется смещение к выходному каналу для создания конечного выходного канала.



*Рис. 1.9.
Применение фильтра к цветному изображению.*

Перед передачей входа другому слою свертки применяется функция активации. Чаще всего — ReLU, так как она не использует дорогостоящие операции (вычисление экспоненты, деление), не склонна к перенасыщению, ускоряет сходимость градиентного спуска по сравнению с другими функциями.

1.3.3 Подвыборочный слой

Подвыборочный, или субдискретизирующий (subsampling), или пулинговый (pooling), слой выполняет функцию уменьшения размерности сформированных карт признаков. Пулинг интерпретируется так: если на предыдущей операции свёртки уже были выявлены некоторые признаки, то для дальнейшей обработки настолько подробное изображение уже не нужно, и оно уплотняется до менее подробного. Из небольшой квадратной области нейронов карты признаков выбирается максимальный (max-pooling) и принимается за один нейрон карты признаков этого слоя. Реже используют операцию нахождения среднего значения. Данный слой ускоряет дальнейшие вычисления и позволяет спроектировать сеть, более инвариантную к масштабу входного изображения.

Современные сети отказываются от субдискретизирующих слоев в пользу чередующихся сверток, когда необходимо уменьшить размер на выходе.

1.3.4 Полносвязный слой

После нескольких прохождений свёртки изображения и уплотнения с помощью пулинга система перестраивается от конкретной сетки пикселей с высоким разрешением к более абстрактным картам признаков. В конце остаётся небольшое число данных, которые интерпретируются как самые абстрактные понятия, выявленные из исходного изображения. Эти данные объединяются и передаются на полносвязную нейронную сеть, которая тоже может состоять из нескольких слоёв. При этом полносвязные слои уже утрачивают пространственную структуру пикселей и обладают сравнительно небольшой размерностью (по отношению к количеству пикселей исходного изображения). Персептрон хорошо зарекомендовал себя в распознавании простых объектов.

1.4 Капсулальная нейронная сеть

В 2017 году один из основоположников подхода обратного распространения ошибки Джонатан Хинтон опубликовал статью [2], в которой описал капсулальные нейронные сети и предложил алгоритм динамической маршрутизации между капсулами. Данный вид архитектуры появился как результат понимания несовершенства сверточных нейронных сетей.

1.4.1 Идея

В сверточной нейронной сети подвыборочный слой (pooling) определяет, какой нейрон пустить в следующий слой. Маршрутизация (routing) — правило, определяющее путь нейрона от текущего слоя к следующему. Нужно различать следующие понятия.

- **Инвариантность.** Неважно, где проявился паттерн в картинке, операция maxpooling ужмет проявленные активации до одного значения. Однако не совсем корректно ожидать, что если в картинке что-то сместились, то результат будет один и тот же (рис. 1.10). Результат должен точно так же сместиться. Теряется позиционная и пространственная информация. Это одна из причин, почему у сверточной нейронной сети могут быть проблемы устойчивости к поворотам в изображении. Это решается тем, что в выборку добавляют повернутые версии объектов.
- **Эквивариантность.** Выполняется, пока нет операции pooling (рис 1.11).

Рис. 1.10.

Если игнорировать позиционные отношения, то могут возникнуть трудности в отличии этих изображений.

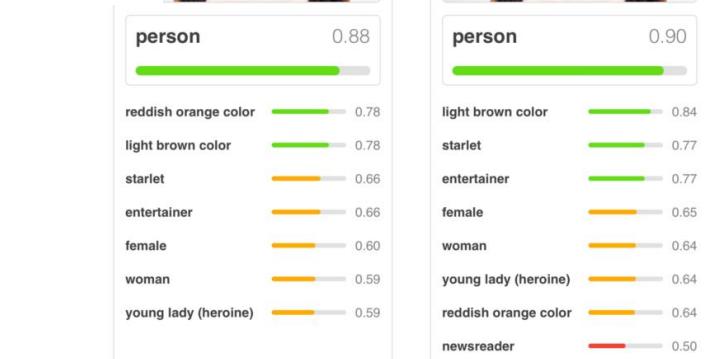
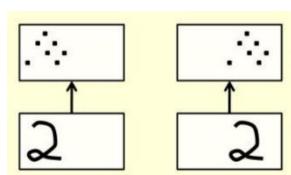
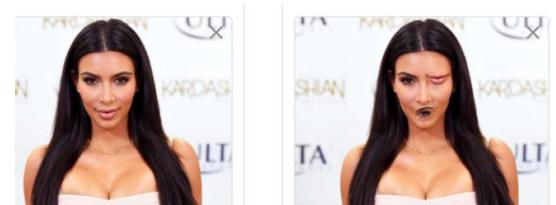


Рис. 1.11.

Интерпретация понятия «эквивариантность».

Было предложено работать не с одним нейроном, выход которого определяет независимый паттерн, а с группой. Получится вектор, описывающий какой-либо признак. **Капсула** — вектор нейронов (чисел), которые описывают существование некоторого объекта. Эти числа могут описывать позицию, поворот, размеры объекта и т.д. В нейроне кодируется одно число, в капсule — несколько. Например, есть 3 капсулы: первая кодирует глаз, вторая — нос, третья — губы. Это капсулы одного уровня. На более высоком уровне находится капсулла, кодирующая положение лица.

Иными словами, капсулы инкапсулируют информацию о состоянии шаблона, который обнаруживаются в векторной форме. Они кодируют вероятность обнаружения объекта как длину выходного вектора. Состояние паттерна кодируется как направление, в котором указывает вектор. Когда обнаруженный объект перемещается по изображению, вероятность остается неизменной (длина вектора не изменяется), но ориентация меняется.

1.4.2 Обзор архитектуры

Условно можно выделить следующие типы слоев, которые могут использоваться в капсулльных сетях: свёрточный слой, слой первичных капсул, капсулльный слой, полносвязный слой. Архитектуру можно разделить на 2 части: кодирующая часть сети (encoder) и декодер (decoder). На рисунке 1.12 представлен пример архитектуры CapsNet, предложенный в оригинальной статье для распознавания цифр.

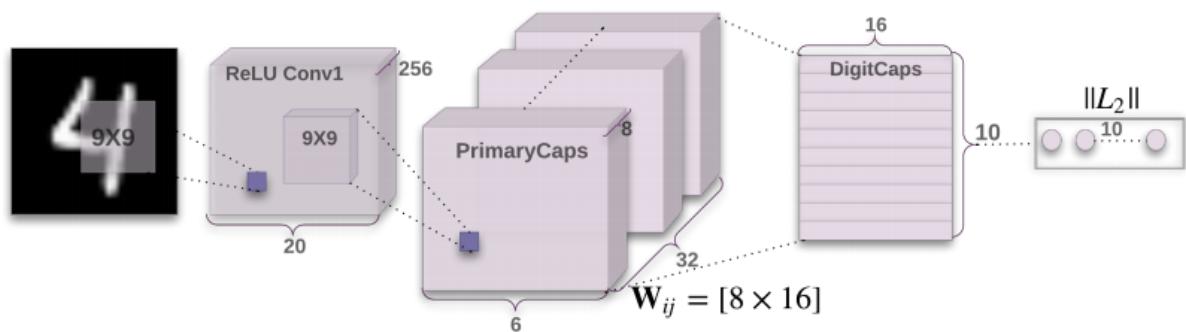


Рис. 1.12. Пример архитектуры капсулльных нейронных сетей CapsNet.

1.4.3 Сверточный слой

К входному изображению сначала применяется один или несколько свёрточных слоев. Цель состоит в извлечении базовых паттернов.

В CapsNet к входному изображению 28x28 применяется один такой слой. Он имеет 256 ядер размером 9x9x1 и шаг 1, затем следует активация ReLU. На выходе получится 256 карт размера $(28-9+1) \times (28-9+1)$, или 20x20, на которых определены некоторые низкоуровневые признаки. Можно вычислить количество параметров. Нужно помнить, что каждое ядро имеет смещение. Следовательно, этот слой имеет $(9 \times 9 + 1) \times 256 = 20992$ обучаемых параметров.

1.4.4 Слой первичных капсул

Цель слоя — взять объекты, обнаруженные свёрточным слоем, и создать комбинации этих объектов, объединённые в капсулы.

В CapsNet проводится свертывание по стеку из 256 выходов. Вместо ядра 9x9x1 имеем ядро размера 9x9x256. Теперь шаг равен 2, и выходная размерность равна $(20-9+1) / 2 = 6$. Результат этого шага — набор из 256x6x6 выходов. Теперь «нарежем» этот стек на 32 колоды (канала) по 8 карт в каждой (рис. 1.13). Каждая такая колода — это тензор 6x6x8. Так формируются первичные капсулы, где каждая капсула — массив из 8 значений. Одна колода имеет 36 капсул, всего 1152 капсулы.

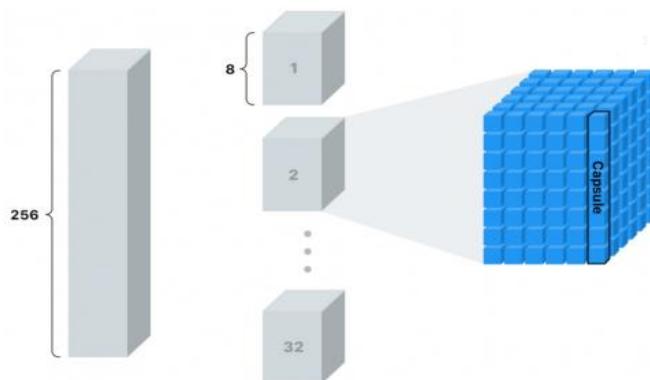


Рис. 1.13.
Формирование
первичных капсул.

Теперь в капсule можно хранить 8 значений для данной локации. Это дает возможность хранить больше информации, чем просто знание о наличии или отсутствии паттерна в этом месте. Выбор количества каналов (колод) и длины одной капсулы — это вопрос архитектуры.

Каждая колода применяет восемь сверточных ядер 9x9x256 (с шагом 2). Тогда количество параметров в этом слое равно $(9 \times 9 \times 256 + 1) \times 8 \times 32 = 5308672$.

1.4.5 Функция сжатия

Длина вектора интерпретируется как вероятность обнаружения данного паттерна капсулой. Поэтому вводится новая нелинейная функция активации, которая «раздавливает» вектор так, чтобы он имел длину не более 1, что соответствует вероятности, но не менял своего направления:

$$v_j = \frac{\|s_j\|^2}{1 + \|s_j\|^2} \frac{s_j}{\|s_j\|}, \text{ где}$$

s_j — исходный вектор произвольной длины;

v_j — "сжатый" вектор, сонаправленный с s_j .

Правый множитель масштабирует входной вектор, чтобы он имел единичную длину и сохранял ориентацию, а левый выполняет дополнительное масштабирование (рис. 1.14).

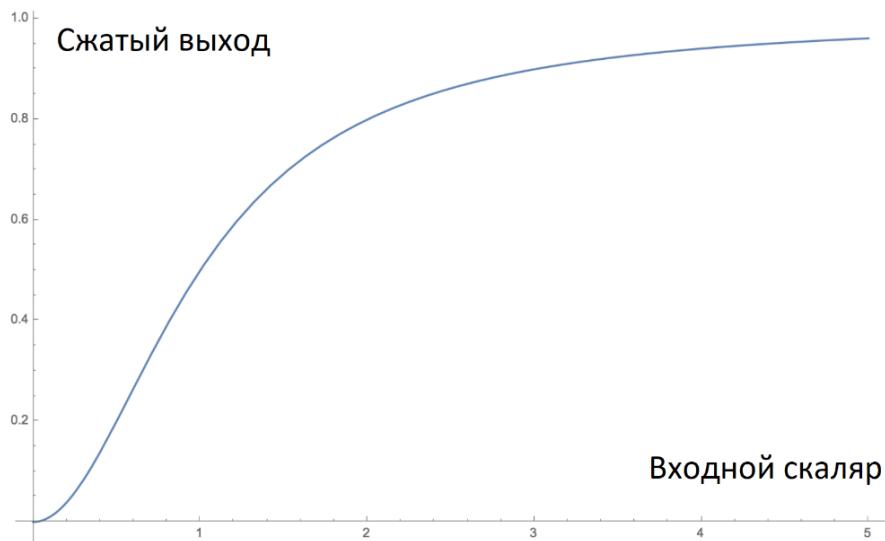


Рис. 1.14. Применение нелинейности в скалярном виде.

В реальном применении функция работает на векторах.

1.4.6 Капсультный слой. Алгоритм динамической маршрутизации

Следующий шаг — решить, какую информацию отправить в следующий слой. В традиционных сетях использовалась бы операция maxpooling. В капсультных сетях используется другой метод — направление по соглашению (routing by agreement). После получения первичных капсул необходимо решить, как отправить выходные вектора в капсулы более высокого уровня.

Сначала каждая капсула, основываясь на самой себе, делает так называемое предсказание. Путь есть 3 капсулы нижнего уровня: u_1, u_2, u_3 : они обнаруживают глаза, рот и нос, а наружная предсказываемая капсула — лицо. Каждый вектор нижнего уровня умножается на соответствующую весовую матрицу:

$$\hat{u}_{j|i} = W_{ij}u_i \rightarrow \text{вектор предсказания для капсулы } j \text{ следующего слоя на основании капсулы } i \text{ предыдущего слоя.}$$

W кодирует важные пространственные и другие отношения между объектами низкого уровня (глазами, ртом и носом) и объектами высокого уровня (лицом). Другими словами, \hat{u}_1 представляет, где должно быть лицо в соответствии с обнаруженным положением глаз, \hat{u}_2 — в соответствии с обнаруженным положением рта и \hat{u}_3 — в соответствии с положением носа. Если эти 3 предсказания объектов более низкого уровня указывают на одно и то же положение и состояние лица, то это лицо должно быть там.

Следует понимать, что учитывается вклад каждой капсулы нижнего уровня. Так в CapsNet 1152 первичные капсулы. Каждая капсула создает предсказание для каждого из 10 классов цифр. В итоге получаем список из 11520 предсказаний. Каждый вес представляет собой матрицу 16×8 . Поэтому предсказание — вектор с 16 элементами. Это число — произвольный выбор, как и 8 для первичных капсул.

Следующий шаг — определить, какие из предсказаний лучше других согласуются друг с другом. Капсула низкого уровня должна решить, как и куда отправить свой выходной вектор предсказания в капсулы более высокого уровня. Вход каждой капсулы высокого уровня s_j представляет из себя взвешенную сумму по всем векторам предсказания капсул низкого уровня:

$$s_j = \sum_i c_{ij} \hat{u}_{j|i}, \text{ где}$$

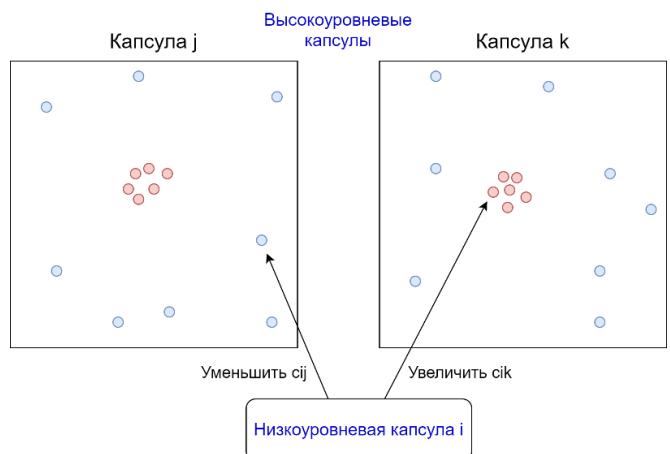
c_{ij} — коэффициент связи капсулы i нижнего уровня с капсулой j высокого уровня; каждый вес c_{ij} является неотрицательным скаляром;

для каждой капсулы нижнего уровня i сумма всех весов c_{ij} равна 1, т. е. $\sum_j c_{ij} = 1$.

Эти веса определяются итеративным алгоритмом динамической маршрутизации по соглашению. Таким образом, в некотором смысле для каждой капсулы низкого уровня веса c_{ij} определяют распределение вероятности ее выхода, принадлежащее каждой капсule более высокого уровня.

На рис. 1.15 капсула более низкого уровня определяет, в какую капсулу она внесет больший вклад. Капсулы более высокого уровня получили много входных векторов от других капсул более низкого уровня. Все эти входные данные представлены красными и синими точками. Там, где точки группируются вместе, это означает, что предсказания капсул более низкого уровня находятся близко друг к другу. Выход нижней капсул, умноженный на соответствующую матрицу W , расположен далеко от красного кластера «правильных» предсказаний в капсule J и близко к «истинному» красному кластеру предсказаний в правой капсule K. Капсula регулирует коэффициенты таким образом, что вес, соответствующий капсule K, будет высоким, а вес, соответствующий капсule J, будет низким.

На рис. 1.16 представлен данный алгоритм из оригинальной статьи.



*Рис. 1.15. Суть динамической маршрутизации.
Точки – вектора предсказаний
(в двумерном пространстве).*

Procedure 1 Routing algorithm.

```

1: procedure ROUTING( $\hat{u}_{j|i}$ ,  $r$ ,  $l$ )
2:   for all capsule  $i$  in layer  $l$  and capsule  $j$  in layer  $(l + 1)$ :  $b_{ij} \leftarrow 0$ .
3:   for  $r$  iterations do
4:     for all capsule  $i$  in layer  $l$ :  $c_i \leftarrow \text{softmax}(\mathbf{b}_i)$             $\triangleright \text{softmax}$  computes Eq. 3
5:     for all capsule  $j$  in layer  $(l + 1)$ :  $\mathbf{s}_j \leftarrow \sum_i c_{ij} \hat{\mathbf{u}}_{j|i}$ 
6:     for all capsule  $j$  in layer  $(l + 1)$ :  $\mathbf{v}_j \leftarrow \text{squash}(\mathbf{s}_j)$             $\triangleright \text{squash}$  computes Eq. 1
7:     for all capsule  $i$  in layer  $l$  and capsule  $j$  in layer  $(l + 1)$ :  $b_{ij} \leftarrow b_{ij} + \hat{\mathbf{u}}_{j|i} \cdot \mathbf{v}_j$ 
return  $\mathbf{v}_j$ 

```

Рис. 1.16. Алгоритм динамической маршрутизации.

Строка 1 говорит, что процедура принимает все капсулы низкого уровня l , их выходы $\hat{u}_{j|i}$, а также количество итераций маршрутизации r .

Строка 2. Инициализируется нулём временный коэффициент b_{ij} , который будет итеративно обновляться, и после завершения процедуры его значение будет сохранено в c_{ij} .

Строка 3. Шаги 4-7 будут повторяться r раз (кол-во итераций маршрутизации).

На практике рекомендуется использовать 3 итерации маршрутизации.

Строка 4 вычисляет значение вектора c_i , который является всеми весами маршрутизации для капсулы более низкого уровня i . Это делается для всех капсул низкого уровня. Функция softmax позаботится о том, чтобы каждый вес c_{ij} был неотрицательным числом, а их сумма равнялась единице. Функция обеспечивает вероятностную природу коэффициентов. На первой итерации значения всех коэффициентов c_{ij} будут равны, так как на второй строке все $b_{ij} = 0$. Например, есть 3 капсулы низкого уровня и 2 капсулы высокого уровня, тогда все $c_{ij} = 0.5$.

Состояние весов при инициализации алгоритма представляет собой состояние путаницы и неопределенности: капсулы низкого уровня не имеют представления, какие капсулы высокого уровня лучше подходят для их вывода.

Строка 5. Вычисляется линейная комбинация входных векторов, взвешенных по коэффициентам связи c_{ij} , определенным на предыдущем шаге. Это означает масштабирование входных векторов и их сложение вместе, что приводит к получению выходного вектора s_j для всех капсул более высокого уровня.

Строка 6. Векторы с последнего шага пропускаются через функцию сжатия, что сохраняет направление вектора, но его длина должна быть не более 1.

Строка 7. Происходит обновление весов. Этот шаг отражает суть алгоритма маршрутизации. Формула говорит, что новое значение веса равно старому значению плюс скалярное произведение текущего выхода капсулы v_j и входа в неё капсулы более низкого уровня $\hat{u}_{j|i}$. Скалярное произведение отражает сходство двух капсул (рис. 1.17).

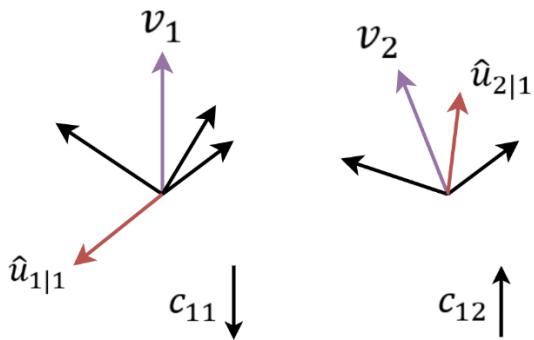


Рис. 1.17.
Изменение весов
в зависимости от сходства двух капсул.

В сети CapsNet слой, где происходит маршрутизация по соглашению, содержит 10 цифровых капсул. Каждая капсула принимает 1152 входных вектора. Каждый из них получает свою весовую матрицу 8x16, которая преобразует 8-мерное пространство в 16-мерное. Существует 1152 матрицы для каждой капсулы, а также коэффициенты 1152 с и 1152 b, используемые в динамической маршрутизации. Получается $1152 \times 8 \times 16 + 1152 + 1152 = 149760$ обучаемых параметров на капсулу. Умножаем на 10, чтобы получить конечное число параметров для этого слоя.

1.4.7 Функция потерь

Во время обучения для каждого обучающего примера будет рассчитано значение потерь для каждого из конечных векторов по формуле:

$$L_k = T_k \max(0, m^+ - \|v_k\|)^2 + \lambda(1 - T_k) \max(0, \|v_k\| - m^-)^2$$

Затем эти значения будут суммированы для расчета конечного убытка. Поскольку речь идет о контролируемом обучении, каждый пример будет иметь правильную метку. Например, в CapsNet для каждого примера это кодированный вектор с 9 нулями и 1 единицей в правильном положении. Тогда $T_k = 1$, если метка соответствует классу, и 0 иначе.

Предположим, что $T_k = 1$. Тогда вычисляется только первое слагаемое функции потерь. Длина выходной капсуллы вычитается из m^+ , который фиксируется на 0.9. Полученное значение сохраняется только в том случае, когда оно больше нуля и ставится в квадрат. Другими словами, потеря будет равна нулю, если капсулла предсказывает правильную метку с вероятностью больше 0.9, и она будет ненулевой, если вероятность меньше 0.9.

Если выходная капсулла определяет неправильную метку (не свой класс), будет оцениваться только второе слагаемое. m^- фиксируется на 0.1, и в этом случае потеря равна нулю, если несоответствующая капсулла предсказывает неправильную метку с вероятностью менее 0.1. Здесь включен лямбда-коэффициент для численной устойчивости при обучении (его значение фиксируется на уровне 0.5).

1.4.8 Реконструкция

Описанные выше слои CapsNet принимают в качестве входного сигнала изображение и учатся кодировать его в 16-мерный вектор параметров. Здесь разработчика волнует только точность предсказания (длина вектора).

Интересной особенностью капсулой сети является то, что выходной вектор может быть использован для восстановления, или реконструкции, входного изображения. Вторая часть архитектуры (decoder) учится декодировать вектор в изображение. Функция потерь в этом случае является евклидовым расстоянием между восстановленным изображением и входным изображением. Декодер изучает особенности капсул, которые полезны для реконструкции. Для этого используются полно связные слои (Fully Connected, FC). Декодер из оригинальной статьи представлен на рис. 1.18. Примеры реконструированных изображений можно увидеть на рис. 1.19.

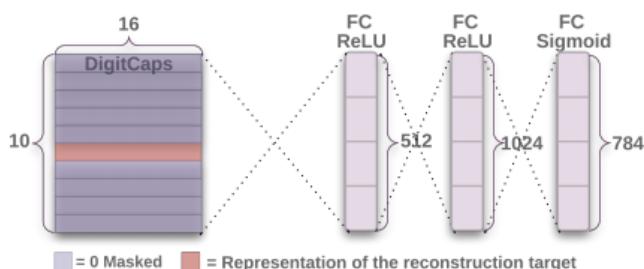


Рис. 1.18.
Декодер сети CapsNet
из оригинальной статьи.

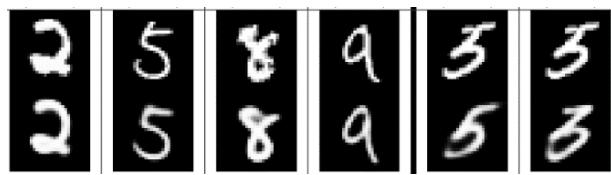


Рис. 1.19.
Верхняя строка –
исходные изображения.
Нижняя строка –
реконструкция (CapsNet).

1.4.9 Особенности

Для сверточных сетей нет такого понимания пространства, как в капсуловых сетях, где сохраняются пространственные и иерархические отношения между частями объекта. Еще одно преимущество капсулного подхода заключается в том, что сеть способна научиться достигать самых современных результатов, используя только часть обучающих данных, которые использует сверточная сеть. В этом смысле теория капсул гораздо ближе к тому, что делает человеческий мозг. Чтобы научиться различать объекты, мозгу нужно увидеть всего лишь пару десятков примеров, а не десятки тысяч.

1.5 Методы детектирования с помощью глубинного обучения

Проблему обнаружение объектов с помощью глубинного обучения можно рассматривать как задачу классификации или как задачу регрессии.

1.5.1 Подход на основе классификации

Изображение делится на участки, содержащие объект, отличный от фона. Затем каждый участок пропускается через классификатор, как самостоятельное изображение, чтобы определить класс объекта на этом участке. Так в 2013 г. была представлена первая версия алгоритма R-CNN (Region-based Convolutional Neural Networks) на основе методов сегментации изображения и сверточной нейронной сети. Модель основана на подходе предположений регионов. Исходная реализация базируется на использовании специальных алгоритмов предобработки — алгоритмов region-proposal-function, обеспечивающих предложение так называемых областей внимания, в которых потенциально могут находиться интересующие нас объекты. Затем эти предположения обрабатываются, и классифицируется объект на каждом участке. В последствии, эти идеи развились в Fast R-CNN и Faster R-CNN.

Алгоритм Faster R-CNN — самый точный в своем классе, но для достижения высокой точности приходится жертвовать производительностью, что в свою очередь приводит к невозможности работы в режиме реального времени.

1.5.2 Подход на основе регрессии

В подходе, использующем регрессию, изображение целиком пропускается через свёрточную нейронную сеть, чтобы произвести обрамляющие окна и вероятности классов для объектов на изображении. В конце 2015 г. был представлен алгоритм YOLO (You Only Look Once) [6], который позволяет производить быструю обработку изображений для работы в реальном времени. Первая версия имела низкую точность на больших и маленьких объектах, однако в улучшенной версии YOLOv2 алгоритм достигает точности Faster R-CNN, не теряя в скорости. На данный момент существует уже третья версия YOLOv3. Также существует модель SSD (Single Shot Detector), похожая на YOLO. Делается только один снимок, чтобы обнаружить несколько объектов на изображении.

1.5.3 Выбор метода

Из современных моделей глубокого обучения для решения задачи детектирования объектов в режиме реального времени можно выделить следующих фаворитов: YOLO, RetinaNet и SSD (рис. 1.20). Начиная со второй модификации, YOLO превосходит как в скорости, так и в точности SSD при определенных модификациях сети, а третья версия показывает лучшие результаты по сравнению с остальными методами, что является решающим фактором для выбора метода решения задачи детектирования дорожных знаков.

Таким образом, выбор сделан в пользу YOLOv3.

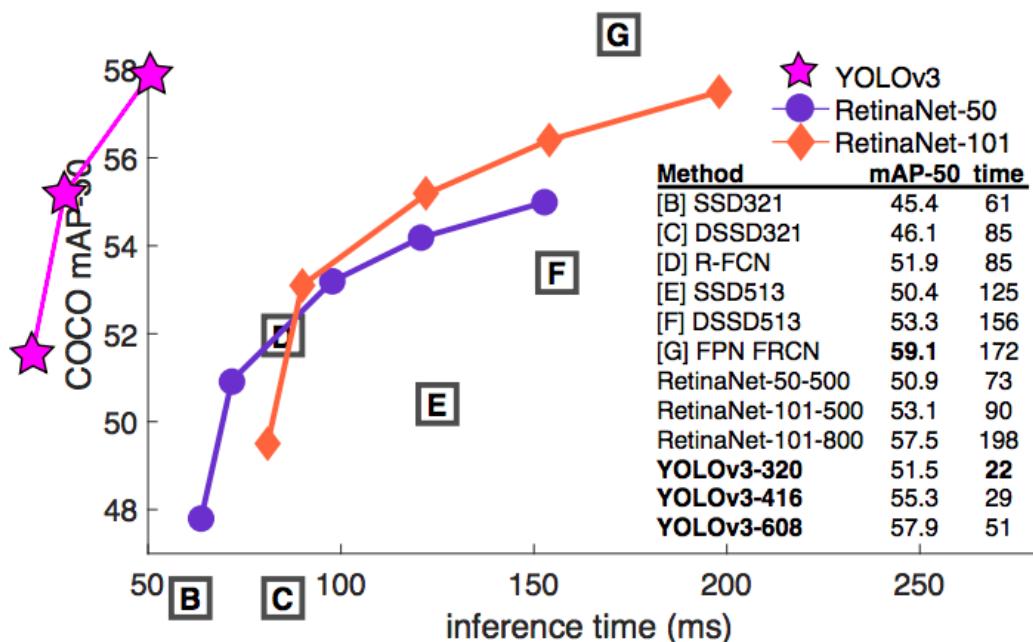


Рис. 1.20. Сравнение моделей глубокого обучения на датасете COCO.

1.5.4 Описание модели YOLOv3

Главная особенность YOLO состоит в том, что большинство систем применяют CNN несколько раз к разным регионам изображения, в YOLO CNN применяется один раз сразу ко всему изображению.

В статье [7] про YOLOv3 авторы представляют глубокую архитектуру сети для извлечения признаков под названием Darknet-53. Сеть использует только сверточные слои, что делает её полностью сверточной сетью (FCN). Как следует из названия, она содержит 53 сверточных слоя, каждый из которых сопровождается активацией ReLU (рис.1.21).

Подвыборочный слой не используется, а сверточный слой с шагом 2 применяется для уменьшения карт признаков.

На практике, при обучении, лучше придерживаться постоянного входного размера изображений выборки. Если обрабатывать изображения в пакетном режиме (batch) (изображения в пакетном режиме могут обрабатываться параллельно графическим процессором, что приводит к увеличению скорости), необходимо иметь все изображения фиксированной высоты и ширины для объединения нескольких изображений в большой пакет. Для YOLO неважен размер входного изображения. Каким бы ни был размер входа, он будет принудительно изменен до предопределенного (например, до 416x416).

Далее сеть уменьшает входной размер в зависимости от множителя уменьшения сети. Например, если он равен 32, то входное изображение 416x416 даст выход 13x13. Множитель на любом слое в сети равен коэффициенту, показывающему во сколько раз выход слоя меньше, чем вход в сеть.

По умолчанию входной сигнал представляет собой пакет изображений вида (m , 416, 416, 3), где m — количество изображений в пакете, 3 — количество каналов (RGB). Выходные данные представляют собой список обрамляющих рамок вместе с вероятностями распознанных классов.

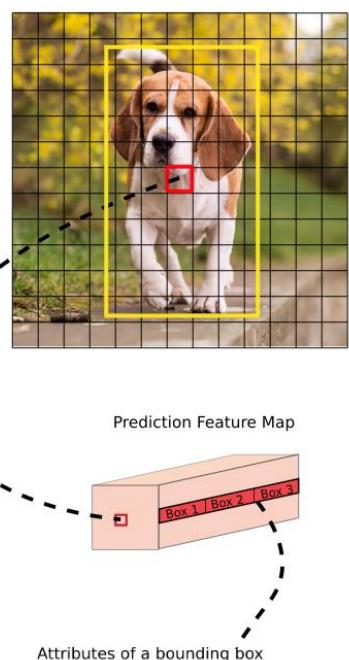
Type	Filters	Size	Output
Convolutional	32	3×3	256×256
Convolutional	64	$3 \times 3 / 2$	128×128
Convolutional	32	1×1	
Convolutional	64	3×3	
Residual			128×128
Convolutional	128	$3 \times 3 / 2$	64×64
Convolutional	64	1×1	
Convolutional	128	3×3	
Residual			64×64
Convolutional	256	$3 \times 3 / 2$	32×32
Convolutional	128	1×1	
Convolutional	256	3×3	
Residual			32×32
Convolutional	512	$3 \times 3 / 2$	16×16
Convolutional	256	1×1	
Convolutional	512	3×3	
Residual			16×16
Convolutional	1024	$3 \times 3 / 2$	8×8
Convolutional	512	1×1	
Convolutional	1024	3×3	
Residual			8×8
Avgpool			Global
Connected			1000
Softmax			

Рис. 1.21. Darknet-53.

Как и во всех объектных детекторах, особенности, полученные свёрточным слоями, передаются в блок принятия решения (классификатор/ регрессор), который делает прогноз обнаружения (координаты ограничивающих прямоугольников, метка класса). В YOLO прогнозирование выполняется с использованием свёрточного слоя, который использует свёртки 1×1 . После применения к карте признаков получится карта предсказаний такого же размера.

На выходе нейросеть дает тензор (например, $13 \times 13 \times \langle\text{параметры}\rangle$), который содержит всю информацию, необходимую для нанесения на изображение обрамляющих окон. На изображение наносится соответствующая сетка. Каждая ячейка может предсказать фиксированное количество ограничивающих рамок (базовых прямоугольников), каждая из которой может специализироваться на обнаружении объекта определенного типа (если в ячейке более одного объекта). Под базовыми прямоугольниками понимаются ряд определенных соотношений ширины и высоты, которые корректируются сетью для получения результирующего описывающего прямоугольника объекта. YOLOv3 прогнозирует 3 прямоугольника для каждой ячейки.

Каждая ячейка имеет ряд параметров, являющихся выходными данными сети (рис.1.22). Данных параметров достаточно для определения описывающего прямоугольника объекта, класса объекта и параметра “уверенности” сети в принятом решении. Размер выходного пространства определяется параметрами:



- количество ячеек по ширине (W) и высоте (H)
- количество базовых прямоугольников (B);
- количество предсказываемых классов (C).

Формула для вычисления выходных параметров выглядит следующим образом: $W * H * B * (4 + 1 + C)$.

Каждый прямоугольник характеризуется показателями:

- (x, y) — координаты центра соответствующей ячейки;
- w — ширина окна относительно всего изображения;
- h — высота окна относительно всего изображения.
- вероятность, что окно правильно детектировало объект.
- вероятности того, что внутри ячейки сетки лежит объект конкретного класса

t_x	t_y	t_w	t_h	p_o	p_1	p_2	\dots	p_c			
Box Co-ordinates				Objectness Score				Class Scores			

× B Рис. 1.22. Представление выходной сетки YOLO.

Таким образом, каждая ячейка предсказывает В ограничивающих окон и доверительные оценки (confidence scores) для них. На рис. 1.23 еще один пример.

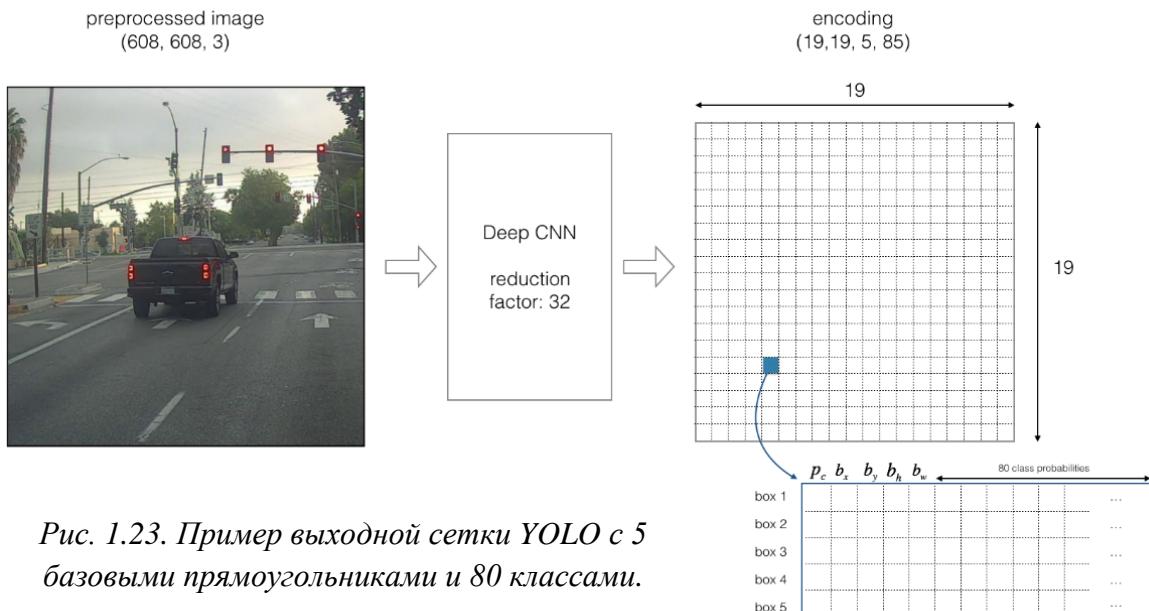


Рис. 1.23. Пример выходной сетки YOLO с 5 базовыми прямоугольниками и 80 классами.

Перед финальным нанесением на изображение прямоугольников применяется алгоритм Non-maximal suppression, который оставит наиболее точное из пересекающихся окон. NMS представляет собой алгоритм исключения повторений предсказаний на основе определенного признака, объединяющего несколько предсказаний. Чтобы один и тот же объект не определялся многократно, используется коэффициент перекрытия окон (IoU, Intersection over Union). Сеть YOLO во всех версиях генерирует большое количество предсказаний, которые объединяются в группы на основе признака пересечения IOU. Далее из группы остается единственное предсказание, которое имеет наибольший показатель уверенности, а остальные предсказания удаляются.

Следует отметить версию YOLOv3-tiny, «обрезанную» версию архитектуры YOLOv3. Она состоит из меньшего количества слоев, несколько хуже предсказывает небольшие объекты и предназначена для небольших выборок. Из-за своего строения веса сети занимают небольшой объем памяти (~35 Мб), и она выдает более высокий FPS. Такая архитектура предпочтительна для использования на мобильном устройстве.

1.6 Выводы

В разделе были выдвинуты основные требования к системе распознавания.

Рассмотрены существующие разработки.

Одним из самых эффективных способов для задачи классификации на данный момент являются нейронные сети. Рассмотрены особенности их работы на примере разных архитектур. В качестве классификатора используется капсулльная нейронная сеть как самая перспективная.

Рассмотрены наиболее распространённые методы детектирования, их недостатки. В результате анализа методов была выбрана модель YOLO третьей версии, так как она показывает лучшие результаты точности и одновременно способна работать в режиме реального времени, что является важным критерием для системы распознавания.

2. Конструкторский раздел

1.1 Подготовка данных для обучения

В открытом доступе представлены несколько баз дорожных знаков. Среди них можно выделить:

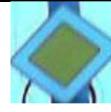
- GTSRB (German Traffic Sign Recognition Benchmark);
- GTSD (German Traffic Sign Detection Benchmark);
- STS (Swedish Traffic Signs);
- BTSD (Belgium Traffic Sign Dataset);
- LISA Traffic Sign Dataset (The Laboratory for Intelligent and Safe Automobiles);
- RTSD (Russian Traffic Sign Dataset).

Несмотря на единообразие, в дорожных знаках разных стран существуют значительные отличия. Они сводятся к графическим различиям (разные типы стрелок, различные шрифты), различиям в значениях (километры в час, мили в час), использованию различных цветовых схем, местных языков.

В данной работе используется российская база автодорожных знаков, представленная в 2016 году [8]. Она разделена на подвыборки, которые содержат группы классов «предписания» (синие круги), «запреты» (красные треугольники), «ограничения» (круги с красной рамкой), «главная дорога» (жёлтый ромб), «сервис» (прямоугольники с синей рамкой), «особые предписания» (синие прямоугольники).

Размеры выборок для детектирования RTSD-D1, RTSD-D2, RTSD-D3 показаны в таблице 2.1.

Таблица 2.1 Размеры выборок для детектирования российской базы RTSD.

						
RTSD-D1 обучение (3821 кадров) тестирование (1274 кадра)	1054 396	1594 578	1842 605			
RTSD-D2 обучение (4786 кадров) тестирование (1596 кадров)	1033 455	1617 591	1848 626	1268 329		
RTSD-D3 обучение (9065 кадров) тестирование (3022 кадра)	1164 501	1800 651	2099 684	1678 431	1235 474	6843 2085

Для классификации отдельно формировались две выборки RTSD-R1 и RTSD-R3. Они совпадают по группам классов с выборками D1 и D3 и содержат вырезанные изображения физических знаков. Размеры представлены в табл. 2.2.

Таблица 2.2. Размеры выборок для классификации российской базы RTSD.

	RTSD-R1	RTSD-R3
обучение	25432	70687
тестирование	7551	22967

Российская база превосходит другие публичные базы по количеству кадров, классов знаков, физических знаков и их изображений. Так, выборка RTSD-D1 по классам знаков аналогична базе GTSDDB. База содержит кадры с различными погодными условиями, освещением и временами года, что хорошо сказывается на её репрезентативности. К недостаткам можно отнести не всегда качественные изображения. Это частично связано с тем, что кадры получены с камер, снимавших изнутри автомобиля, а не снаружи, как в других базах.

Для решения задачи детектирования и классификации знаков было принято решение использовать выборки RTSD-D1 и RTSD-R1 соответственно.

1.2 Валидация данных

Во время обучения нейронная сеть «видит» только тренировочный набор данных и принимает решения, каким образом изменить значения внутренних параметров — весов и смещений. Затем, оценивают работу сети на тех данных, которых она никогда не видела (тестовом наборе). После обучения можно получить высокую точность на тренировочном наборе данных, к примеру 97%, и всего лишь 92% на тестовом наборе данных. Это происходит потому, что модель переобучается. Другими словами, сеть начинает запоминать тренировочный набор данных, а её способность к обобщению ухудшается.

Для избежания подобной проблемы часто используется отдельный набор данных для валидации. Данные из этого набора никогда не используются моделью для корректировки значений внутренних параметров. Идея состоит в том, что после каждой обучающей итерации проверяется состояние модели, вычисляя значение функции потерь на обучающем наборе и на валидационном наборе.

Результаты работы модели на валидационном наборе данных сообщают, насколько хорошо модель научилась обобщать полученные данные и применять это обобщение на новом наборе данных.

Один из способов избежать переобучение является изучение графика значений функции потерь на тренировочном и валидационном наборах данных на протяжении всех обучающих итераций (рис. 2.1).

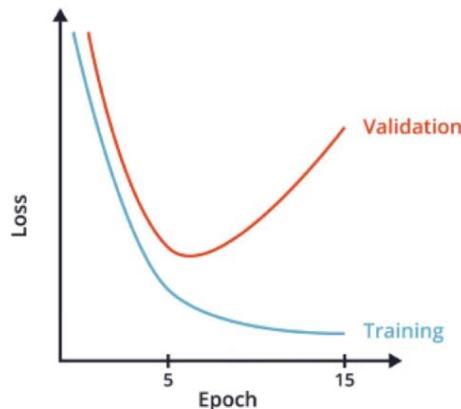


Рис. 2.1. Зависимость функции потерь от количества эпох на тренировочном и валидационном наборе.

После определённой итерации значение функции потерь на валидационном наборе данных начинает возрастать, в то время как значение функции потерь на тренировочном наборе данных продолжает снижаться. Это и есть индикатор переобученности нейронной сети. Это позволяет определить количество обучающих итераций, которые необходимо провести, чтобы сеть была точной и не переобучилась.

Подобный подход также полезен в том случае, если есть выбор из нескольких моделей нейронных сетей. Можно сравнить их точность, используя набор данных для валидации.

В идеальном случае должно быть 3 уникальных набора данных: для обучения, тестирования и валидации. Часто из-за небольшого размера всей выборки наборы для тестирования и валидации приравниваются, как в данной работе. При этом обычно 15-30% занимает тестовая выборка.

1.3 Используемые метрики

Для оценки моделей нейронных сетей используется тестовая выборка. Для задачи классификации в простейшем случае такой метрикой является доля экземпляров всей выборки, по которым классификатор принял правильное решение (Accuracy):

$$Accuracy = \frac{P}{N},$$

P – количество правильно классифицированных экземпляров,

N – размер обучающей выборки.

Метрика предполагает, что все экземпляры выборки имеют одинаковый вес, что может быть некорректно в случае, если распределение в обучающей выборке сильно смещено в сторону какого-то одного или нескольких классов. Тогда у классификатора есть больше информации по этим классам, и в рамках этих классов он будет принимать более адекватные решения. Поэтому необходимо проводить обучение на сбалансированном наборе данных.

Перед описанием других метрик необходимо ввести важную концепцию в терминах ошибок классификации. Допустим, есть два класса и алгоритм, предсказывающий принадлежность объекта одному из классов, тогда матрица ошибок классификации будет выглядеть следующим образом (табл.2.3):

Таблица 2.3. Матрица ошибок классификации.

\hat{y} — ответ алгоритма, y — истинная метка класса.

	$y = 1$	$y = 0$
$\hat{y} = 1$	True Positive (TP)	False Positive (FP)
$\hat{y} = 0$	False Negative (FN)	True Negative (TN)

Таким образом, ошибки классификации бывают двух видов: FN и FP.

Для оценки качества работы алгоритма на каждом из классов по отдельности вводятся метрики Precision (точность) и Recall (полнота). Точность системы в пределах класса — это доля примеров, действительно принадлежащих данному классу, относительно экземпляров, которые система отнесла к этому классу. Полнота — это доля правильно найденных экземпляров, принадлежащих классу относительно всех примеров этого класса в тестовой выборке.

$$Precision = \frac{TP}{TP + FP},$$

$$Recall = \frac{TP}{TP + FN}.$$

К примеру, есть тестовая выборка, в которой 10 сообщений, из них 4 — спам. Классификатор пометил 2 сообщения как спам, причем одно действительно является спамом, а второе было помечено, как нормальное. Имеем 1 истинно-положительное решение, 1 ложно-положительное и 3 ложно-отрицательных. Тогда для класса «спам» точность составляет $\frac{1}{2}$ (50%), а полнота равна $\frac{1}{4}$ (25%). Recall отображает способность алгоритма обнаруживать данный класс вообще, а Precision — способность отличать этот класс от других классов.

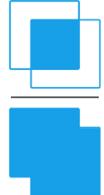
В задаче детектирования существует две различные задачи для измерения:

- определение того, существует ли объект на изображении (классификация)
- определение местоположения объекта (локализация, задача регрессии).

Чтобы была возможность оценить правильность заключения объекта в обрамляющее окно используется метрика IoU (Intersection over Union). Она измеряет перекрытие между двумя областями (рис.2.2):

$$IoU = \frac{\text{Площадь пересечения}}{\text{Площадь объединения}}.$$

Рис. 2.2. IoU.



Типичное пороговое значение, указывающее на правильное обнаружение, составляет $IoU > 50\%$.

Также необходимо оценить риск неправильной классификации. Важно отметить, что существует обратная зависимость между точностью Precision и полнотой Recall. Эти показатели зависят от качества модели и порога оценки модели, который заранее устанавливается. Для определенного класса строится кривая Precision/Recall из выходных данных (рис. 2.3).

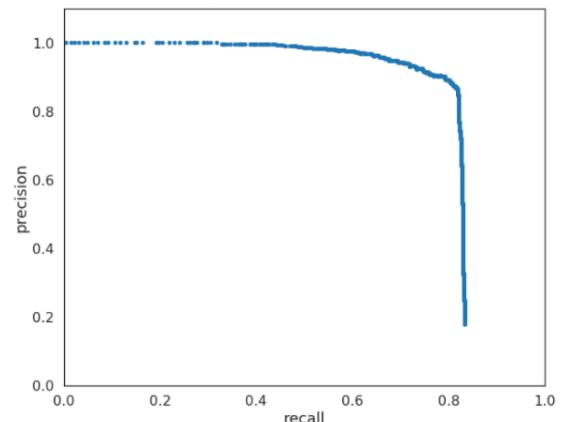


Рис. 2.3.
Пример кривой Precision-Recall.

Вводится понятие средней точности (Average Precision, AP). Общее определение можно сформулировать как поиск области под кривой:

$$AP = \int_0^1 p(r)dr, \text{ где } p - Precision, r - Recall.$$

Интерполированная AP вычисляется как среднее значение точности по всем значениям полноты, т.е.:

$$AP = \frac{1}{11} \sum_{Recall_i} Precision_i, \text{ где } Recall_i = [0, 0.1, 0.2, \dots, 1.0].$$

Главная метрика называется mean Average Precision (mAP). Она вычисляется путем взятия среднего AP по всем классам и по всем порогам IoU. mAP вычисляется по всему датасету. Данную величину нелегко интерпретировать, т.к. она учитывает влияние сразу нескольких показателей. Кроме того, в зависимости от того, как распределены классы в обучающих данных, средние значения точности могут сильно варьироваться для некоторых классов. Важно понимать, что при детектировании одного класса mAP приравнивается к AP.

1.4 Предлагаемое решение

Алгоритм работы системы предлагается реализовать по схеме, показанной на рисунке 2.4

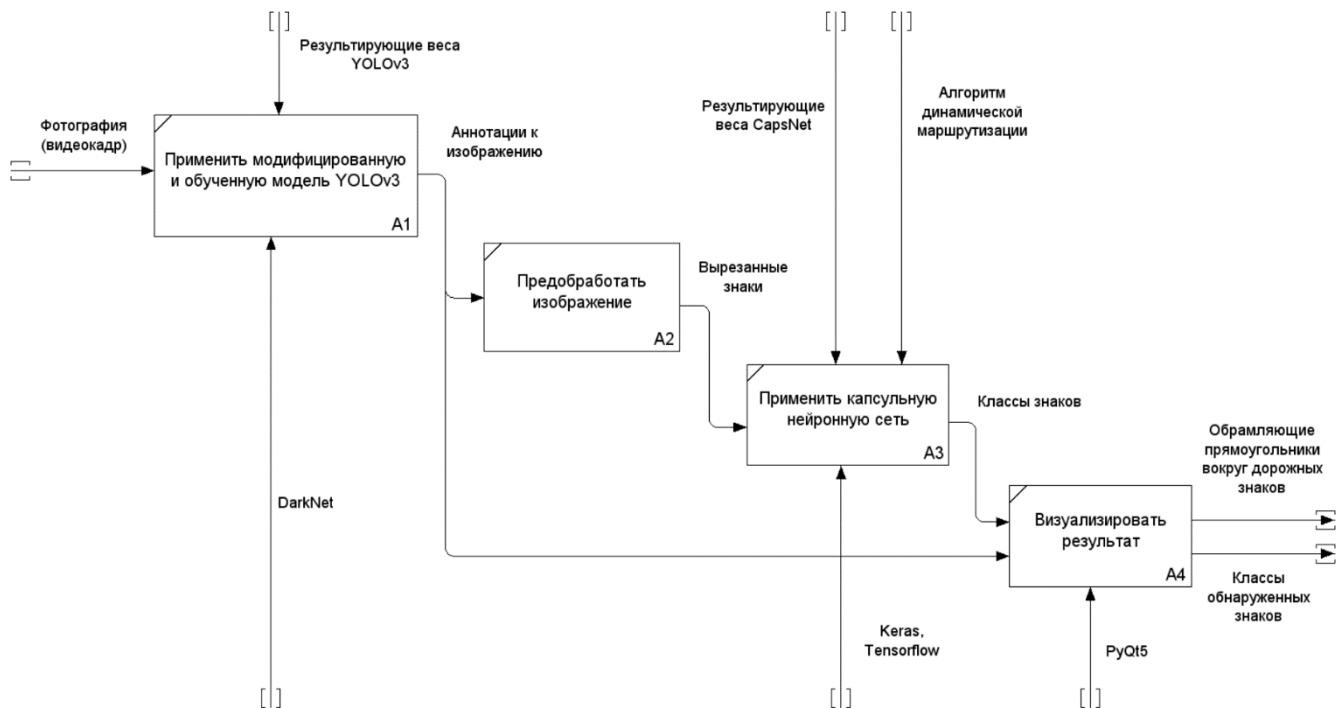


Рис. 2.4. Диаграмма работы предлагаемой системы распознавания.

Первым делом необходимо модифицировать и обучить модель YOLOv3 для детектирования знаков, а также построить и обучить модель капсулойной нейронной сети для классификации.

В результате применения YOLO к входному изображению будут получены аннотации, содержащие координаты обрамляющих рамок. Далее задетектированные дорожные знаки вырезаются и предварительно преобразовываются в формат, необходимый для классификатора (CapsNet). Капсулальная сеть классифицирует обнаруженные области, в которых есть знаки. После чего результаты визуализируются.

1.5 Выводы

Таким образом, в разделе был проведен анализ доступных баз дорожных знаков, по результатам которого выбран российский датасет RTSD. Определены выборки, которые используются для обучения детектированию и классификации. Описано, как будет контролироваться обучение с помощью валидации данных. Приведены используемые метрики для оценки качества моделей. Также рассмотрена работа предлагаемой системы распознавания.

3. Технологический раздел

3.1 Средства разработки

Для обработки данных, разработки и обучения капсулной нейронной сети использовался следующий технологический стек:

- Язык программирования Python. Вокруг него сложилось самое большое сообщество по нейросетям. На Python удобно решать вопросы подготовки обучающих данных.
- Открытая программная библиотека для машинного обучения TensorFlow, предназначенная для проектирования, создания моделей глубокого обучения.
- Keras — высокоуровневый API нейронных сетей способный работать поверх TensorFlow. Представляет собой надстройку над фреймворком. Библиотека спроектирована так, чтобы быть компактной, модульной и расширяемой.
- Прочие библиотеки (matplotlib, pandas, opencv, imgaug, PyQt5 и др.).

Следует отметить, что реализация алгоритма YOLO, основанная на Darknet, написана на C и CUDA. Она поддерживает CPU и GPU вычисления.

Вычисления происходили с использованием аппаратного ускорения на GPU GTX 970 4Gb.

Использовалась ОС Windows 10.

3.2 Предварительная обработка данных для задачи классификации

Используемый датасет для классификации состоит из 25432 обучающих и 7551 тестовых изображений (66 классов). Все изображения были сгруппированы в директории по классам. В результате анализа выборки сделан вывод, что она имеет два существенных недостатка.

1. Плохое качество изображений. Есть много вырезанных знаков, которые не распознает даже человек. Нейросеть не сможет научиться извлекать правила и паттерны из таких изображений (рис. 3.1).

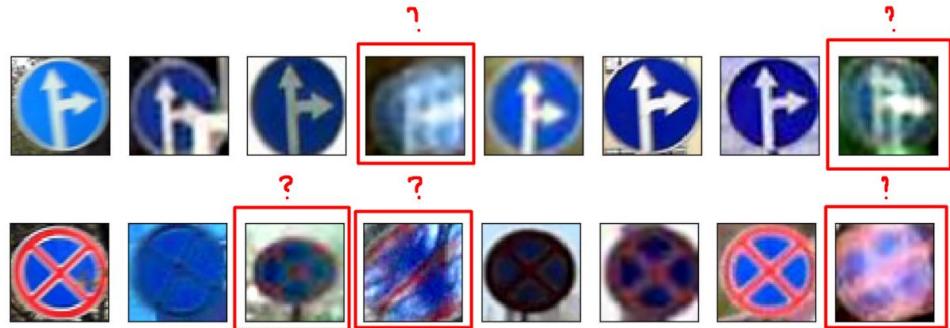


Рис.3.1. Пример изображений плохого качества. 8 случайных изобр. для 2 классов.

2. Сильный разброс по количеству изображений в каждом классе. Некоторые классы имеют 10-30 изображений, другие — больше 2000 (рис. 3.2). Такое неравномерное распределение приводит к «смещению» модели во время обучения в сторону определенных типов изображений.

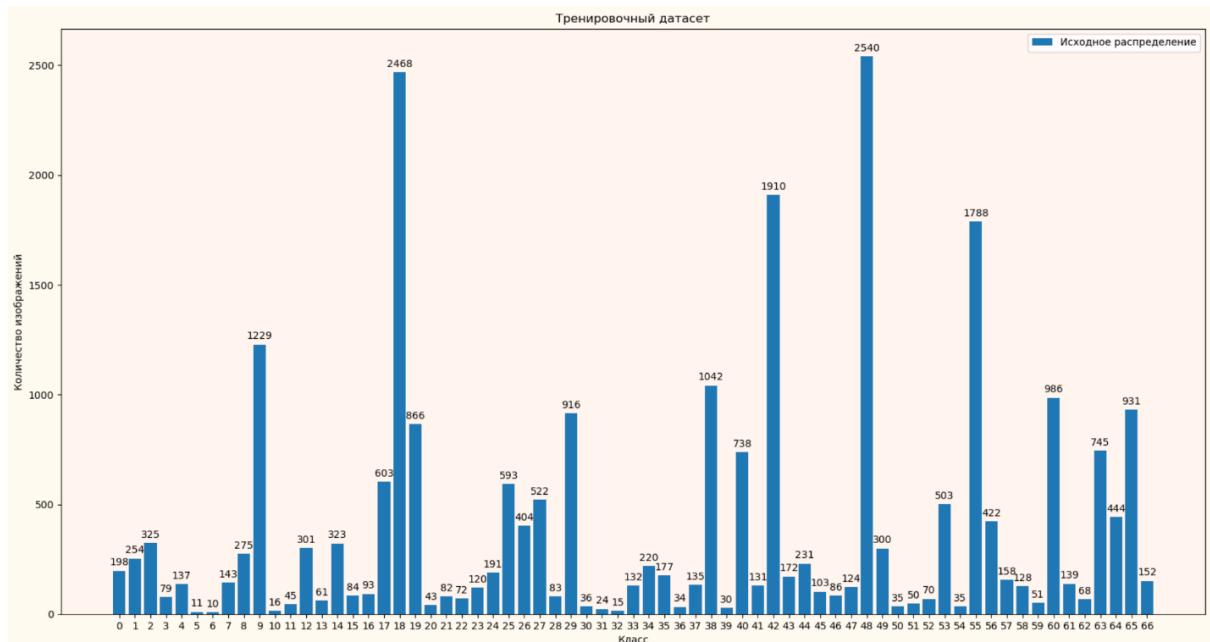


Рис.3.2. Гистограмма распределения изображений по классам (тренировочный датасет).

Чтобы решить первую проблему, была проведена небольшая программная «чистка» выборки от сильно размытых изображений. В качестве меры качества использовался результат оператора Лапласиана, который выделяет области изображения, содержащие быстрые изменения интенсивностей. Данный метод использует 1 канал изображения (оттенки серого) и сворачивает его ядром 3x3. Затем берется дисперсия, и, если она падает ниже заранее определенного порога, то изображение считается размытым. Данный порог подбирался экспериментальным путем и составил значение, равное 150 (рис. 3.3).



Рис.3.3. Значения дисперсий оператора Лапласиана для выявления изображений плохого качества. Порог=150.

На рис 3.4 показано сравнение размеров выборки до и после удаления сильно размытых изображений. Аналогичная процедура сделана и для тестовой выборки.

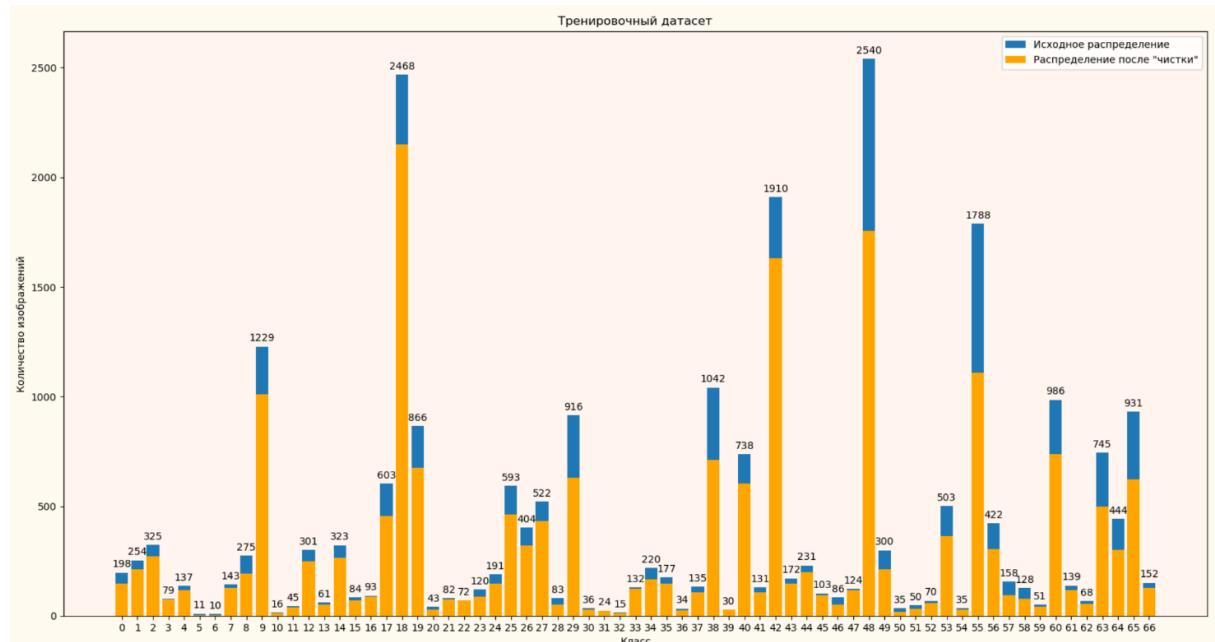


Рис.3.4. Сравнение размеров тренировочных данных до и после удаления сильно размытых изображений с помощью оператора Лапласиана с порогом 150.

Вторая проблема проявляется меньше если взять для обучения не все 66 классов, а только те, которые содержат наибольшее количество изображений. Это допущение также уменьшает нагрузку на вычислительные ресурсы. Таким образом, был взят топ 25 классов по количеству изображений. Полученная выборка была также осмотрена вручную для удаления неразличимых изображений, которые не смог выявить оператор Лапласиана (рис. 3.5).

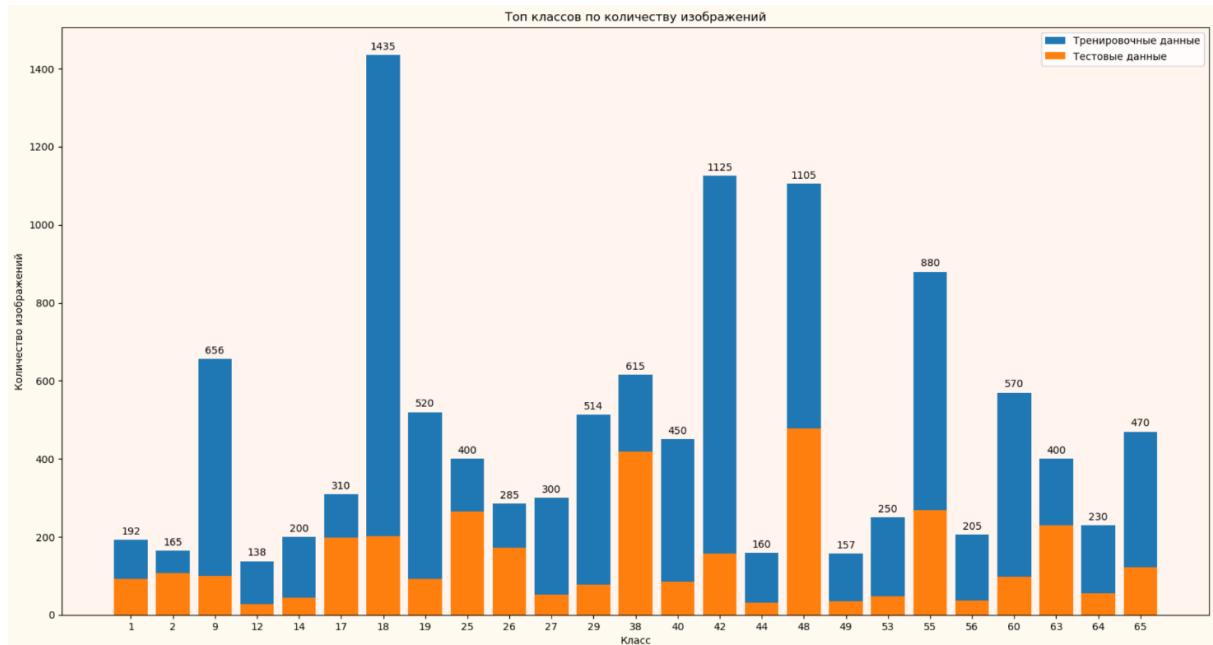


Рис.3.5. Топ 25 классов по количеству изображений (тренировочные и тестовые данные).

Следует отметить, что тренировочная и тестовая выборки обрабатывались независимо. Изображения не смешивались и не перемещались из одной выборки в другую, чтобы сохранить уникальность тестовых изображений, предполагаемую автором базы российских дорожных знаков.

Кроме того, чтобы полностью устранить второй недостаток и повысить репрезентативность выборки, была применена **аугментация** данных. Это метод, используемый для искусственного создания вариаций изображений, чтобы расширить существующий набор данных. Это делается путем применения различных методов, таких как масштабирование изображения, поворот на несколько градусов, сдвиг, обрезка, наложение шумов, изменение яркости и т.д. Это помогает повысить качество модели за счет лучшего обобщения. Стратегии аугментации для увеличения размера набора обучающих данных необходимо применять после тщательного изучения проблемной области.

Аугментация может проходить на этапе предварительной обработки (оффлайн), как в данной работе, а также во время обучения (онлайн), где модель каждую эпоху просматривает похожие, но разные изображения.

Для аугментации была использована специальная библиотека *imgaug*. Для увеличения выборки дорожных знаков использовались 4 вида аугментации: изменение яркости (1 из 2 видов), поворот, шум Гаусса, эффект дождя (рис. 3.6).

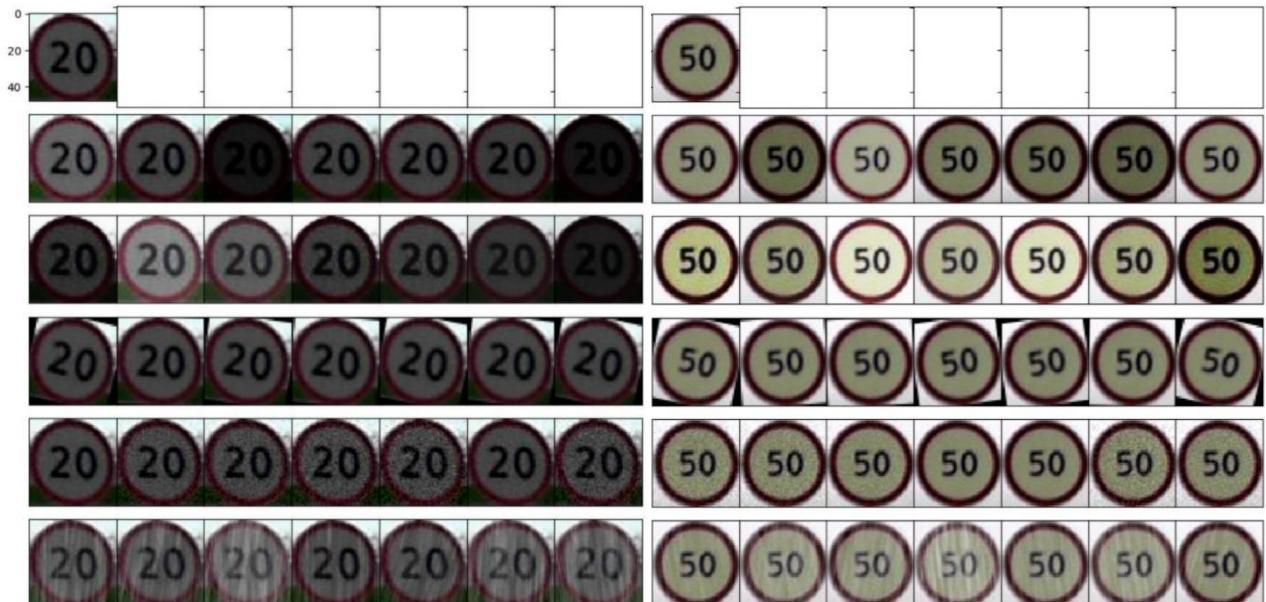


Рис.3.6. Примеры аугментации изображений.

1 строка – исходное изображение

4 строка – поворот

2 строка – изменение яркости (вид 1, *GammaContrast*)

5 строка – наложение шума (ряби)

3 строка – изменение яркости (вид 2, *SigmoidContrast*)

6 строка – эффект дождя

Первые 3 способа накладывались друг на друга в случайном порядке и в случайном количестве, т. е. использовались как по одному, так и одновременно. 93% недостающего количества изображений получены таким методом. Было принято решение не смешивать ни с чем 4 способ, эффект дождя (для 7%).

Параметры аугментации представлены в листинге 3.1.

Листинг 3.1. Метод получения аугментаторов.

```

@staticmethod
def _get_augmenters():
    """Get the augmenters used to equalize the class."""
    aug_1_1 = iaa.GammaContrast(gamma=(0.5, 1.8))
    aug_1_2 = iaa.SigmoidContrast(gain=(5, 9), cutoff=(0.25, 0.55))
    aug_1 = iaa.SomeOf(1, [aug_1_1, aug_1_2])
    aug_2 = iaa.Affine(rotate=(-14, 14))
    aug_3 = iaa.AdditiveGaussianNoise(scale=(0, 15))
    aug_4 = iaa.Rain(speed=(0.6, 0.8))

    aug_some_of = iaa.SomeOf((1, None), [aug_1, aug_2, aug_3])
    aug_rain = aug_4
    return aug_some_of, aug_rain

```

В результате аугментации каждый класс изображений имеет не менее 800 изображений в тренировочной выборке и не менее 180 в тестовой (рис.3.7).

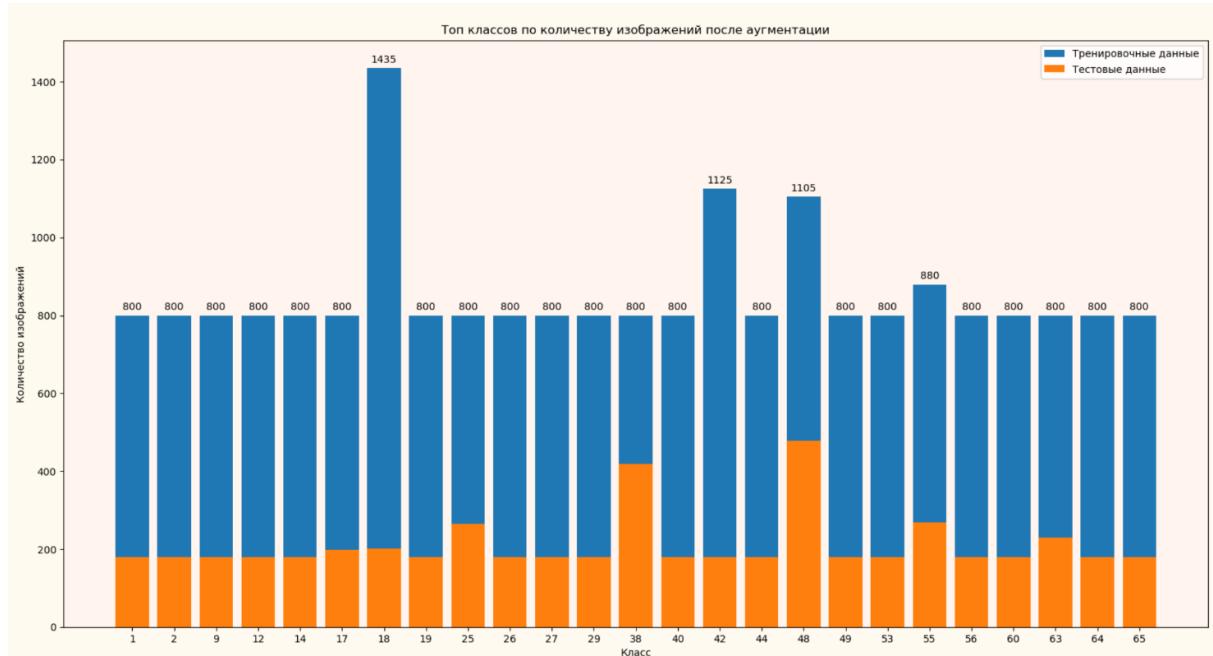


Рис.3.7. Результат аугментации.

Для окончательной подготовки к обучению выборка из всех классов изображений была случайным образом перемешана, а каждый пиксель был нормализован, то есть приведен от диапазона [0-255] к диапазону [0-1].

3.3 Построение и обучение капсульной нейронной сети

На основе шаблона, предложенного Хинтоном, была построена модель капсульной нейронной сети:

INPUT → [CONV → RELU] → DROPOUT → PRIMARYCAPS → CLASSIFICATIONCAPS.
 INPUT – входной слой;
 CONV – сверточный слой;
 RELU – активация ReLU;
 DROPOUT – слой исключения;
 PRIMARYCAPS – первичный капсульный слой;
 CLASSIFICATIONCAPS – вторичный слой капсул.

В последнем слое (вторичных капсул) находится 25 капсул. Они выполняют роль высокоуровневого классификатора, и их количество должно быть равным количеству классов, которые были подготовлены ранее. В задаче нет смысла использовать слой реконструкции, кроме того, это сильно замедлит процесс обучения. Этот слой не используется.

Одна из проблем, с которой можно столкнуться во время обучения нейронной сети — огромные значения в одной части нейронной сети и маленькие значения в другой части. В результате нейроны с большими весами играют большую роль в процессе обучения, а нейроны с меньшими весами перестают быть значимыми. Один из способов избежать подобного — использовать произвольное отключение нейронов (слой DROPOUT). Отключение одних нейронов позволяет активно задействовать другие нейроны в обучении. В процессе обучающих итераций произвольным образом отключаются некоторые нейроны. Тренируя сеть таким образом можно избежать переобучения. Сеть становится более устойчивой, потому что при таком подходе она не может полагаться абсолютно на все нейроны для решения поставленной задачи. Таким образом, все нейроны начинают принимать более активное участие в формировании требуемого выходного значения. Такой подход требует указания вероятности исключения каждого из нейронов на любой обучающей итерации.

Итак, каждый слой представляет из себя класс, унаследованный от абстрактного класса Layer, предоставляемого API Keras. Для архитектуры капсулевых нейронных сетей реализованы классы PrimaryCaps, CapsuleLayer, Length. Для обучения и формирования вторичного капсулевого слоя написан алгоритм динамической маршрутизации между капсулами. Все слои объединяются в последовательную модель.

Был произведен подбор оптимальных значений гиперпараметров модели капсулевой нейронной сети. Окончательная архитектура модели представлена в листинге 3.2. Далее приведены пояснения к аргументам.

Листинг 3.2. Модель капсулевой нейронной сети.

```
def caps_net_model(input_shape, n_class, routings):
    x = layers.Input(shape=input_shape)
    conv = layers.Conv2D(filters=225, kernel_size=7, strides=2, padding='valid', activation='relu', name='conv')(x)
    drop = layers.Dropout(0.3)(conv)
    primary_caps = PrimaryCaps(drop, dim_capsule=15, n_channels=25, kernel_size=8, strides=2, padding='valid')
    class_caps = CapsuleLayer(num_capsule=n_class, dim_capsule=30, routings=routings, name='class_caps')(primary_caps)
    out_caps = Length(name='capsnet')(class_caps)
    model = models.Model(x, out_caps)
    return model
```

На входной слой поступает изображение дорожного знака размера 48x48. С помощью сверточного слоя выделяются базовые паттерны, используя 225 ядер размера 7x7 с шагом 2, в результате чего получается тензор размера 21x21x225. К нему применяется активация ReLU. На этом моменте во время обучения с вероятностью 0.3 отключается каждый нейрон. Затем происходит свертка выходного стека ядром 8x8x225 и шагом 2. Формируются 25 каналов, где каждый канал содержит 7x7 капсул размера 15 (всего 1225 капсул). Эти первичные капсулы поступают на вход алгоритма динамической маршрутизации, в результате которого образуется последний слой из 25 капсул размера 30. Структура полученной модели представлена на рисунке 3.8.

Layer (type)	Output Shape	Param #
<hr/>		
input_1 (InputLayer)	(None, 48, 48, 3)	0
<hr/>		
conv (Conv2D)	(None, 21, 21, 225)	33300
<hr/>		
dropout_1 (Dropout)	(None, 21, 21, 225)	0
<hr/>		
primarycap_conv2d (Conv2D)	(None, 7, 7, 375)	5400375
<hr/>		
primarycap_reshape (Reshape)	(None, 1225, 15)	0
<hr/>		
primarycap_squash (Lambda)	(None, 1225, 15)	0
<hr/>		
class_caps (CapsuleLayer)	(None, 25, 30)	13781250
<hr/>		
capsnet (Length)	(None, 25)	0
<hr/>		
Total params: 19,214,925		
Trainable params: 19,214,925		
Non-trainable params: 0		

Рис.3.8. Числовые характеристики структуры разработанной капсулной нейросети.

Таким образом, было натренировано около 19 млн. параметров сети. Для обучения использованы значения гиперпараметров, представленные в табл. 3.1.

Таблица 3.1 Гиперпараметры.

Количество эпох (epochs)	10
Размер мини-партии (batch_size)	50
Коэффициент скорости обучения (lr)	0.001
Значение, умножаемое на коэффициент скорости обучения на каждой эпохе (lr_decay)	0.9
Количество итераций алгоритма динамической маршрутизации	3

Капсулльная нейронная сеть обучалась на протяжении 10 эпох. Одна эпоха обозначает, что весь набор обучающих данных прошел через нейросеть один раз.

Использовался размер пакета, равный 50. Этот гиперпараметр обозначает количество примеров обучающей выборки, которое необходимо пропустить через нейросеть перед тем, как обновить внутренние параметры модели.

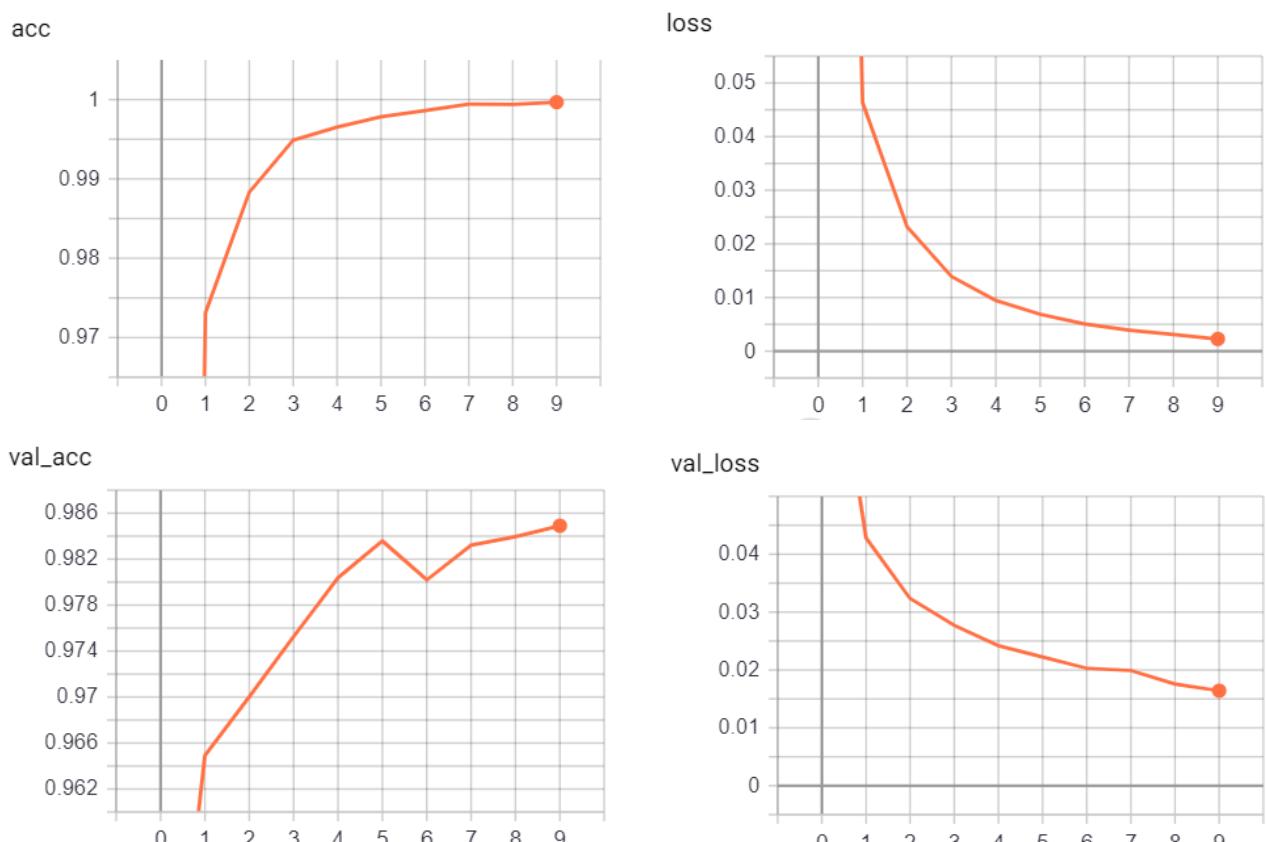
Коэффициент скорости обучения равен 0.001 (коэффициент η при градиенте: $\Delta w = -\eta \frac{\partial C}{\partial w}$).

После каждой эпохи коэффициент скорости обучения умножался на 0.9 для уменьшения шага коррекции с целью более точной настройки.

Число итераций для алгоритма динамической маршрутизации равно 3.

После обучения получен следующий результат. Показатель точности (Accuracy) на тренировочном наборе составляет 99.97%, на тестовом — 98.49%.

Графики точности и потери представлены на рис. 3.9.



*Rис. 3.9. Графики точности и потери.
Сверху — на тренировочном наборе
Снизу — на валидационном (тестовом) наборе.*

3.4 Модификация и обучение YOLOv3

Модель YOLO была разработана на основе Darknet. Это фреймворк нейросетей с открытым исходным кодом, написанный на языках С и CUDA.

На официальном сайте YOLO [9] можно найти модели, которые уже прошли обучение с набором данных COCO Dataset для детектирования 80 различных классов предметов (автомобиль, человек, кот, собака и т.д.). К сожалению, там нет класса дорожных знаков, поэтому нужно натренировать сеть на пользовательском наборе данных дорожных знаков. Для этого, используя исходный код модели YOLOv3 [10], сделаны следующие действия:

1. Используя необходимые инструменты для вычислений на GPU (CUDA, OpenCV и др.) была скомпилирована модель.
2. Чтобы не тренировать нейросеть с нуля, были использованы существующие предварительно обученные веса сверточных слоев *darknet53.conv.74*, которые уже обучены на огромных наборах данных. Файл помещен в папку *build\darknet\x64*. После корректировки этих весов будет получен свой файл с расширением *.weights* для детектирования знаков.
3. Внесены изменения в конфигурацию нейросети. Сначала был создан файл с названием *yolo-obj.cfg* с тем же содержимым, что и в *yolov3.cfg*, который содержит в себе архитектуру версии YOLOv3. Файлы находятся в директории *build\darknet\x64\cfg*. В новой конфигурации установлены такие параметры:
 - batch=64 — размер пакета (партии), количество изображений на 1 итерацию;
 - subdivisions=16 — на сколько мини-партий (mini-batches) делится партия, $64/16=4$ изображения на мини-партию, это отправляется на GPU;
 - max_batches = classes*2000 (но не меньше 4000). classes — количество классов в новой модели. Так как необходимо детектировать 1 класс (дорожные знаки), то параметр устанавливается в 4000. Это максимальное число итераций.
 - steps = 3200, 3600 (80% и 90% max_batches);
 - width=416, height=416 (значение, кратное 32);
 - classes = 1 — в каждом из трёх [yolo]-слоев.
 - filters = (classes+5)*3 = 18 в трёх [convolutional] перед каждым слоем [yolo].

4. Создан файл *obj.names* в директории build\darknet\x64\data\, в котором перечислены названия всех новых классов. В нём одна строка: Traffic sign
5. Создан файл *obj.data* в директории build\darknet\x64\data\, содержащий различные пути:

```
classes = 1
train = build/darknet/x64/data/train.txt
valid = build/darknet/x64/data/test.txt
names = build/darknet/x64/data/obj.names
backup = build/darknet/x64/backup/
```

6. Все изображения RTSD-D1 и аннотации к ним помещены в build\darknet\x64\data\obj\. Для этого написан скрипт на Python, который из исходных аннотаций к изображениям создавал аннотации, требуемые для обучения модели YOLO. Для каждого изображения .jpg создан файл .txt, содержащий одну или несколько строк, соответствующих имеющимся на изображении дорожным знакам, следующего вида:

<object-class> <x_center> <y_center> <width> <height>

<object-class> — целое число (номер класса) от 0 до (classes-1).

<x_center> <y_center> <width> <height> — вещественные значения относительно ширины и высоты изображения в диапазоне (0.0, 1.0], соответствующие центру и размерам окна детектирования. Например, для img1.jpg может быть создан img1.txt со следующим содержимым (3 знака):

```
0 0.716797 0.395833 0.216406 0.147222
0 0.687109 0.379167 0.255469 0.158333
0 0.420312 0.395833 0.140625 0.166667
```

Также написанная программа формирует файлы *train.txt* и *test.txt*, содержащие пути к изображениям. Файлы находятся в директории build\darknet\x64\data\.

7. Наконец, было запущено обучение с помощью команды:
- | | | | |
|---|-----------------|--------------|--|
| <i>darknet.exe</i> | <i>detector</i> | <i>train</i> | <i>build/darknet/x64/data/obj.data</i> |
| <i>build/darknet/x64/cfg/yolo-obj.cfg</i> | | | <i>build/darknet/x64/darknet53.conv.74</i> |

Обучающая выборка для детектирования RTSD-D1 содержит 3821 тренировочных и 1274 тестовых кадра.

Обучение YOLO длилось на протяжении 2500 итераций. Так как на одну итерацию приходится один batch (64 изображения), то получится $3821/64=60$ итераций на эпоху, тогда всего — коло 42 эпохи. В промежутке между 1500 и 2500 итераций значение функции потерь практически не изменялось, поэтому было принято решение прекратить обучение. При этом переобучения не наблюдалось: значение mAP на валидационном (тестовом) наборе не уменьшалось.

Каждые 100 итераций программа сохраняла значения весов, чтобы можно было, к примеру, прервать обучение и продолжить его через некоторое время. Также во время обучения строился график, по которому можно было наблюдать значение потерь. Удалось сохранить только часть графика всего процесса обучения по причине частого прерывания (рисунок 3.10).

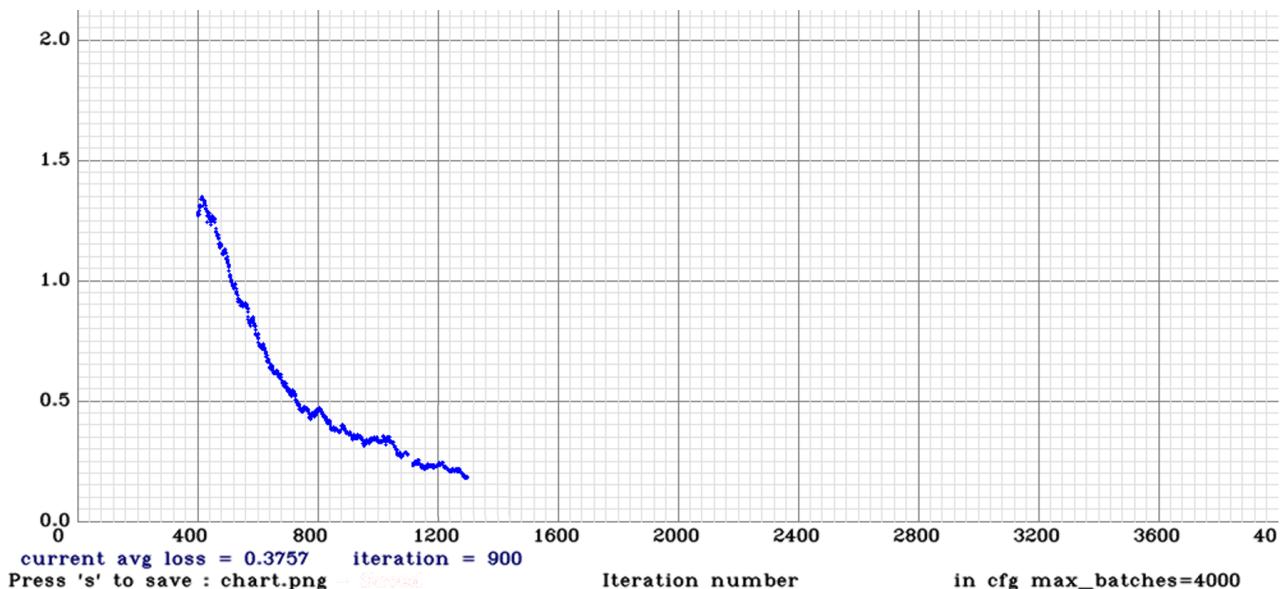


Рис.3.10. Значение функции потерь на итерациях 400 – 1300.

В результате обучения получены следующие оценки на тестовом наборе:

- IoU = 64.14 %;
- mAp (AP) = 88.36 %.

Более подробную информацию можно увидеть на рисунке 3.11.

```
[yolo] params: iou_loss: mse (2), iou_norm: 0.75, cls_norm: 1.00, scale_x_y: 1.00
Total BFLOPS 65.304
avg_outputs = 516723
Allocate additional workspace_size = 13.11 MB
Loading weights from build/darknet/x64/backup/yolo-obj_2500.weights...
seen 64, trained: 160 K-images (2 Kilo-batches_64)
Done! Loaded 107 layers from weights-file

calculation mAP (mean average precision)...
1244
detections_count = 3724, unique_truth_count = 1905
class_id = 0, name = Traffic sign, ap = 88.36% (TP = 1632, FP = 271)

for conf_thresh = 0.25, precision = 0.86, recall = 0.86, F1-score = 0.86
for conf_thresh = 0.25, TP = 1632, FP = 271, FN = 273, average IoU = 64.14 %

IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
mean average precision (mAP@0.50) = 0.883637, or 88.36 %
Total Detection Time: 46 Seconds

Set -points flag:
`-points 101` for MS COCO
`-points 11` for PascalVOC 2007 (uncomment `difficult` in voc.data)
`-points 0` (AUC) for ImageNet, PascalVOC 2010-2012, your custom dataset
```

Рис.3.11. Расчёт оценок после обучения на 2500 итераций.

Таким образом, полученные веса выдают достаточно высокие показатели, что говорит о хорошей точности алгоритма.

3.5 Демонстрация работы

Детектирование знаков с помощью полученной модели YOLO можно протестировать напрямую, используя опции исполняемого файла:

```
darknet.exe detector test build/darknet/x64/data/obj.data build/darknet/x64/cfg/yolo-obj.cfg
build/darknet/x64/backup/yolo-obj_last.weights -ext_output sign_06.jpg
```

Чтобы продемонстрировать работу системы распознавания был написан графический пользовательский интерфейс (GUI) для работы с полученными моделями. Для этого были написаны API для работы как с капсулой нейронной сетью, так и с YOLOv3. Примеры работы приложения можно увидеть на рисунках 3.12 – 3.19.

После запуска приложения появляется главное окно, представленное на рис. 3.12. Главное окно предполагает работу с кадрами для детектирования дорожных знаков на фотографии.

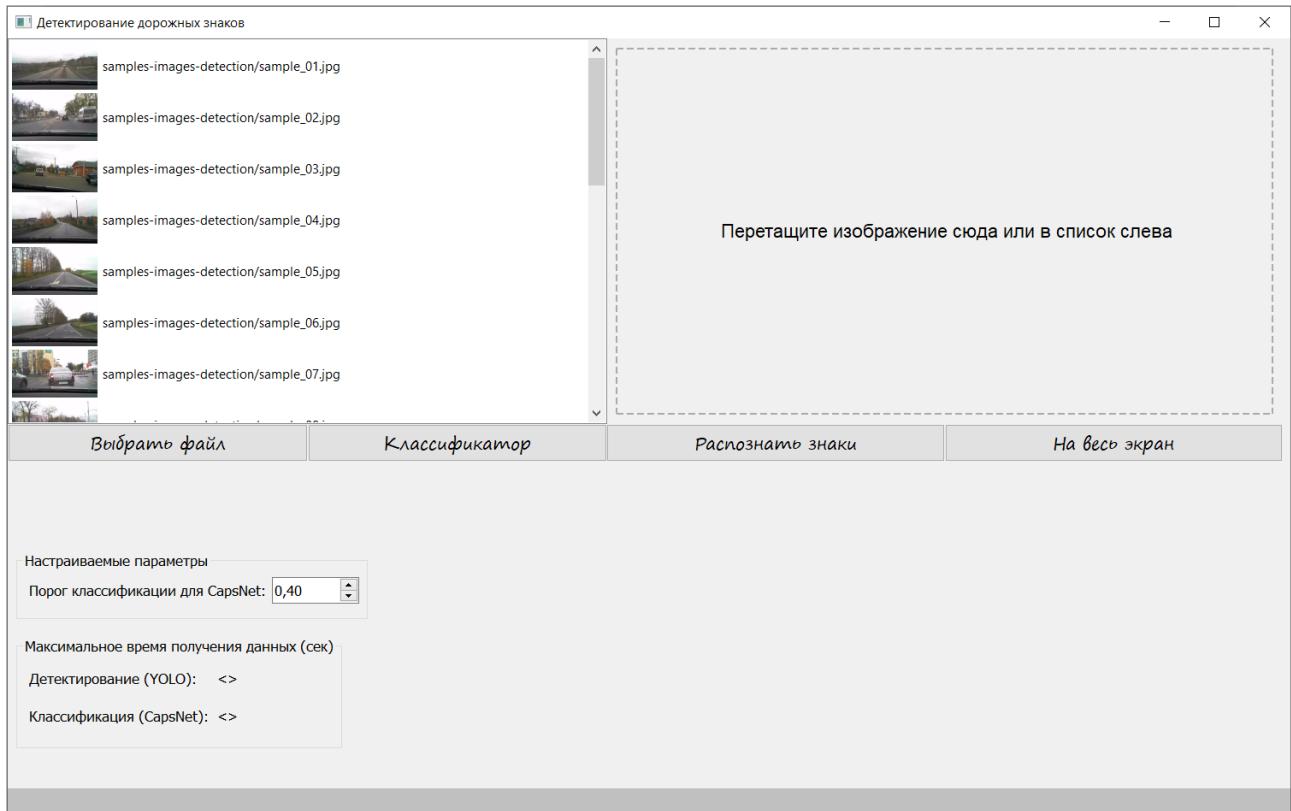


Рис. 3.12. Внешний вид программы (GUI).

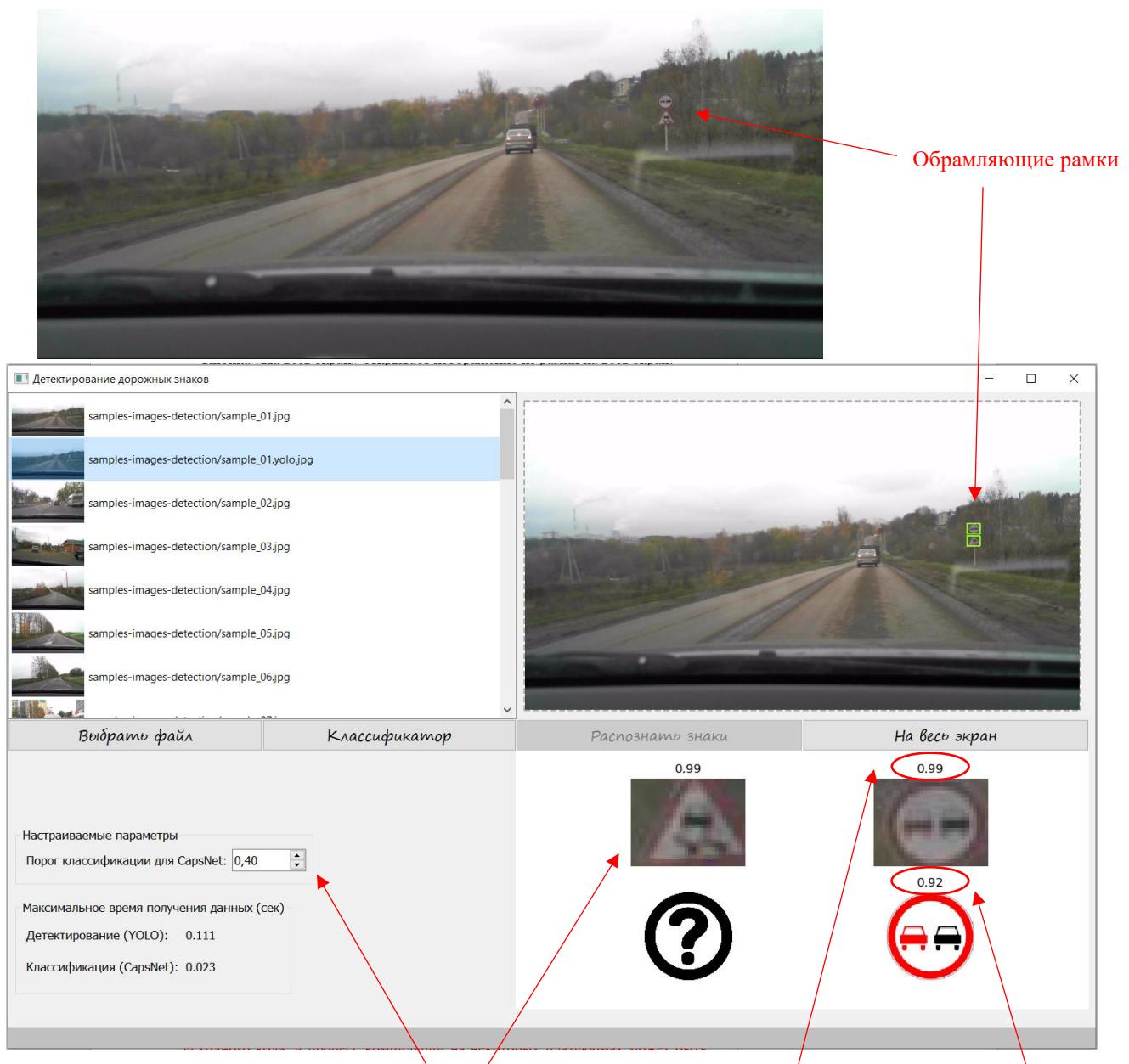
Функционал программы:

- С помощью механизма Drag and Drop («Перетаскивание и Отпускание») можно «перетащить» мышкой интересующее изображение в список предпросмотра или в специальную рамку.
- Можно не использовать механизм D&D, а нажать кнопку «Выбрать файл» и найти нужное изображение через проводник ОС.
- При нажатии на кнопку «Распознать знаки» программа запускает алгоритм YOLO, получает информацию о детектировании, далее по полученным данным запускает модель капсулной нейронной сети для классификации. Весь вывод отображается в нижней половине главного окна:
 - вырезанные знаки (с показателем уверенности обнаружения);
 - классификация знаков (с показателем уверенности капсулы);
 - максимально время, затраченное на детектирование и классификацию.
- Кнопка «На весь экран» открывает изображение из рамки на весь экран.

- Кнопка «Классификатор» открывает самостоятельное окно, аналогичное главному окну, для тестирования капсулой сети исключительно на изображениях вырезанных дорожных знаков.
- Можно ввести порог для классификации знаков.

Следует еще раз отметить, что классификатор обучен идентифицировать только 25 классов дорожных знаков. Все классы приведены в приложении А.

Итак, пример работы представлен на рисунке 3.13



*Рис. 3.13.
Пример работы
с пояснениями.*

Данного класса не было в обучающей выборке.
Капсулальная сеть его не идентифицировала
(уверенность капсулы < 0.4).

Уверенность
детектирования

Уверенность
капсулы

Можно убрать порог классификации (обнулить), чтобы для предыдущего примера ради интереса посмотреть капсулу с наибольшей уверенностью. Тогда получиться результат, как на рисунке 3.14.

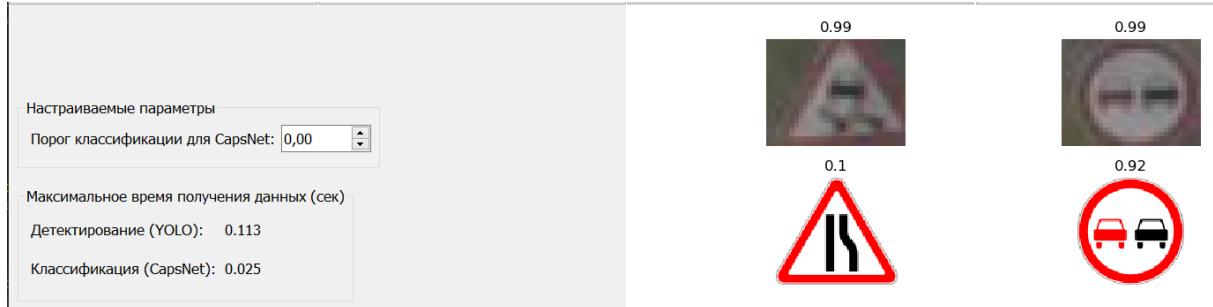


Рис. 3.14. Сравнение с примером на рисунке 3.13 при нулевом пороге классификации.

Другие примеры приведены на рисунках 3.15 – 3.19.

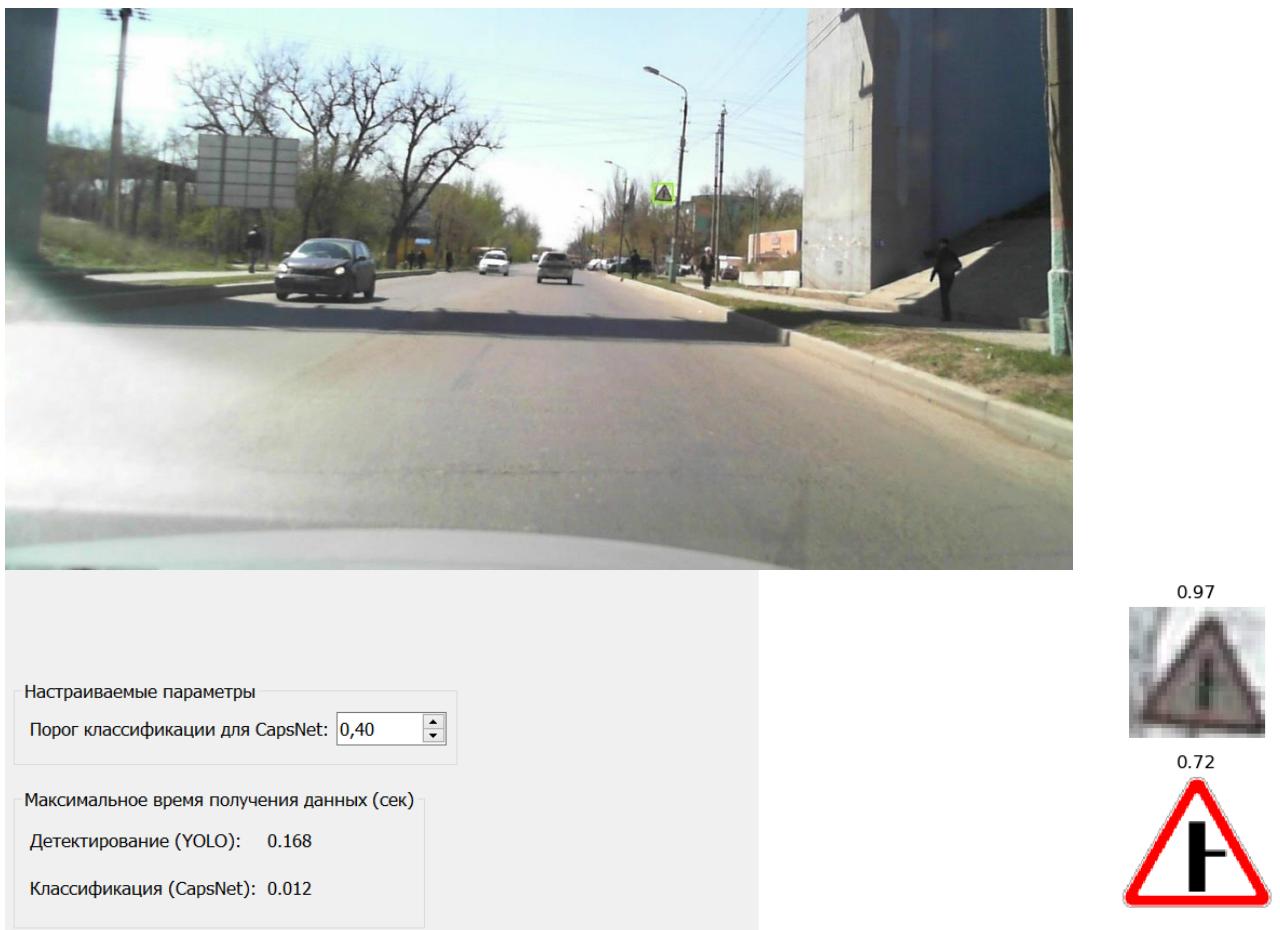


Рис. 3.15. Пример распознавания.

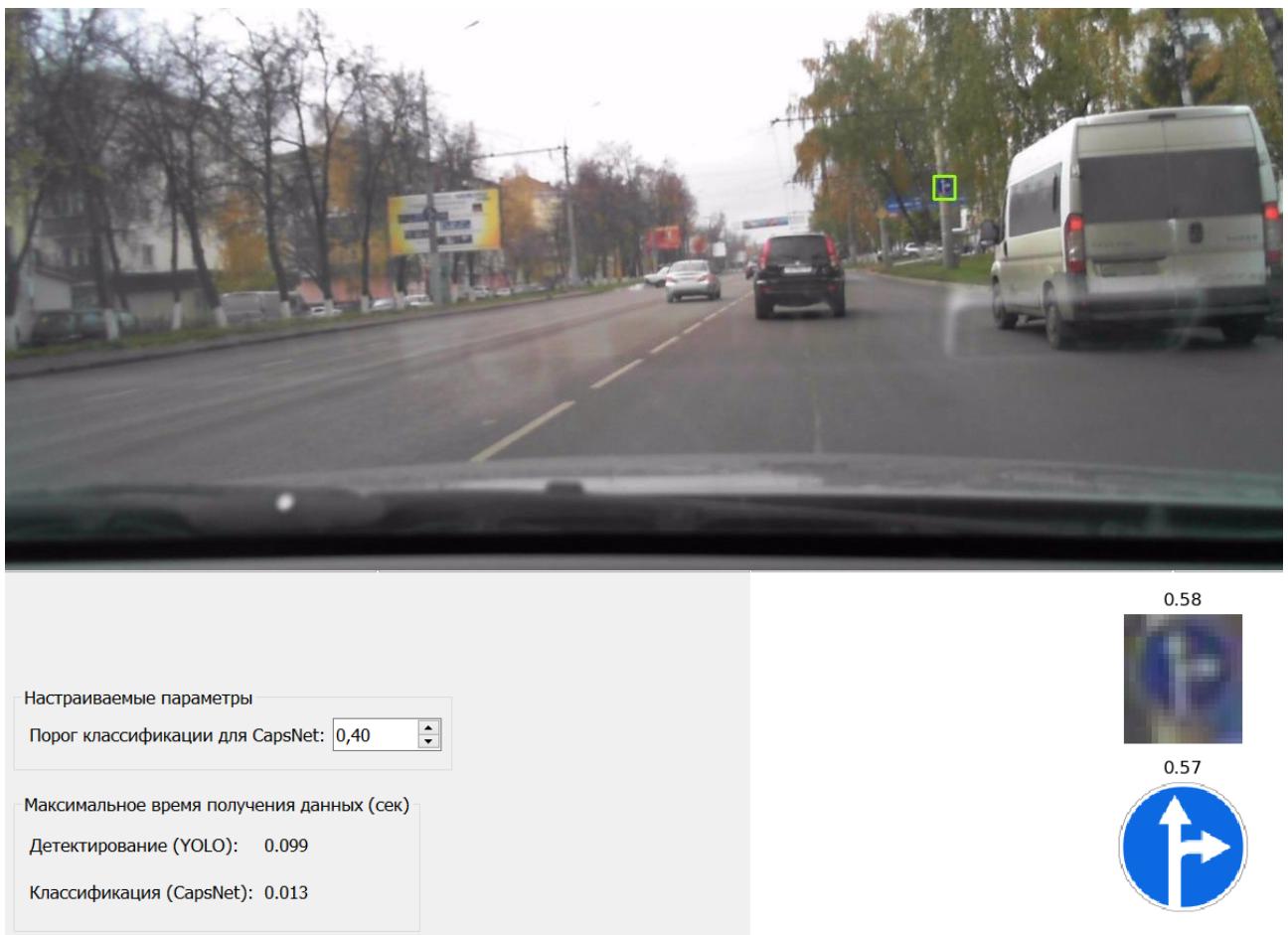


Рис. 3.16. Пример распознавания.

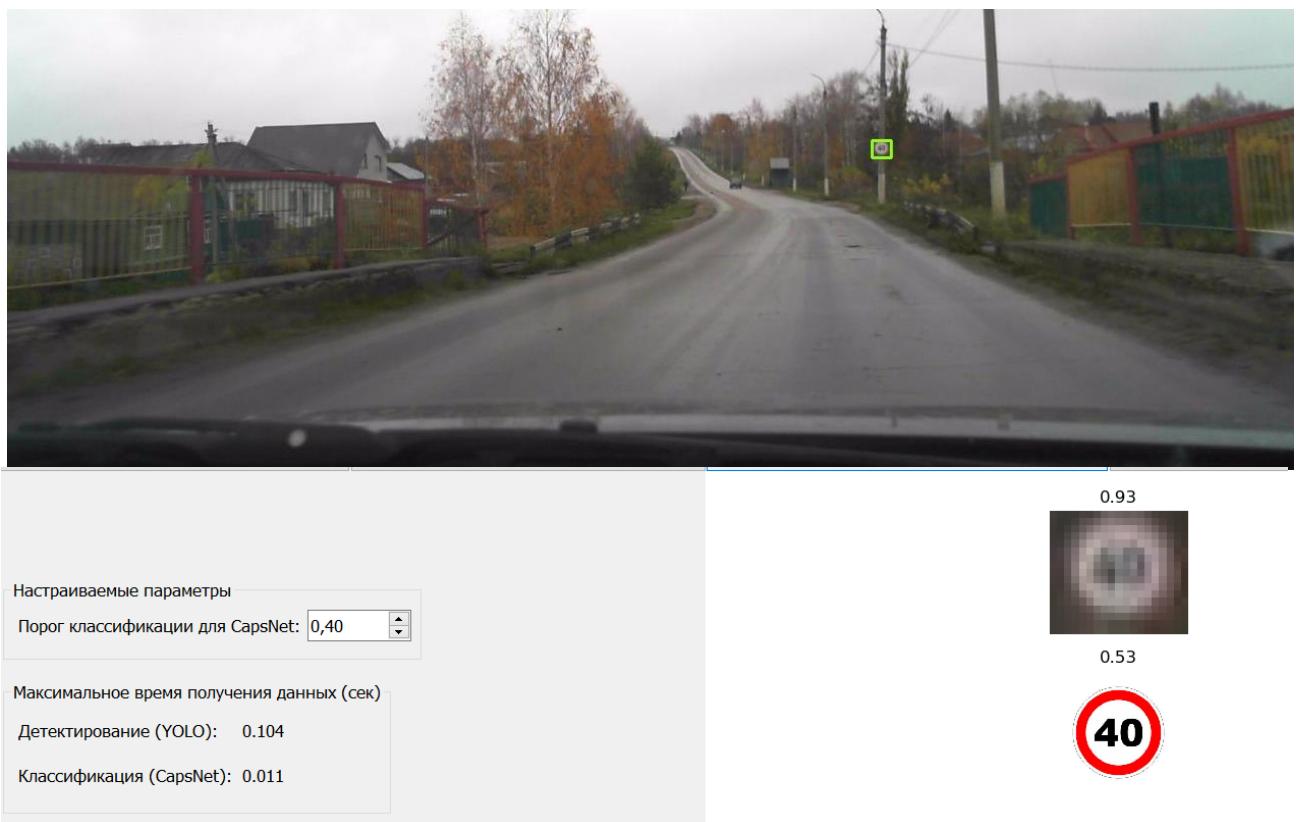


Рис. 3.17. Пример распознавания.

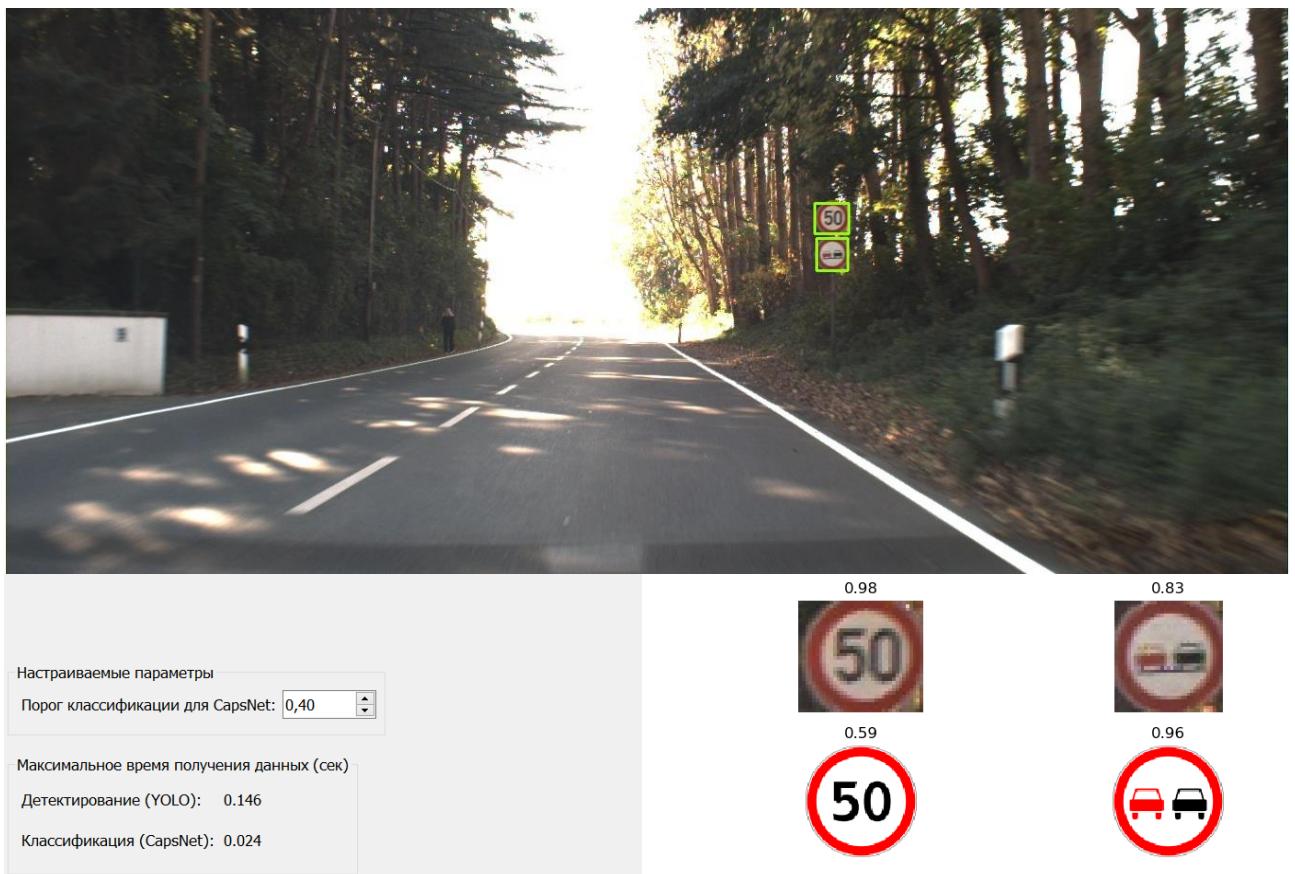


Рис. 3.18. Пример распознавания.

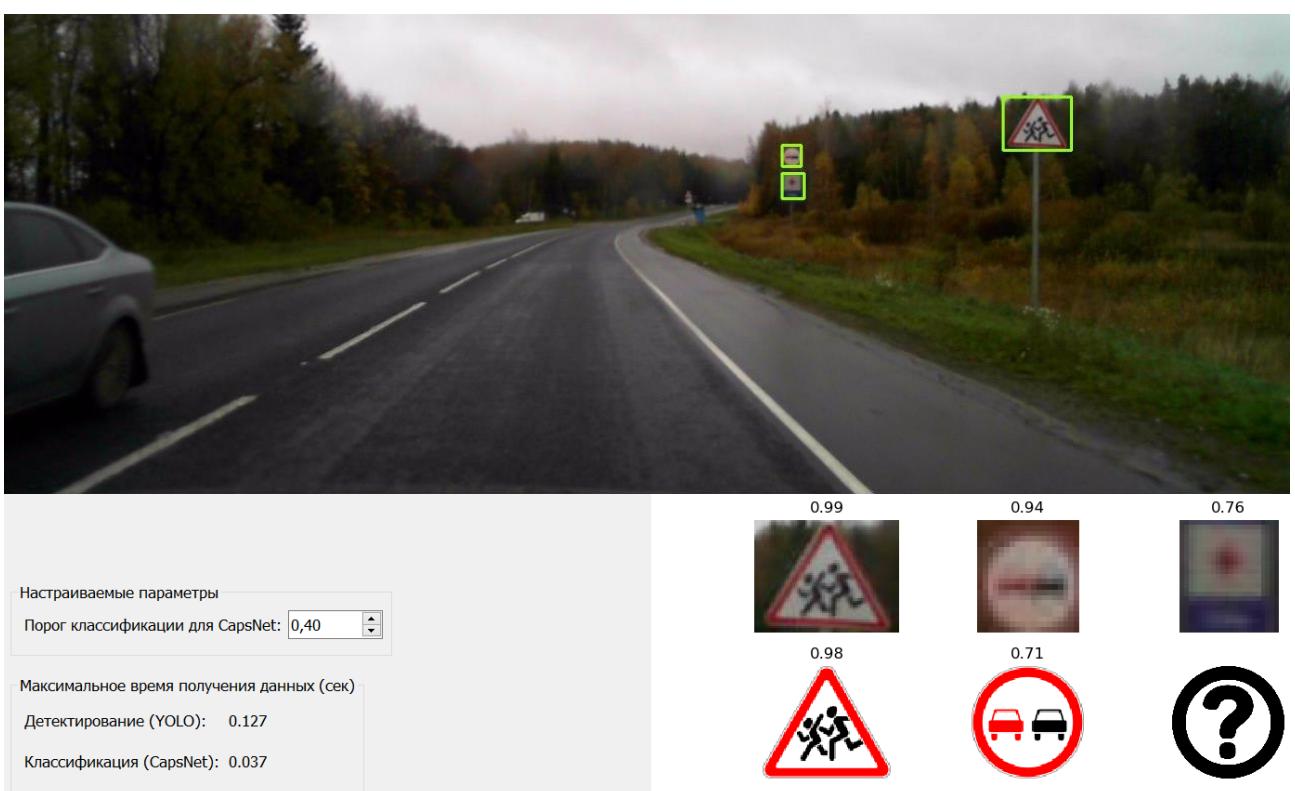


Рис. 3.19. Пример распознавания.

3.6 Выводы и особенности работы системы обнаружения

В разделе описаны применяемые алгоритмы предварительной обработки изображений, улучшена обучающая выборка на основе методов аугментации, получены разнообразные изображения в большом количестве.

Представлены модификации нейронных сетей YOLO и CasNet для распознавания дорожных знаков, а также процессы их обучения.

По результатам разработанного ПО, видно, что объекты, присутствующие в списке классов, обнаружены с достаточно хорошей точностью. Время, затрачиваемое на получение данных детектирования и классификации на отдельном кадре с использованием GPU, говорит о высокой скорости алгоритма (суммарно не более 150 мс.).

Протестировав различные кадры, было обнаружено, что алгоритм лучше справляется с детектированием знаков, если они располагаются на среднем и дальнем расстоянии от камеры. Если знак расположен слишком близко, то есть он имеет большой размер относительно всего изображения, то алгоритм может его не задетектировать. Это связано с особенностями тренировочной выборки: все знаки на обучающих кадрах расположены далеко от камеры и имеют небольшой размер.

Darknet написан на С и не имеет другого программного интерфейса, поэтому использование других языков программирования требует дополнительной работы над интеграцией. Также darknet распространяется только в формате исходного кода, и процесс компиляции на разных платформах может быть весьма долгим и проблематичным.

Характеристики решений, полученных с помощью машинного обучения, существенно зависят от размера и качества обучающей выборки. Дальнейшее развитие разработанной системы для повышения качества и точности заключается именно в пополнении обучающей выборки и повышении её репрезентативности.

4. Экспериментальный раздел

Целью исследований является выяснение эффективности реализованных методов, сравнение с другими реализациями и выяснение особенностей работы капсулльных нейронных сетей.

4.1 Предварительная обработка

Цель: определить эффективность предварительной обработки изображений обучающей выборки для классификации и сравнить результаты обучения капсулльной нейронной сети на двух выборках: до и после обработки.

Как было описано в технологическом разделе (3.2), выборка RTSD-R1, которая использовалась для обучения капсулльной нейронной сети была предварительно обработана:

- отброшены неразличимые изображения знаков;
- применены методы аугментации, чтобы создать больше вариаций изображений.

Объем выборки до обработки: 21428 (обучение) и 5848 (тестирование).

Объём выборки после обработки: 21345 (обучение) и 5301 (тестирование).

Количество классов: 25.

На рисунке 4.1 представлен процесс обучения ранее описанной модели капсулльной сети на двух выборках.

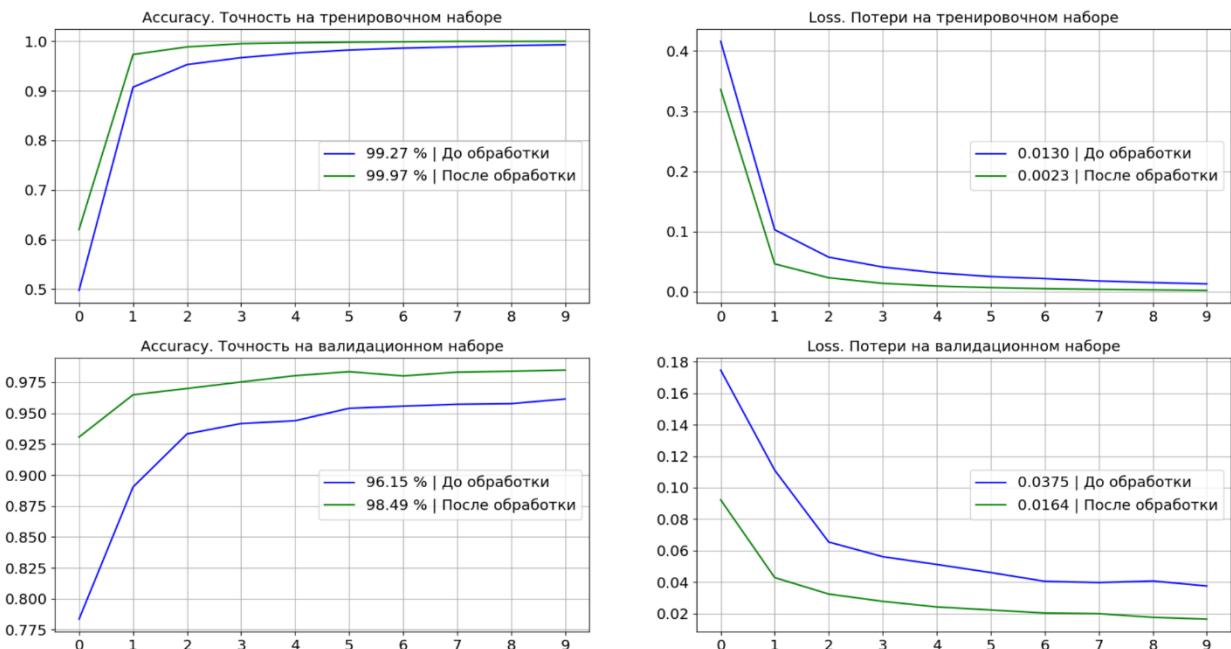


Рис. 4.1. Сравнение процесса обучения на исходной и обработанной выборке (10 эпох).

Вывод: реализованные методы предварительной обработки положительно сказываются на процессе обучения, они позволяют более эффективно и быстро тренировать сеть и показывают лучшие результаты точности как на тренировочной, так и на тестовой выборках. Функция потерь также меньшие показатели на протяжении обучения.

4.2 Влияние параметров на обучение

Описанные в технологическом разделе гиперпараметры и используемая архитектура (3.3) показывают достойные результаты. Архитектура модели и значения гиперпараметров выбирались с учетом следующих показателей:

- Сходимость сети. Это ответ на вопрос: «способна ли сеть обучаться?» Для этого наблюдались изменения значений потерь на первых эпохах. Если ошибка оставалась высокой и не менялась на протяжении одной-двух эпох, то делался вывод, что сеть не может скорректировать коэффициенты, и процесс обучения останавливался.
- Насколько быстро сеть учится. Были приняты во внимание количество эпох, которые проходили для достижения приемлемых результатов.
- Точность по результатам обучения Accuracy.

В сообществе машинного обучения общеизвестно, что сложно делать общие заявления о влиянии гиперпараметров, поскольку поведение часто варьируется от набора данных к набору данных и от модели к модели. Поэтому сделанные в дальнейшем выводы служат только указаниями, касающимися текущей работы, а не общими утверждениями.

Капсулльная сеть сильно чувствительна к изменениям параметров, в отличие от сверточных, где можно более свободно варьировать значениями для достижения приемлемых результатов.

4.2.1 Скорость обучения

Цель: определить оптимальный гиперпараметр для скорости обучения капсулльной нейросети на построенной архитектуре.

Было проведено сравнение процесса тренировки на разных коэффициентах скорости обучения (learning rate). Это параметр градиентных алгоритмов обучения нейронных сетей, позволяющий управлять величиной коррекции весов на каждой итерации. Выбирается в диапазоне от 0 до 1. Выбор параметра противоречив. Большие значения будут соответствовать большому значению шага коррекции. При этом алгоритм будет работать быстрее, т.е. для поиска минимума функции ошибки потребуется меньше итераций. Однако может возникнуть ситуация, которая приведет к расхождению оптимизации, потому что алгоритм будет «перепрыгивать» с одной стороны «каньона» на другую, либо снизиться точность настройки модели на минимум функции ошибки, что потенциально увеличит ошибку обучения. При малых значениях скорость сходимости к минимуму функции будет малой, однако меньше шанс его пропустить. Кроме того, малые значения скорости обучения могут не позволить добиться реального прогресса, и даже после обучения нейронной сети в течение длительного времени она будет далека от оптимальных результатов. В выборе скорости часто полагаются на интуицию и эксперименты.

На рис. 4.2 представлены результаты на разных коэффициентах.

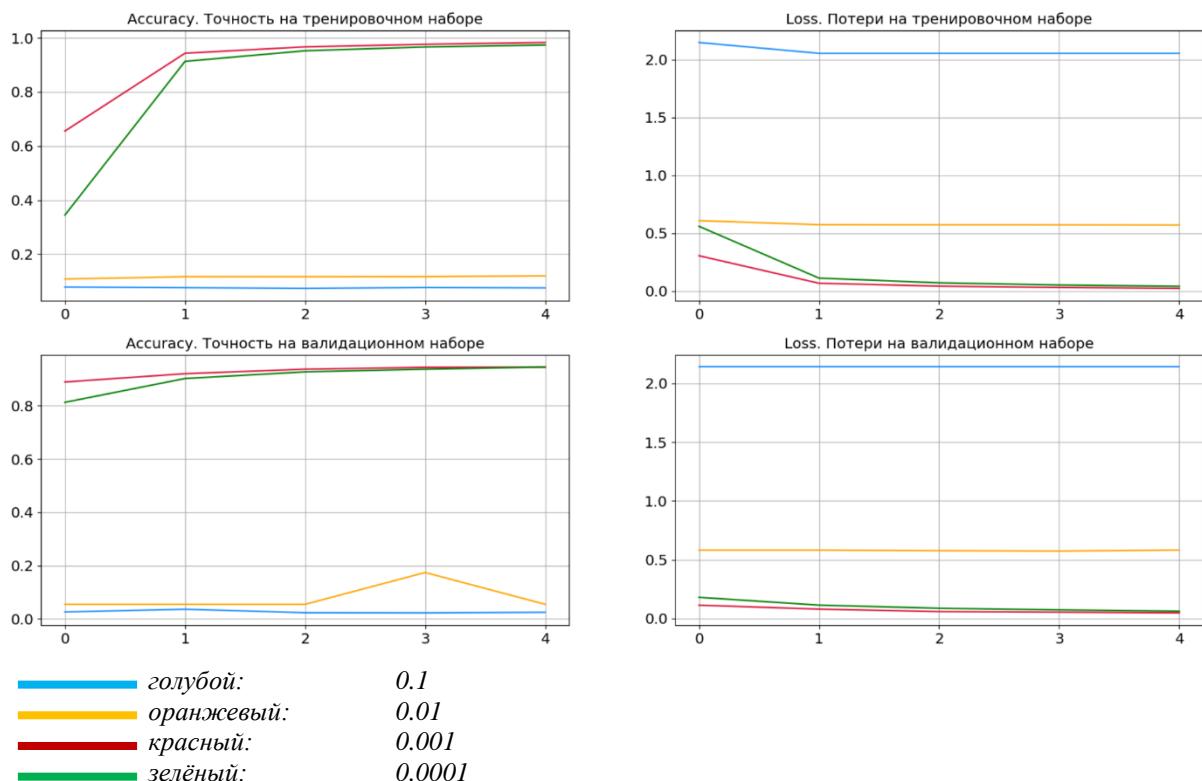


Рис. 4.2. Сравнение процесса обучения на разных коэффициентах скорости обучения (5 эпох)

Вывод: по результатам экспериментов видно, что коэффициенты 0.1 и 0.01 не позволяют оптимизировать функцию потерь (слишком высокие значения). Сеть на таких значениях не обучалась. Другие два коэффициента дали возможность тренировать сеть и минимизировать функцию ошибки. Однако обучение с коэффициентом 0.001 проходило более быстро и эффективно, чем с 0.0001. Таким образом, выбранное значение скорости обучения 0.001 является оптимальным.

4.2.2 Размер пакета

Цель: определить оптимальное значение размера пакета для обучения капсулой нейросети на построенной архитектуре.

Для реализации алгоритма обучения возможны 3 подхода:

1. Пакетный (полный) градиентный спуск — на каждой итерации обучающая выборка просматривается целиком, и только после этого изменяются веса модели (обновление в конце каждой эпохи). Вычисление такого градиента эффективно (т. к. точно сходится к минимуму), но занимает много времени.
2. Стохастический градиентный спуск (SGD) вычисляет ошибку и обновляет модель для каждого отдельного примера в наборе обучающих данных. Поведение функции потерь будет хаотично меняться при каждой итерации, но в долгосрочной перспективе она также уменьшается и сходится к минимуму.
3. Мини-пакетный градиентный спуск — компромисс между первыми двумя методами. Обучающий набор данных разбивается на небольшие партии, которые используются для расчета ошибки модели и обновления коэффициентов. Это наиболее распространенный способ.

Размер партии является одним из наиболее важных гиперпараметров для настройки. Практики часто хотят использовать больший размер пакета для обучения своей модели, поскольку это позволяет ускорить вычисления из-за параллелизма графических процессоров. Однако слишком большой размер пакета может привести к плохому обобщению (в настоящее время неизвестно, почему это так). Эмпирически было показано, что использование партий меньшего размера обеспечивает более быструю сходимость к «хорошим» решениям. Это интуитивно объясняется тем фактом, что меньшие размеры пакетов позволяют

модели «начать обучение, прежде чем увидеть все данные». Существенным недостатком использования меньшего размера партии является то, что модель не гарантирует достижения глобального минимума. Это зависит от отношения размера партии к размеру набора данных. Рекомендуется начинать с небольшого размера партии, получая преимущества более быстрой динамики обучения и неуклонно увеличивая размер партии за счет обучения, а также получая преимущества гарантированной конвергенции.

Был проведен эксперимент с разными размерами партий (рис. 4.3).

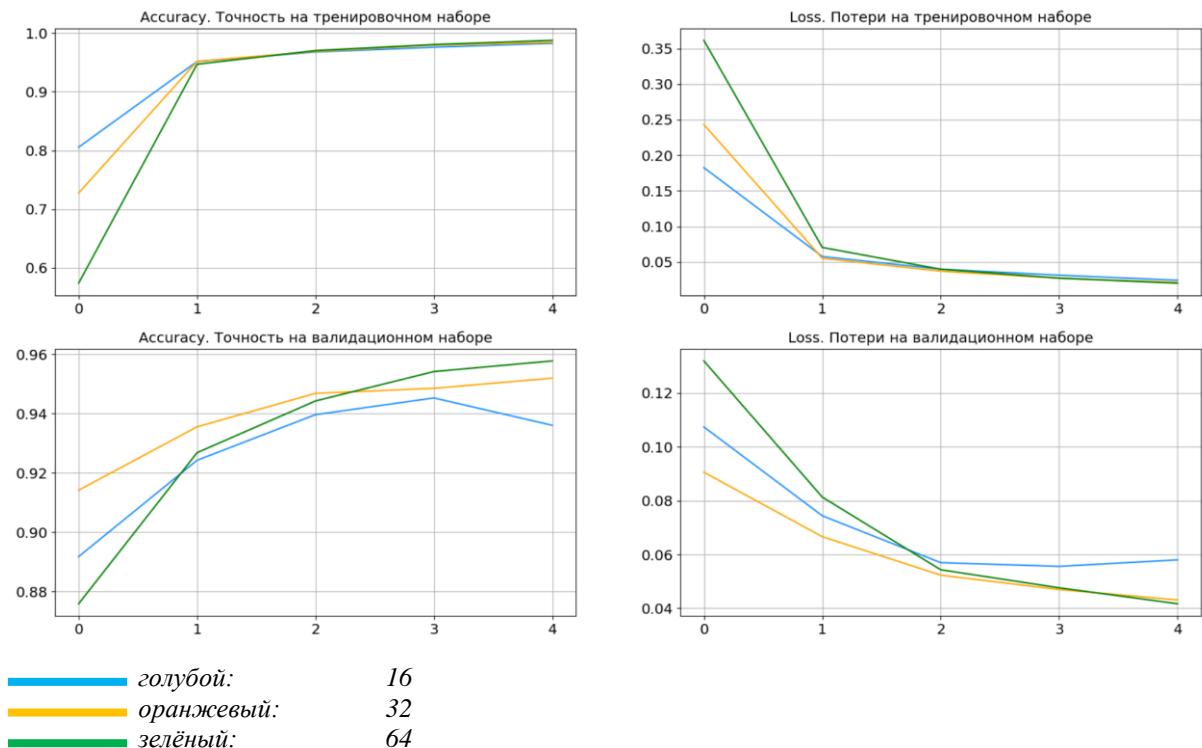


Рис. 4.3. Сравнение процесса обучения на разных размерах партий (5 эпох).

Вывод: как видно из графиков, 16 изображений на партию действительно обладает высокой динамикой обучения: точность на тренировочном наборе быстро возрастает, а потери на первых эпохах меньше по сравнению с остальными размерами, но на 5 эпохе можно наблюдать признаки переобучения. Размеры 32 и 64 показывают более медленную динамику, но лучшую сходимость. К сожалению, более высокие размеры не удалось протестировать из-за ограничения видеопамяти GPU. Таким образом, взят размер партии 50, что является оптимальным выбором с учетом наблюдаемых показателей.

Следует отметить, что часто вместе с увеличением размера пакета повышают скорость обучения, чтобы оптимизировать обучение модели.

4.3 Сравнение с другими реализациями

В таблице 4.1 приведено сравнение на различных реализациях и выборках.

Большинство данных взяты из статьи [11].

Таблица 4.1. Сравнение точности распознавания на различных методах и выборках.

Данные	Классов	Основной метод	Точность %
RTSD-R1. Выборка с обработкой (сбалансированная)	25	Капсулльная сеть [текущая работа]	98.49
RTSD-R1. Выборка без обработки	25	Капсулльная сеть [текущая работа]	96.15
RTSD-R1 (полная, без обработки)	66	Свёрточная сеть [8]	90,78
GTSRB	43	Капсулльная сеть [11]	97.62
GTSRB	43	Random Forests [12]	96.14
GTSRB	43	LDA (HOG 2) [13]	95.68
GTSRB	43	LDA (HOG 1) [14]	93.18
GTSRB	43	LDA (HOG 3) [13]	92.34

Обнаружение дорожных знаков является сложной задачей, особенно учитывая «тяжелую» выборку российских дорожных знаков. Таким образом, капсулльные сети, работают лучше по сравнению с сетями CNN и другими методами, что говорит об их надежности и точности для классификации и распознавания изображений даже на размытых, повернутых и искаженных картинках, которые в большом количестве присутствуют в российской базе дорожных знаков.

4.1 Выводы

Проведены эксперименты с капсулльной нейронной сетью, показывающие высокую чувствительность архитектуры к значениям используемых параметров. Исследования показывают, какие значения гиперпараметров для обучения являются оптимальными. Небольшое количество используемых эпох говорит о способности сети быстро обучаться.

Показана эффективность предварительной обработки, а также преимущества использования модели капсул по сравнению с другими методами.

Заключение

В рамках дипломной работы:

- Проведено сравнение нейросетей как классификаторов, среди которых многослойный персепtron, сверточная нейронная сеть и капсульная сеть.
- Рассмотрены особенности работы современной модели YOLOv3 для детектирования объектов.
- Использованы выборки из базы российских дорожных знаков, и предварительно обработана выборка вырезанных знаков в целях лучшего обучения капсулльной нейронной сети для задачи классификации.
- Построена капсулльная нейронная сеть и выбраны оптимальные гиперпараметры, в результате чего после обучения была достигнута точность классификации равная 98.49%. Такой результат свидетельствует о хорошей работоспособности представленной архитектуры капсулльных сетей и целесообразности её применения для задач классификации.
- Модифицирована модель YOLOv3 для решения задачи детектирования дорожных знаков, которая показывает отличный результат на кадрах, в которых знаки находятся на среднем и дальнем расстоянии от камеры. Полученное решение показало достаточно высокую точность локализации.
- Создан удобный графический интерфейс, позволяющий экспериментировать с разработанной системой распознавания, в которой результаты детектора в виде вырезанных дорожных знаков подаются на вход капсулльной нейронной сети для их классификации.
- Проведены эксперименты с капсулльной нейронной сетью, показывающие адекватность используемых параметров обучения для построенной архитектуры, эффективность предварительной обработки, а также преимущества использования модели капсул с присущей им способностью определять пространственные отношения.

Таким образом, в результате проделанной работы разработана система распознавания дорожных знаков с использованием капсулльных нейронных сетей. Все поставленные задачи были выполнены.

Разработанная система имеет перспективу дальнейшего развития и улучшения. Высокие показатели скорости распознавания с использованием видеокарт позволяют создать API для детектирования и классификации знаков на видеопотоке в режиме реального времени. Возможны улучшения, связанные с интеграцией капсульной нейронной сети в архитектуру YOLO для создания единой модели сети. В целях экономии времени тренировки моделей на имеющихся вычислительных ресурсах были использованы ограниченные выборки данных для обучения. Поэтому для повышения показателей точности, а также для распознавания большего количества классов возможно дообучение моделей на большем количестве данных и на других выборках. Возможны улучшения с точки зрения дополнительной функциональности и оформления программного интерфейса.

Список источников и литературы

1. Созыкин А.В. Обзор методов обучения глубоких нейронных сетей // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2017. Т. 6, № 3. С. 28-59.
2. Hinton G.E., Sabour S., Frosst N. Dynamic routing between capsules // Advances in neural information processing systems. 2017. – С.3856-3866.
3. Viola P. Rapid object detection using a boosted cascade of simple features / P. Viola, M. Jones // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. – 2001. – Vol. 1. – P. 511-518.
4. Dalal N. Histograms of oriented gradients for human detection / N. Dalal, B. Triggs // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. – 2005. – Vol. 1. – P. 886-893.
5. Krizhevsky A. Imagenet classification with deep convolutional neural networks / A. Krizhevsky, I. Sutskever, G.E. Hinton // Advances in Neural Information Processing Systems. – 2012. – P. 1097-1105.
6. Redmon J., Divvala S., Girshick R., Farhadi A. You Only Look Once: Unified, Real-Time Object Detection // Computer Vision and Pattern Recognition – 2015.
7. Redmon J., Farhadi A. YOLOv3: An Incremental Improvement // Computer Science, arXiv: 1804.02767. – 2018.
8. Шахуро В. И., Конушин А. С. Российская база изображений автодорожных знаков // Компьютерная оптика. 2016. Т. 40. № 2. С. 294-300.
9. YOLO: обнаружение объектов в реальном времени [Электронный ресурс]. – URL: <https://pjreddie.com/darknet/yolo/> (29.04.2020)
10. YOLO. Версии Windows и Linux для детектирования объектов. [Электронный ресурс] – URL: <https://github.com/AlexeyAB/darknet>
11. Amara Dinesh Kumar. Novel Deep Learning Model for Traffic Sign Detection Using Capsule Networks // International Journal of Pure and Applied Mathematics Volume 118 No. 20 2018, 4543-4548 ISSN: 1314-3395
12. Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. The german traffic sign recognition benchmark: a multi-class classification competition. In Neural Networks (IJCNN), The 2011 International Joint Conference on, pages 1453–1460. IEEE, 2011.
13. David G. Lowe. Distinctive image features from scale-invariant keypoints. Int. J. Comput. Vision, 60(2):91–110, November 2004.
14. David G. Lowe. Object recognition from local scale-invariant features. In Proceedings of the International Conference on Computer VisionVolume 2 - Volume 2, ICCV '99, pages 1150–, Washington, DC, USA, 1999. IEEE Computer Society.
15. Dollár, P. Piotr's Computer Vision Matlab Toolbox (PMT) [Электронный ресурс]. – URL: <http://vision.ucsd.edu/~pdollar/toolbox/doc/index.html> (request date 26.02.2019).

Приложение А

Распознаваемые классы дорожных знаков (25 классов)

№	Код	Название	Изображение
-1	-	Знак не определён	
0	1.11.1	Опасный поворот (правый)	
1	1.11.2	Опасный поворот (левый)	
2	1.17	Искусственная неровность	
3	1.2	Железнодорожный переезд без шлагбаума	
4	1.20.2	Сужение дороги (справа)	
5	1.22	Пешеходный переход	
6	1.23	Дети	
7	1.25	Дорожные работы	
8	2.3.2	Примыкание второстепенной дороги (справа)	
9	2.3.3	Примыкание второстепенной дороги (слева)	
10	2.5	Движение без остановки запрещено	
11	3.1	Въезд запрещен	
12	3.20	Обгон запрещен	
13	3.24	Ограничение максимальной скорости (20 км/ч)	
14	3.24	Ограничение максимальной скорости (40 км/ч)	
15	3.24	Ограничение максимальной скорости (50 км/ч)	
16	3.27	Остановка запрещена	
17	3.28	Стоянка запрещена	
18	3.4	Движение грузовых автомобилей запрещено	

19	4.1.1	Движение прямо	
20	4.1.2	Движение направо	
21	4.1.4	Движение прямо или направо	
22	4.2.1	Объезд препятствия справа	
23	4.2.2	Объезд препятствия слева	
24	4.2.3	Объезд препятствия справа или слева	