Министерство науки и высшего образования Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий/
Высшая школа программной инженерии.

Работа допущена к защите

Директор ВШ ПИ

П.Д. Дробинцев

« __ »_____2020 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

работа бакалавра

СИСТЕМА ДЕТЕКТИРОВАНИЯ И РАСПОЗНАВАНИЯ НОМЕРНЫХ ЗНАКОВ ТРАНСПОРТНЫХ СРЕДСТВ С ПОМОЩЬЮ НЕЙРОННЫХ СЕТЕЙ

по направлению подготовки (специальности) 09,03,04 Программная инженерия

Направленность (профиль) 09,03.04 01 Технология разработки и сопровождения качественного программного продукта

Выполнил
студент гр. 3530904/60103 А.А. Алиев

Руководитель д.т.н,
профессор С.А. Молодяков

Консультант
по нормоконтролю Е.Г.Локшина

Санкт-Петербург

2020

Министерство науки и высшего образования Российской Федерации Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа программной инженерии

УТВЕРЖДАЮ

Директор ВШ ПИ

П.Д. Дробинцев

« » 20 г.

ЗАДАНИЕ

на выполнение выпускной квалификационной работы

студенту Алиев Али Ахмед оглы, 3530904/60103

фамилия, имя, отчество (при наличии), номер группы

1. Тема работы: Система детектирования и распознавания номерных знаков транспортных средств с помощью нейронных сетей

2. Срок сдачи студентом законченной работы: 01 06.2020

3. Исходные данные по работе: Документация по библиотеке ОрепСУ
Документация по библиотеке PyTorch
Документация по библиотеке NumPy
Документация по Python 3

4. Содержание работы (перечень подлежащих разработке вопросов):
   1) Актуальность темы
   2) Обзор существующих решений
   3) Постановка задачи
   4) Техническая спецификация
   5) Архитектура программы
   6) Программная реализация
   7) Результаты тестирования

5. Перечень графического материала (с указанием обязательных чертежей):

6. Консультанты по работе: _____

7. Дата выдачи задания 06.02.2020 _____

Руководитель ВКР _____ С.А. Молодяков
(подпись) инициалы, фамилия

Задание принял к исполнению 06.02.2020
(дата)

Студент _____ А.А. Алиев
(подпись) инициалы, фамилия

Реферат

На 66 с., 38 рисунков, 4 таблицы.

КЛЮЧЕВЫЕ СЛОВА: КОМПЬЮТЕРНОЕ ЗРЕНИЕ, НЕЙРОННЫЕ СЕТИ, ГЛУБОКОЕ ОБУЧЕНИЕ, СВЁРТОЧНЫЕ НЕЙРОННЫЕ СЕТИ, РЕКУРРЕНТНЫЕ НЕЙРОННЫЕ СЕТИ, ДЕТЕКТИРОВАНИЯ И РАСПОЗНАВАНИЯ НОМЕРНЫХ ЗНАКОВ. OPENCV, PYTORCH.

Тема выпускной квалификационной работы: «Система детектирования и распознавания номерных знаков транспортных средств».

Данная работа направлена на разработку системы детектирования и распознавания номерных знаков транспортных средств. Задачи, которые решались в ходе выполнения работы:

1. Изучение особенностей детектирования и распознавания символов.
2. Сбор и разметка обучающей выборки данных.
3. Обучения нейронных сетей для детектирования, выравнивания и распознавания номерных знаков транспортных средств
4. Разработка системы для обработки видсопотоков для запуска нейронных сетей

В результате работы был получен конкурентоспособный алгоритм для детектирования и распознавания номерных знаков транспортных средств.

Abstract

66 pages., 38 figures, 4 tables.

KEYWORDS: COMPUTER VISION, NEURAL NETWORKS. DEEP LEARNING. CONVOLUTIONAL NEURAL NETWORKS. RECURRENT NEURAL NETWORKS, DETECTION AND RECOGNITION OF LICENSE PLATES, OPENCV, PYTORCH.

The subject of the graduate qualification work is "License plate detection and recognition system".

This work is aimed at developing a license plate detection and recognition system Tasks that were solved during the work:

1. The study of the features of detection and recognition of characters.
2. Collecting and marking of training data sample.
3. Training neural networks for the detection, alignment and recognition of vehicles license plates
4. Development of a system for processing video streams to run neural networks

As a result of the work we got, a competitive algorithm for detection and recognition of vehicles license plates.

Содержание

Введение

В настоящее время вычислительная техника активно внедряется в большинство задач, связанных с автомобильным транспортом. Создаются и совершенствуются системы автопилотов, безопасности движения, учета транспортных средств и др. Одной из актуальных задач является задача распознавания номерных знаков автомобилей.

Известны системы распознавания номерных знаков [1, 2]. В них используются ряд методов, позволяющих проводить распознавание знаков. Последовательность применения методов сводится к следующим этапам: улучшение качества потока поступающих с видеокамеры кадров, сегментирование области расположения номерного знака, распознавание с использованием свёрточной нейронной сети. Известен метод использования инфракрасной камеры с подсветкой. Он используется в ГИБДД. Благодаря известным алгоритмам компьютерного зрения, качественным видео и инфракрасным камерам задача распознавания номерных знаков решается достаточно успешно. Однако данные системы хорошо работают в нормальных условиях, но ряд факторов существенно ухудшают качество распознавания. К таким факторам следует отнести отражение солнечного света и яркого света автомобильных фар, движение автомобиля, загрязнение номера, низкое разрешение видеокамеры, погодные условия. Поэтому задача разработки эффективных программных алгоритмов распознавания номерных знаков автомобилей продолжает быть актуальной.

Целью работы является разработка системы детектирования и распознавания номерных знаков транспортных средств с применением нейронных сетей.

Глава 1. Обзор предметной области

1.1. Задача распознавания номерных знаков транспортных средств

Автоматическое распознавание номерных знаков часто становилось предметом исследований в силу многих практических применений, таких как автоматический сбор платы за проезд, обеспечение соблюдения правил дорожного движения, контроль доступа к частным территориям и мониторинг дорожного движения.
Для решения данной задачи в основном используется программно-аппаратные комплексы, которые являются достаточно дорогими и сложными для интеграции с существующими системами видеонаблюдения. А классические алгоритмы компьютерного зрения не дают такой же точности, как и ранее упомянутые программно-аппаратные комплексы из-за этого данная технология в большинство случаях является недоступной для частных пользователей. Данная работа призвана решить данную проблему .

1.2. Методы детектирования и распознавания номерных знаков транспортных средств

1.2.1. Алгоритмы для детектирования объектов

На сегодняшний существует достаточно большое количество методов компьютерного зрения, которых можно использовать для этой задачи. Алгоритм детектирования номерных является достаточно сложной задачей, так как зависит от таких факторов, как: разрешения камеры, размер самого детектируемого объекта, наличие шумов, чистота номерных знаков, угол наклона камеры относительно объекта и наличие грязи на поверхности номерного знака. Все эти факты очень сильно усложняют работу классических методов компьютерного зрения. Исходя из этого было выбрано два алгоритма детектирования номерных знаков на основе методов компьютерного зрения.

Первый метод не имеет названия. На основе этого метода лежит последовательное выполнение классических методов компьютерного зрения. Алгоритм представлен на рисунке 1.2.1

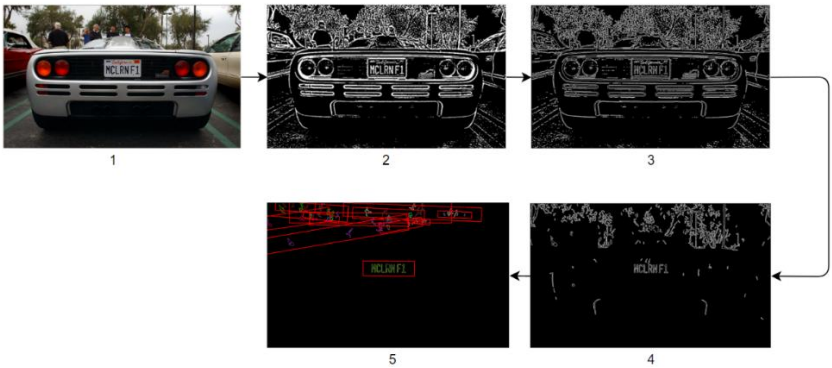


Рис. 1.2.1. Алгоритм детектирования текста.

Представленный алгоритм является универсальным для детектирования любого текста на любом языке. Для реализации этого метода была выбрана библиотека OpenCV. Данный метод состоит из следующих этапов:

1. На вход подается цветное изображения любого разрешения.
2. Конвертация изображения в бинарный вид.

2.1 . Перевод в черно-белый режим.

$$X = 299 \cdot \mathrm{R} + 0.587 \cdot \mathrm{G} + 0.114 \cdot \mathrm{B}$$

2.2 . Применения Гауссово размытие.

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

2.3. Применения метода Оцу[4]. Метод Оцу — это метод определения глобальных пороговых значений. Используется гистограмма изображения для поиска

пороговых значений. И максимизации дисперсии между классами.

$$\sigma_w^2(t) = q_1(t)\sigma_1^2(t) + q_2(t)\sigma_2^2(t)$$

$$q_1(t) = \sum_{i=1}^{t} P(i) \quad \& \quad q_2(t) = \sum_{i=t+1}^{I} P(i)$$

$$\mu_1(t) = \sum_{i=1}^{t} \frac{iP(i)}{q_1(t)} \quad \& \quad \mu_2(t) = \sum_{i=t+1}^{I} \frac{iP(i)}{q_2(t)}$$

$$\sigma_1^2(t) = \sum_{i=1}^{t} [i - \mu_1(t)]^2 \frac{P(i)}{q_1(t)} \quad \& \quad \sigma_2^2(t) = \sum_{i=t+1}^{I} [i - \mu_2(t)]^2 \frac{P(i)}{q_2(t)}$$

3. Нахождения контуров изображения с помощью алгоритма Сузуки.
4. Фильтрация контуров, которые незамкнутые
5. Нахождения контуров, которые находятся последовательно под одним углом и имеют одинаковый размер.

После всех этих шагов мы получаем координаты номерного знака, но данный метод можно обмануть с помощью любых объектов, которые идут последовательно.

Рис. 1.2.2. Пример ложного срабатывания.

На рисунке 1.2.3, алгоритм сработал на декоративных решетках автомобиля. Чтобы избежать этого нужно

распознать, то, что находится внутри этих областей. Для этого нужно перевести изображения в бинарный режим и найти контуры, которые имеют определенный размер относительного всей области. А затем каждый из этих контуров нужно распознать с помощью одного из методов машинного обучения.

Еще одним методом, который выдает удовлетворительные результаты является **каскад Хаара**[5]. Данный алгоритм включает в себя обучения с учителем, поэтому требует достаточно большой выборки данных из положительных и отрицательных классов. В основе этого метода лежат примитивы Хаара, которые представлены на рисунке 1.2.4.

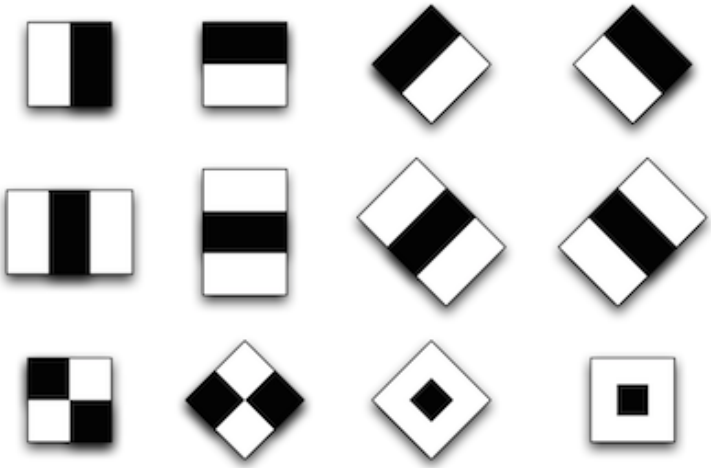


Рис. 1.2.4. Примитивы Хаара

Алгоритм данного метода состоит из следующих шагов:

1. Нужно собрать примитивы Хаара. Функция Хаара рассматривает смежные прямоугольные области в определенном месте в окне обнаружения, суммирует интенсивность пикселей в каждой области и вычисляет

разницу между этими суммами.

$$feature = \sum_{i \in I = 1,\ldots,N} w_i \cdot RectSum(r_i)$$

2. Начинается фаза детектирования, где скользящее окно проходится по изображению. Во время данного этапа мы вычисляем примитивы Хаара.
3. Этап бустинга. Для этого применяется метод AdaBoost[6] (англ. Adaptive Boosting). На вход подается набор данных, где $x_i \in \mathbb{R}^d, y_i \in \{-1, 1\}$.

   Здесь -1 обозначает отрицательный класс, а 1 обозначает положительный. Инициализируем вес для каждой точки данных как:

$$w(x_i, y_i) = \frac{1}{n}, i = 1, \ldots, n.$$

   А затем проходимся по циклу показанной на рисунке 1.2.5

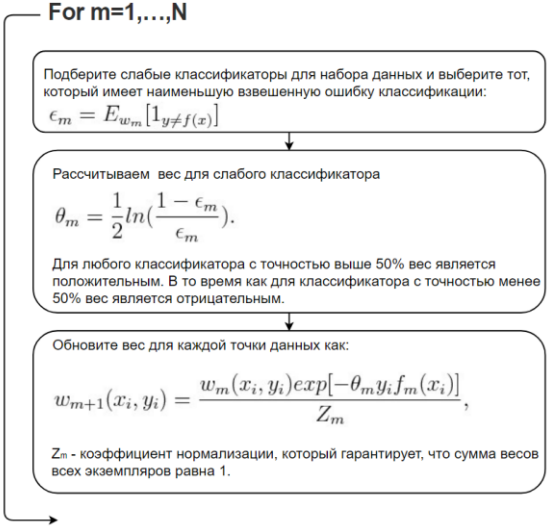


Рис. 1.2.5. Алгоритм Adaboost

После N итерации мы можем получить окончательный прогноз, суммируя взвешенный прогноз каждого классификатора.

4. Дальше идет этап классификации. Каскадный классификатор состоит из набора этапов, где каждый этап представляет собой ансамбль слабых учеников. Слабые ученики — это простые классификаторы, называемые решениями. Каждый этап тренируется с использованием техники бустинга. Бустинг дает возможность обучать высокоточный классификатор, принимая средневзвешенное значение решений, принятых слабым учеником. Каждый этап классификатора помечает регион, определенный текущим местоположением скользящего окна, как положительный или отрицательный. Положительный указывает на то, что объект был найден, а отрицательный указывает на то, что ни один объект не был найден. Если метка отрицательная, классификация этой области завершена, и детектор перемещает окно в следующее место. Если метка положительная, классификатор передает регион на следующий этап. Детектор сообщает об объекте, найденном в текущем местоположении окна, когда последний этап классифицирует регион как положительный. Этапы предназначены для максимально быстрого отбраковки отрицательных образцов. Предполагается, что подавляющее большинство окон не содержат объект интереса. И наоборот, настоящие позитивные объекты встречаются редко и стоит потратить время на проверку

   Чтобы работать хорошо, каждая ступень в каскаде должна иметь низкий уровень ложных отрицательных результатов. Если этап неправильно помечает объект как отрицательный, классификация прекращается, и вы не можете исправить ошибку. Тем не менее, каждый этап может иметь высокий уровень ложных срабатываний. Даже если детектор неправильно помечает не-объект как положительный,

вы можете исправить ошибку на последующих этапах. Добавление большего количества этапов уменьшает общий уровень ложных срабатываний, но также уменьшает общий уровень истинных положительных результатов. Алгоритм работы каскада Хара изображен на рисунке 1.2.6.

Рис.1.2.6. Этапы классификации каскада Хаара.

1.2.2. Алгоритмы машинного обучения для распознавания символов

Для задачи распознавания в основном используется метод опорных векторов или же метод к-ближайших соседей

Метод опорных векторов [7] или SVM — это алгоритм машинного обучения с учителем, который может использоваться как для классификации, так и для регрессии. SVM чаще используется в задачах классификации, и как таковой, это то, на чем мы сосредоточимся в рамках нашей задачи. SVM основаны на идее нахождения гиперплоскости, которая лучше всего делит набор данных на N класса, как показано на рисунке 1.2.7.

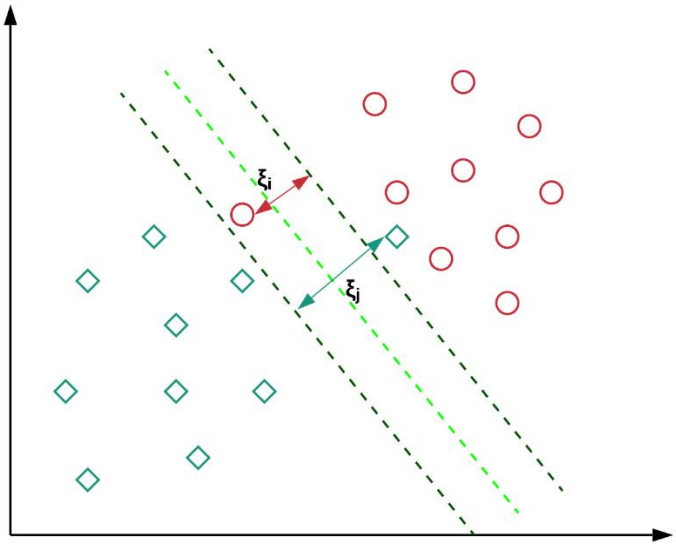


Рис. 1.2.7. Метод опорных векторов.

Наилучшую разделяющую гиперплоскость сего классификатора находят из обстоятельства максимизации ширины промежутка M, но при всем данном позволяется неверно классифицировать некоторую малую группу точек, имеющих отношение к опорным векторам. Для чего задаем вспомогательный условие оптимизации $\sum_j \leq \xi_J^*$, где J – допустимое численность неверных классов. Вслед за тем предоставленная задачка просто сводится к решению задачи квадратичной оптимизации, которая приведет к единственному глобальному минимуму.

Метод к-ближайших соседей [8] (англ. k-nearest neighbors algorithm, KNN) – это алгоритм машинного обучения с учителем. Алгоритм KNN предполагает, что похожие объекты существуют поблизости. Другими словами, похожие объекты близки друг к другу. Визуализация

алгоритма изображена на рисунке 1.2.8

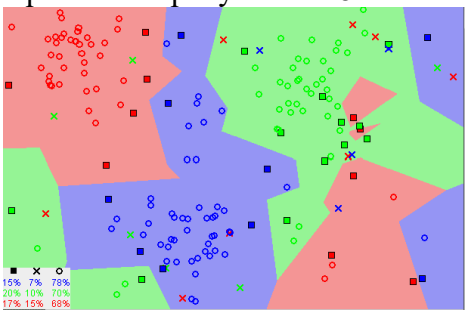


Рис. 1.2.8. Метод k-ближайших соседей.

Обратите внимание на то, что в большинстве случаев похожие точки данных находятся близко друг к другу. KNN использует идею сходства (иногда называемого расстоянием или близостью). Существует множество способов вычисления расстояния, и один из способов может быть предпочтительным в зависимости от решаемой нами задачи. Однако, прямолинейное расстояние (также называемое евклидовым расстоянием) является популярным выбором при использование KNN.

Алгоритм KNN

1. Загрузить данные
2. Инициализировать K в выбранное число соседей
3. Для каждого примера в данных
    3.1 Рассчитать расстояние между образцом запроса и текущим примером из данных.
    3.2 Добавить расстояние и индекс образца.
4. Сортировать упорядоченный набор расстояний и индексов от наименьших до крупнейших (в порядке возрастания) по расстояниям
5. Выбрать первые позиции категории K из отсортированной коллекции
6. Получить метки выбранных позиций K
7. Вернуть моду K классов.

1.3. Использования нейронных сетей для детектирования и распознавания символов.

На сегодняшний день существует огромное количеств разных видов нейронных сетей(прямые, многослойные, глубокие, свёрточные, …) и разных методов их использования. Все их перечислить в рамках этой главы невозможно. Поэтому здесь будет описаны базовые принципы, из которых состоят нейронные сети. Конкретные архитектурные решения, которые были применены в рамках этого проекта будут описаны в третьей главе.

Нейронные сети [9] — это математические модели, которые построены по образу человеческого мозга. Они могут обработать любой вид информации.

Глубокие нейронные сети[10] – это один из видов нейронных сетей, где имеется множество слоев. Каждый слой состоит из множества узлов. Каждый узел выполняет вычисления, узлы контролируются с помощью весов. После каждого узла идет его функция активации. Функция активации отвечает за распространение сигнала данного узла. Пример простой нейронной сети изображен на рисунке 1.3.1.

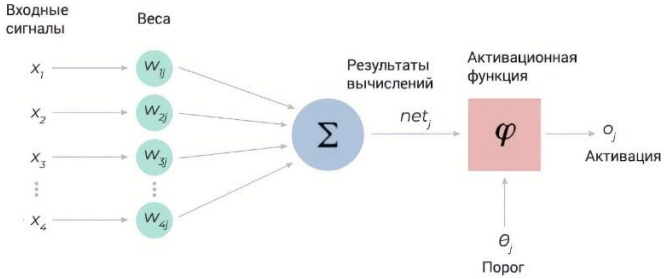


Рис. 1.3.1. Пример однослойной нейронной сети.

Для управления весами узлов используются различные методы оптимизации. Название одной обычно используемой функции оптимизации, которая регулирует весовые коэффициенты в соответствии с вызванной ими ошибкой, называется **градиентный спуск** [11]. Суть данного метода заключается достижении глобального минимума функции, как показана на рисунке 1.3.2.

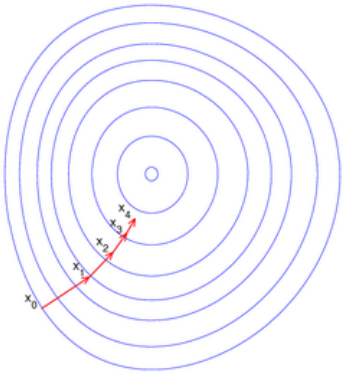


Рис.1.3.2. Градиентный спуск

Функция активации [12] — это математические уравнения, которые определяют выход нейронной сети. Функция прикреплена к каждому нейрону в сети и определяет, должна ли она быть активирована («запущена») или нет, в зависимости от того, имеет ли вход каждого нейрона отношение к предсказанию модели. Ниже на рисунке 1.3.3 представлены некоторые функции активации для нейронных сетей.

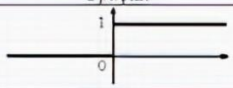
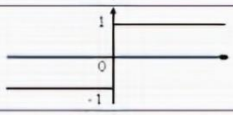
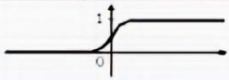
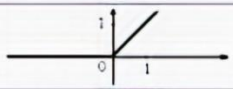
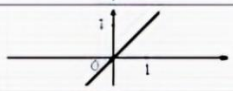
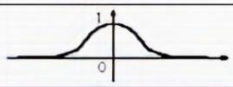
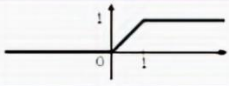
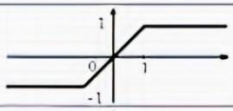
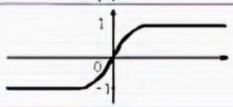
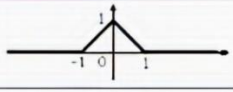
Название	Формула	График
Пороговая	$f(u) = \begin{cases} 0 & u < 0 \\ 1 & u \geq 0 \end{cases}$	
Знаковая (сигнатурная)	$f(u) = \begin{cases} 1 & u > 0 \\ -1 & u \leq 0 \end{cases}$	
Сигмоидальная (логистическая)	$f(u) = \frac{1}{1+e^{-u}}$	
Полулинейная	$f(u) = \begin{cases} u & u > 0 \\ 0 & u \leq 0 \end{cases}$	
Линейная	$f(u) = u$	
Радиальная базисная (гауссова)	$f(u) = e^{-u^2}$	
Полулинейная с насыщением	$f(u) = \begin{cases} 0 & u \leq 0 \\ u & 0 < u < 1 \\ 1 & u \geq 1 \end{cases}$	
Линейная С насыщением	$f(u) = \begin{cases} -1 & u \leq -1 \\ u & -1 < u < 1 \\ 1 & u \geq 1 \end{cases}$	
Гиперболический тангенс (сигмоидальная)	$f(u) = \frac{e^{u} - e^{-u}}{e^{u} + e^{-u}}$	
Треугольная	$f(u) = \begin{cases} 1-\lvert u \rvert & \lvert u \rvert \leq 1 \\ 0 & \lvert u \rvert > 0 \end{cases}$	

Рис.1.3.3. Различные типы функции активации

Сети с глубоким обучением отличаются глубиной от более распространенных нейронных сетей с одним скрытым слоем, то есть количество уровней узлов, через которые должны проходить данные в многоступенчатом процессе распознавания образов. В сетях глубокого обучения каждый слой узлов обучается определенному набору

функций на основе выходных данных предыдущего уровня. Чем дальше вы продвигаетесь в нейронную сеть, тем сложнее объекты, которые могут распознаваться вашими узлами, поскольку они объединяют и комбинируют значения из предыдущего слоя(рисунок 1.3.4).

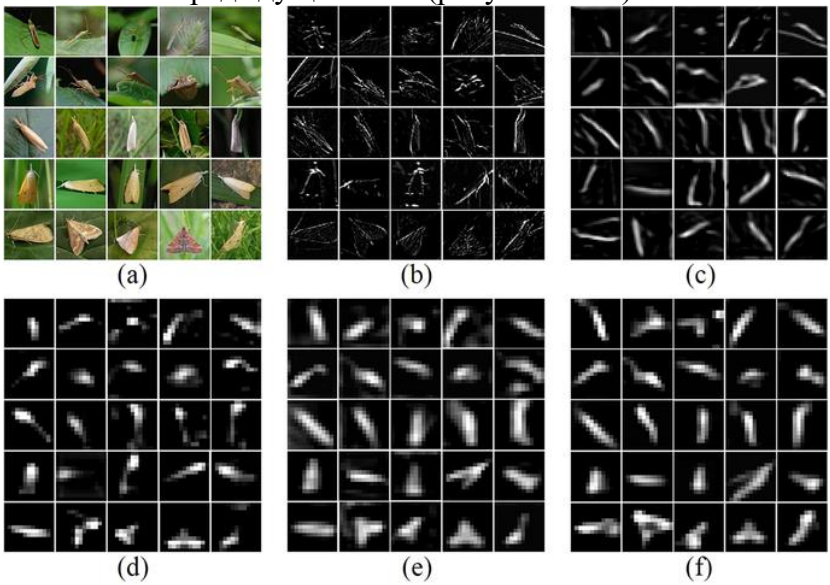


Рис.1.3.4.Визуализация скрытых слоев глубокой нейронной сети.

Все вышеописанное, является только маленькой частью мира нейронных сетей. Все использованные методы и архитектуры будут описаны дальше в соответствующих разделах.

1.4. Обзор существующих реализации для детектирования и распознавания номерных знаков.

1.4.1. OpenALPR

OpenALPR [13] — это библиотека автоматического распознавания номерных знаков, написанная на C ++. Программное обеспечение распространяется как в

коммерческой, так и в версии с открытым исходным кодом. OpenALPR был первоначально разработан командой из двух человек во главе с Мэттом Хиллом. Программное обеспечение с открытым исходным кодом стало доступно для бесплатной загрузки в конце 2015 года. В марте 2016 года OpenALPR запустил платный облачный сервис API.

Бесплатная версия OpenALPR, представляет собой комбинацию каскада Хаара для детектирования и Tesseract OCR [14].

Движок Tesseract изначально разрабатывался как проприетарное программное обеспечение в лабораториях Hewlett Packard в Бристоле, Англия и Грили, штат Колорадо, между 1985 и 1994 годами, с некоторыми другими изменениями, сделанными в 1996 году для переноса на Windows, и некоторыми переходами с C на C ++ в 1998 году. кода был написан на C, а затем еще несколько был написан на C ++. С тех пор весь код был преобразован, по крайней мере, для компиляции с помощью компилятора C ++. В следующем десятилетии было проделано очень мало работы. Затем он был выпущен как открытый исходный код в 2005 году Hewlett Packard и Университетом Невады, Лас-Вегас (UNLV). Спонсором разработки Tesseract является Google с 2006 года.

На данный момент Tesseract использует LSTM [15] в своей архитектуре. Долгая краткосрочная память — это архитектура искусственной рекуррентной нейронной сети (RNN) [16] , используемая в области глубокого обучения. В отличие от стандартных нейронных сетей с прямой связью, LSTM имеет соединения обратной связи. Он может обрабатывать не только отдельные точки данных (например, изображения), но и целые последовательности данных (например, речь или видео). Например, LSTM применим к таким задачам, как несегментированное распознавание рукописного ввода, распознавание речи и

обнаружение аномалий в сетевом трафике или IDS (системы обнаружения вторжений).
Обычный модуль LSTM состоит из ячейки, входного вентиля, выходного вентиля и логического элемента забытия. Ячейка запоминает значения в произвольные промежутки времени, а три входа регулируют поток информации в ячейку и из нее.
В репозитории OpenALPR содержаться конфигурационные файлы для каскада Хаара и для Tesseract OCR. На данный момент поддерживаются номерные знаки следующих стран:

- США
- Австралия
- Бразилия
- ЕС
- Вьетнам
- Корея

Плюс данного решения является то, что он написан на C++ и имеет поддержку всех основных ОС на сегодняшний день. Платная версия OpenALPR представляет собой локальное или серверное решения. Платное решения поддерживает номерные знаки следующих стран:

- Аргентина
- Австралия
- Бразилия
- Китай
- ЕС
- Великобритания
- Индия
- Индонезия
- Япония
- Корея
- Малазия
- Новая Зеландия

- США
- Канада
- Россия
- Сингапур
- Северная Африка
- Тайланд

И в отличие от бесплатной версии работает гораздо лучше. К сожалению, нету информации об архитектурных решениях, которые используются в платной версии. В таблице 1.1 и 1.2 приведены результаты тестов на различных платформах.

Процессор	1080p	720p	480p
Xeon E5-2666 v3 @ 2.9 ГГц	12.7 К/С	15.0 К/С	16.8 К/С
Core i5-5250U @ 1.6 ГГц	17.4 К/С	20.4 К/С	22.9 К/С
Xeon E7-8880 v3 @ 2.3 ГГц	20.1 К/С	23.4 К/С	26.2 К/С
Core i7-7700K @ 4.2 ГГц	52.4 К/С	61.8 К/С	69.9 К/С
Core i7-8750H @ 2.2 ГГц	53.3 К/С	60.8 К/С	67.3 К/С
Xeon Platinum 8124M @ 3.0 ГГц	203.8 К/С	236.0 К/С	275.5 К/С

Таблица 1.1

Такой большой прирост производительности на процессоре Xeon Platinum 8124M связана с поддержкой новой инструкции AVX-512, которая ускоряет обработку больших данных.

Графический процессор	1080p	720p	480p

NVIDIA Jetson TX-1	6.2 К/С	15К/С	44 К/С
NVIDIA Jetson TX-2	14 К/С	34 К/С	86 К/С
NVIDIA Tesla M60	99 К/С	164 К/С	202 К/С
NVIDIA GeForce GTX 1060	165 К/С	191 К/С	206 К/С
NVIDIA Tesla V100	184 К/С	329 К/С	364 К/С
NVIDIA GeForce RTX 2080	221 К/С	297 К/С	351 К/С

Таблица 1.2

Как мы видим, процессоры последнего поколения Xeon догоняют графические процессоры по производительности. Но стоит отметить, что процессоры Xeon Platinum стоят дороже NVIDIA GeForce RTX 2080, но при этом дешевле серверного решения NVIDIA Tesla V100. Поэтому я бы рекомендовал использовать Xeon Platinum при использование серверного решения, а RTX 2080 при локальном использовании. Данное решение связано с тем, что NVIDIA запрещает ставит видеокарты серии GeForce в дата-центры и сервера.

Кроме информации о номерных знаках, OpenALPR так же выдает следующую информацию:

- Цвет автомобиля
- Марка
- Модель
- Тип автомобиля
- Расположения камеры относительно автомобиля
- Год выпуска

Результаты работы изображены на рисунке 1.4.1.

Рис.1.4.1. Результаты работы OpenALPR

Для каждой камеры при локальном установке ПО, нужно заплатить 49$. В облачной версии нужно заплатить 99$, 395$ или 1995$ за 25 000, 125 000 или 1 000 000 распознавания номерных знаков в месяц соответственно.

1.4.2. Plate Recognizer

Создавался как более удобная и дешевая альтернатива для OpenALPR. По функциональности не отличается от OpenALPR. Поддерживает абсолютно все тоже самое, что и OpenALPR. К сожалению, тоже отсутствует информация об архитектуре. Но судя по скорости работы, вместо детектирования через компьютерное зрения используются свёрточные нейронные сети. В таблице 1.3 будут приведены результаты скорости работы программы.

Скорость работы	**720p**
n1-standard-4 (4 vCPUs, 15 GB RAM),1 x NVIDIA Tesla T4	22 К/С
n2-standard-8 (8 vCPUs, 32 GB RAM)	23 К/С
c2-standard-4 (4 vCPUs, 16 GB RAM)	17 К/С
n2-standard-4 (4 vCPUs, 16 GB RAM)	15 К/С
n1-standard-4 (4 vCPUs, 15 GB RAM)	10 К/С
n2-standard-2 (2 vCPUs, 8 GB RAM)	8 К/С
e2-standard-2 (2 vCPUs, 8 GB RAM)	7 К/С

Intel Core i7-8550U CPU @ 1.80ГГц	18 К/С
Raspberry Pi 3/4	0.7 К/С
LattePanda Alpha	6 К/С
LattePanda V1	0.75 К/С
Nvidia Jetson Nano	3 К/С

Можно заметить, что Plate Recognizer [17] намного хуже оптимизирован и уступает по скорости OpenALPR. Данная система умеет работать с номерными знаками 95 стран.

Так же в отличие от OpenALPR, данная система лучше работает. Так же сама умеет определять регион номерного знака и включить соответствующий алгоритм распознавания. С помощью рисунков 1.4.2 – 1.4.4 мы можем сравнить результаты работы OpenALPR и Plate Recognizer

Рис.1.4.2. Результаты работы OpenALPR

Рис. 1.4.3. Результат работы Plate Recognizer

Рис. 1.4.4. Результаты работы OpenALPR

Рис. 1.4.4. Результат работы Plate Recognizer

Как можно увидеть результаты сравнения неоднозначны.

Нельзя сказать, какой из продуктов работает лучше при относительно сложных ситуациях.

Бесплатно	75$ в месяц	225$ в месяц	375$ в месяц
2500 номерных знаков	50000 номерных знаков	25000 номерных знаков	500000 номерных знаков

Данные цены актуальны для локального и облачного решения. В отличие от OpenALPR, Plate Recognizer стоит дешевле и имеет более удобный интерфейс и способы взаимодействия с облаком для распознавания. Но все зависит от конкретного случая. Если у вас много камер, на которых редко появляются автомобили, то лучше всего здесь подходит Plate Recognizer. Так как в этом продукте вы платите за количество распознаваний. А если же у вас несколько камер и при это вам нужно распознать большое количество автомобилей, то здесь лучше всего подойдет OpenALPR. Так как в данном случае вы платите за каждую камеру, и сама система работает намного быстрее, что в свою очередь уменьшает расходы на оборудования для распознавания.

Глава 2.Сбор и подготовка данных

2.1. Поиск выборки данных для задачи распознавания номерных знаков

На данный момент в открытом доступе имеется множество данных с изображениями автомобилей с номерными знаками и их координатами расположения на изображении. Но, к сожалению, ни один из этих наборов данных не содержит в себе российские номерные знаки. Поэтому требовалось самостоятельно собрать и разметить набор данных для детектирования и распознавания номерных знаков.
Естественно, вручную собрать такой большой набор данных невозможно. Поэтому существует 2 способа сделать это:

1. Выполнив разные запросы в поисковых, система и с помощью технологии веб-парсинга скачивать изображения.
2. Найти веб-сайт по данной тематике и написать собственный веб-парсер для скачивания нужных номерных знаков.

К сожалению, первый способ не очень хорошо подходит для задачи получения большого набора данных. Так как, выполнив разные запросы есть вероятность получения повторенных и нерелевантных изображений. Из-за этого приходится вручную фильтровать полученный набор данных. К тому же, поисковые системы ограничивают количество одновременных запросов с одного устройство, что сильно замедляет скорость скачивания данных.
Оптимальным способ является найти веб-сайт, где можно получить доступ к изображениям с номерными знаками. Одним из таких сайтов является avto-nomer.ru [18](изображённую на рисунке 2.1.1.).

Рис.2.1.1 .Главная страница веб-сайта avto-nomer.ru

На данный момент в этом веб-сайте доступно около 13.5 миллионов изображений автомобилей с номерными знаками, из которых 9.5 принадлежат российским номерным знакам.

2.2. Изучения структуры веб-сайта avto-nomer для веб-парсинга

Перейдя по ссылке avto-nomer.ru, мы попадаем на главную странице. Далее мы выбираем Россию и попадаем на следующую страницу, изображённую на рисунке 2.2.1.

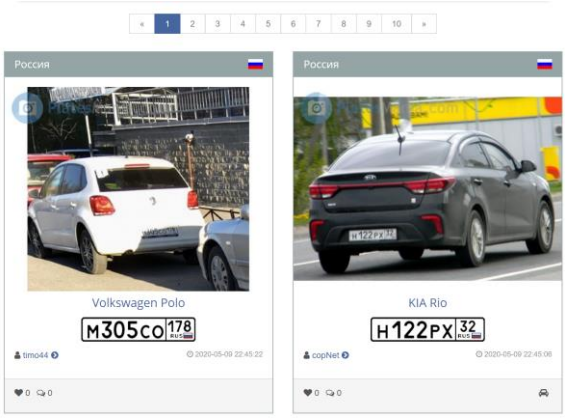


Рис.2.2.1. Список с изображениями

Как может показаться с первого, что мы уже можем начать скачивать изображения с данной страницы, для этого есть вся информация. Но здесь находятся миниатюры изображении, что указывает, на то, что это сжатые версии оригинальных изображений.

Если нажать на один из изображений, мы попадаем на отдельную страницу с этим постом. Ссылка (*https://avto-nomer.ru/ru/nomer14678954*) на каждый из этих постов содержит в себе свой идентификационный номер(*14678954*), это значит, что, мы можем пройтись циклом по всем возможным постам. Еще раз нажав на изображение, мы получаем доступ к ссылке на оригинальное изображение (*https://avto-nomer.ru/ru/foto14678954*). Но скачать это невозможно, так как для скачивания нужно найти ссылку на изображения в одном из доступных форматов (jpg, png, bmp, …). Анализировав сгенированный HTML-код страницы, можно обнаружить ссылку такого формата:

- *http://img03.platesmania.com/200509/o/14678954.jpg*

Это то, что нам нужно. Теперь изучив структуру веб-сайта, можно приступать к разработке веб-парсера для скачивания изображений с данного веб-сайта.

2.3. Разработка веб-парсера для скачивания изображений

Веб-парсер был написан на языке Python 3.6. Алгоритм работы программы изображен на рисунке 2.3.1

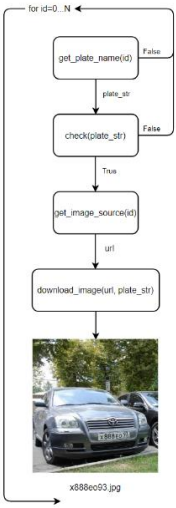


Рис.2.3.1. Алгоритм работы веб-парсера

1. Сначала делаем по запрос по https://avto-nomer.ru/ru/nomer + id. Если в ответ мы получили не пустое сообщение, то продолжаем. В ином случае пропускаем данный элемент цикла.
2. Затем проверяем полученную строку через регулярное выражение:
   [А-я][0-9][0-9][0-9][А-я][А-я][0-9][0-9]
   Данное регулярное выражение описывает российские номерные знаки. Если строка не проходит через данное регулярное выражение, то пропускаем данный элемент цикла.
3. Получаем ссылку на изображение через страницу *https://avto-nomer.ru/ru/foto* + id
4. Скачиваем изображения и сохраняем с названием plate_str.jpg

2.4. Запуск веб-парсера и решения проблем

Цикл в веб-парсере был запущен с помощью библиотеки MultiProccesing.Pool, чтобы добавить поддержку многопроцессорной обработки данных. Каждый из потоков процессора занимался обработкой одного из элементов цикла веб-парсера.

Программа была запущена на компьютере с процессором Intel Core i7 6700k, с частотой 4.5 ГГц и оперативной память DDR4 16 ГБ с частотой 2.9 ГГц. При первом запуске получилось 50-60 итерации в минуту, что показалось подозрительно низким для такого процессора. После мониторинга компонентов системы было обнаружено, что узким горлышком послужило жесткий диск компьютер. Он был на 100%, при это процессор был загружен примерно 30%. После изменения место сохранения изображений на твердотельный накопитель была получена скорость скачивания около 250 изображений в минуту. Следующим узким горлышком послужило ограничения скорости интернет-соединения равное 100 Мбит/с. При этом уже процессор был загружен на 90%. К сожалению, изменить скорость интернет-соединения было невозможно.

Еще одним любопытным фактом является, что веб-сайт не имеет автоматической системы против веб-парсинга, что очень сильно упростило процесс разработки веб-парсера.

В результате работы программы в течение 6 часов было получено 103 000 изображений.

2.5 Подготовка данных для обучения нейронных сетей

Для начала обучения детектора объектов, нам нужно имеет информацию а расположении номерного знака на изображении. Данные должны имеет следующий формат $\{x_1, y_1, x_2, y_2\}$, где x_1 и y_1 координаты левого верхнего угла прямоугольника, x_2 и y_2 правого нижнего угла прямоугольника.

Но вручную разметить такой большой набор данных заняло бы сотню часов работы. Для упрощения работы был выбран метод псевдо-разметки [19].
Общий алгоритм метода изображен на рисунке 2.5.1.

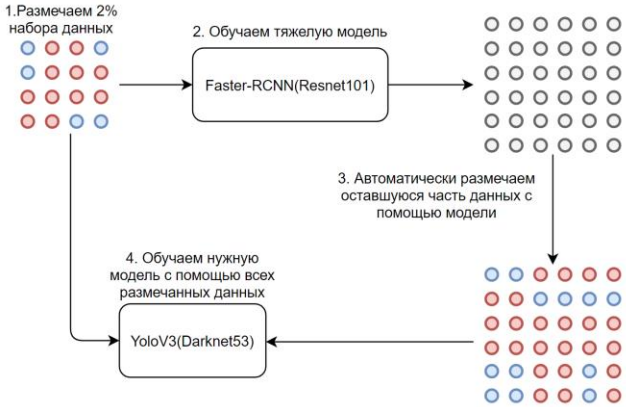


Рис.2.5.1. Схема работы метода псевдо-разметки

1. Размечаем небольшую часть (2%) данных из общего набора.
2. На основе размеченных данных обучаем тяжелую модель Faster-RCNN
3. С помощью полученной модели размечаем в автоматическом режиме все оставшуюся часть данных.
4. Объединяем наборы данных и обучаем нужную модель.

Для разметки изображений было использовано инструмент Supervisely. Данный веб-сайт позволяет загрузить изображения на собственные сервера и размечать данные с любой точки планеты. Интерфейс представлен на рисунке 2.5.2

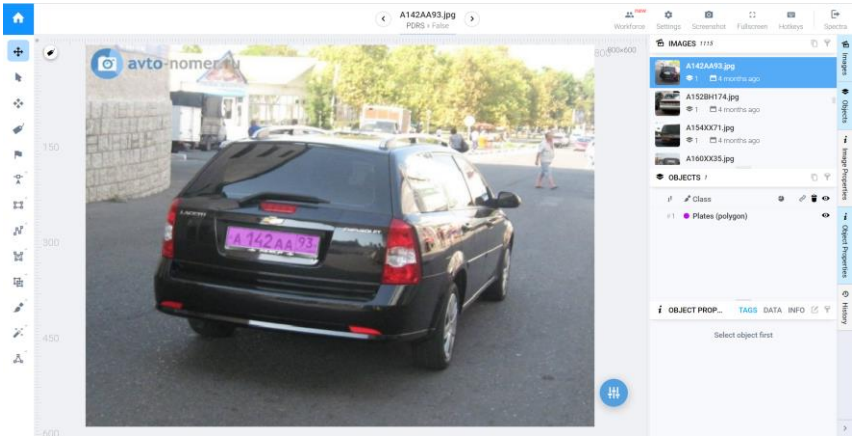


Рис.2.5.2. Интерфейс Supervisely

А в качестве модели для разметки используется Faster-RCNN с Resnet-101 [21]. Faster RCNN — это архитектура детектирования объектов, представленная Россом Гиршиком, Шаоцингом Реном, Каймингом Хе и Цзянем Саном в 2015 году, и одна из известных архитектур детектирования объектов, использующая свёрточные нейронные сети, такие как YOLO [22] и SSD [23]. Архитектура Resnet-101 и Faster-RCNN представлена ниже на рисунках 2.5.3 и 2.5.4 ссоответственно

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Рис.2.5.3. Архитектура Resnet

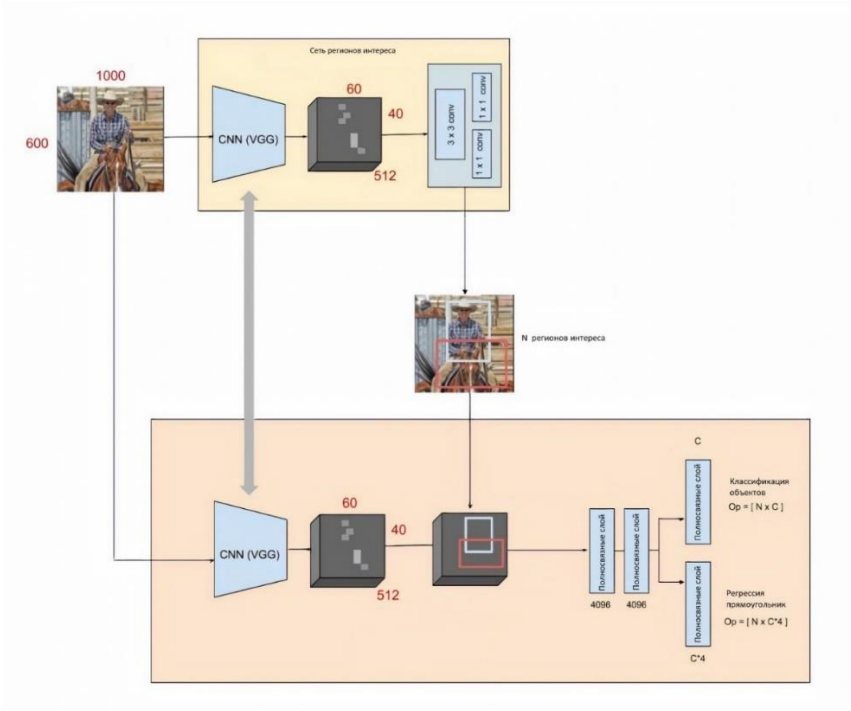


Рис.2.5.4.Архитектура Faster-RCNN

В архитектуре Faster-RCNN была заменена свёрточная сеть VGG на Resnet-101. Так как Resnet-101 намного лучше по соотношению точности и скорости работы по сравнению с VGG. На рисунке 2.5.5 приведены графики с результатами сравнения этих моделей.

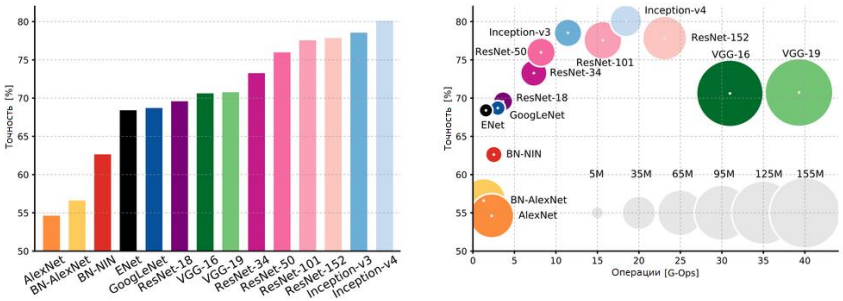


Рис.2.5.5. Сравнение Resnet-101 и VGG-19

После всех этих этапов мы получил 94 000 изображений с размеченным координатами номерных знаков на них. Метод псевдо-разметки позволило нам сэкономить огромное количество времени на этапе разметки данных, при этом важно отметить, что не пострадало качество разметки данных.

Глава 3. Разработка алгоритмов для детектирования и распознавания номерных знаков

Система детектирования и распознавания будет состоять из следующих частей:

1. Детектор объектов для нахождения координат номерных знаков.
2. Детектор ключевых точек, чтобы выравнивать изображения номерного знака.
3. Рекуррентно-свёрточная нейронная сеть для распознавания символов номерных знаков.

3.1 Детектор объектов

Ранее при использовании метода псевдо-разметки мы использовали архитектуру Faster-RCNN. Но, к сожалению, данная архитектура требует достаточно больших ресурсов. На видеокарте NVIDIA GTX 1070ti с входным разрешением картинки 800x600 можно обработать только 3.5 кадра в секунду. Данные показатели не дают нам возможности сделать систему детектирования в реальном времени. Для решения этой проблемы была выбрана архитектура YoloV3. Вы смотрите только один раз (англ. Yolo - You Look Only Once) – это архитектура для детектирования объектов.

В отличие от архитектуры Faster-RCNN в YoloV3 применяется абсолютно другой подход для детектирования. Если в случае Faster-RCNN, еще до детектирования происходит выделения областей интереса, то в YoloV3 нету такого этапа. Сеть разделяет изображение на регионы и предсказывает ограничивающие рамки и вероятности для каждого региона. Такая модель имеет ряд преимуществ перед архитектурами на основе классификаторов. Он просматривает всё изображение при запуске, прогнозирование происходит на основе всего изображения, а не одного участка.

YOLO использует только свёрточные слои, что делает его полностью свёрточной сетью (FCN). В статье YOLOv3 авторы представляют новую, более глубокую архитектуру экстрактора функций под названием Darknet-53. Как следует из названия, он содержит 53 свёрточных слоя,

каждый из которых сопровождается уровнем пакетной нормализации и активацией Leaky ReLU. Форма объединения не используется, а свёрточный слой с шагом 2 используется для уменьшения выборки карт объектов. Это помогает в предотвращении потери функций низкого уровня, часто приписываемых объединению. Арихтектура экстрактора Darknet-53 отображена на рисунке 3.1.1.

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

Рис.3.1.1. Архитектура Darknet-53

YOLO не зависит от размера входного изображения. Однако на практике, желательно придерживаться постоянного размера ввода из-за различных проблем, которые проявляются только при реализации алгоритма. Проблема заключается в том, что, если мы хотим обрабатывать наши изображения в пакетном режиме (изображения в пакетном режиме могут обрабатываться

параллельно графическим процессором, что приводит к увеличению скорости), нам нужно иметь все изображения фиксированной высоты и ширины. Это необходимо для объединения нескольких изображений в большой пакет.

Сеть сокращает изображение с помощью фактора, называемого шагом сети. Например, если шаг сети равен 32, тогда входное изображение размером 416 x 416 даст выход размером 13 x 13. Как правило, шаг любого слоя в сети равен коэффициенту, с помощью которого выходной сигнал слой меньше, чем входное изображение в сеть.

Как и во всех детекторах объектов, функции, изученные свёрточными слоями, передаются в классификатор / регрессор, который делает прогноз обнаружения (координаты ограничивающих рамок, метка класса и т. д.). В YOLO прогнозирование выполняется с использованием свёрточного слоя, который использует свертки 1 x 1. Итак, первое, на что нужно обратить внимание, это наш вывод, это карта объектов. Поскольку мы использовали свертки 1 x 1, размер карты прогнозирования в точности соответствует размеру карты объектов до нее. В YOLO v3 способ интерпретации этой карты прогнозирования состоит в том, что каждая ячейка может предсказать фиксированное количество ограничивающих рамок. Например, если у нас есть (B x (5 + C)) записей в карте объектов. B представляет количество ограничивающих рамок, которые каждая ячейка может предсказать. Согласно статье, каждый из этих ограничивающих прямоугольников B может специализироваться на обнаружении объекта определенного типа. Каждый из ограничивающих прямоугольников имеет атрибуты 5 + C, которые описывают координаты центра, размеры, вероятность наличие объекта и вероятность отношения класса C для каждого ограничивающего прямоугольника. На рисунке 3.1.2 показан наглядный пример работы данного алгоритма.

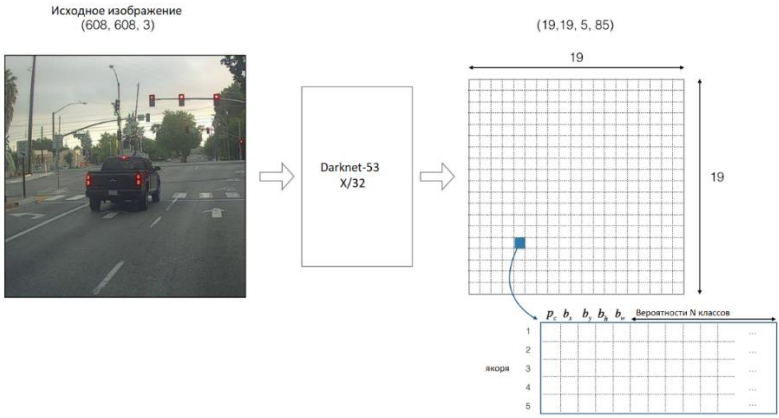


Рис.3.1.2. Архитектура детекторной части YoloV3

Теперь для каждого блока (каждой ячейки) мы вычислим вероятность того, что в блоке содержится определенный класса, как показана ниже на рисунке 3.1.3.

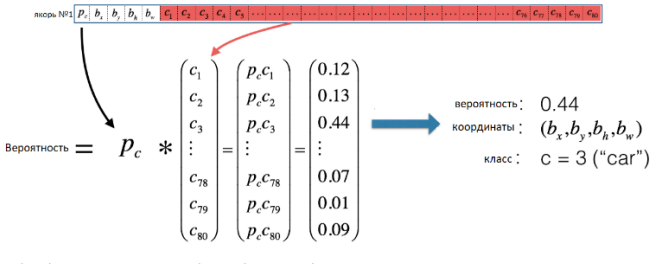


Рис.3.1.3. Нахождения класса объекта в ячейке

Данный алгоритм мы используем для каждого якоря. YOLO v3 имеет три якоря на для каждого этапа, которые приводят к предсказанию трех ограничивающих рамок на ячейку. Якоря - это своего рода ограничивающие априорные рамки, которые были рассчитаны на основе нашего набора данных

с использованием кластеризации k-средних. Мы собираемся предсказать ширину и высоту блока как смещения от центроидов кластера. Координаты центра поля относительно местоположения приложения фильтра прогнозируются с использованием сигмоидальной функции. Следующая формула описывает, как преобразовывается выходной сигнал сети для получения предсказаний ограничивающего прямоугольника:

$$b_x = \sigma(t_x)c_x$$
$$b_y = \sigma(t_y)c_y$$
$$b_w = p_w e^{t_w}$$
$$b_h = p_h e^{t_h}$$

Здесь bx, by, bw, bh - координаты центра x, y, ширина и высота нашего прогноза. tx, ty, tw, th - это то, что выводит сеть. cx и cy - верхние левые координаты сетки. pw и ph - размеры якорей для ячейки.

Выше описанная методика детектирования применяется три раза. Каждый из этапов взаимосвязнные и служат для детектирования объектов определенного размера.Подытоживая все вышеперечисленное, мы получаем архитектуру указанное на рисунке 3.1.4.

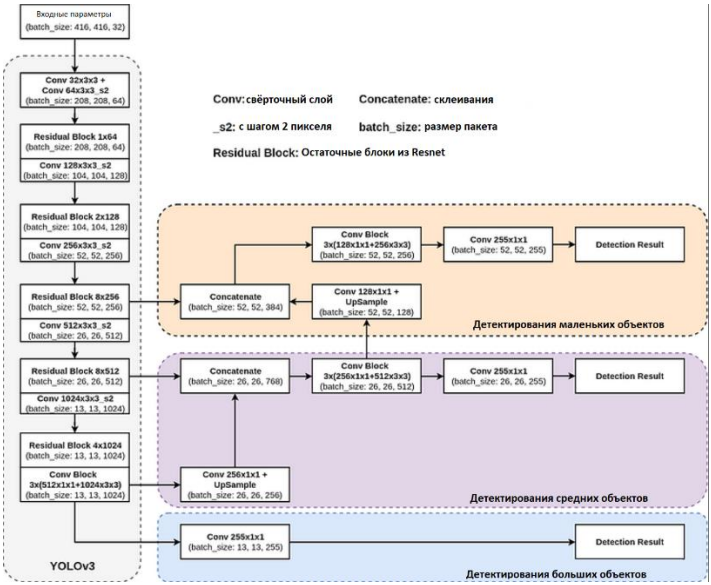


Рис.3.1.4. Архитектура YoloV3

Слой детектирования используется для обнаружения на картах характеристик трех разных размеров, имеющих фактор уменьшения 32, 16, 8 соответственно. Это означает, что при входе 416 x 416 мы делаем детектирование в 13 x 13, 26 x 26 и 52 x 52, как показана на рисунке 3.1.5.

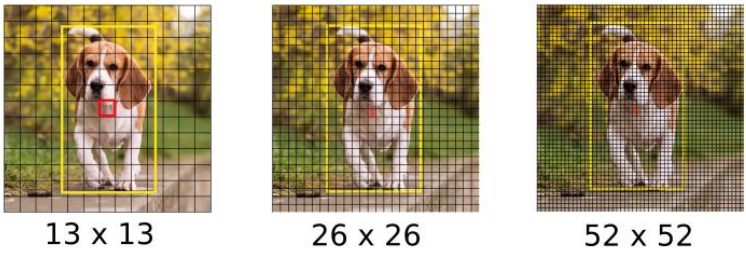


Рис.3.1.5. Этапы перед детектированием.

Каждая ячейка предсказывает 3 ограничивающих прямоугольника с использованием 3 якорей, в результате чего общее количество используемых якорей равно 9.

Авторы YoloV3 сообщают, что это помогает улучшить обнаружение небольших объектов.
Для изображения размером 416 x 416 YoloV3 предсказывает ((52 x 52) + (26 x 26) + 13 x 13)) x 3 = 10647 ограничивающих рамок. Однако в случае с нашим изображением есть только один объект - собака. Итак, как мы можем уменьшить количество рамок с 10647 до 1?
Во-первых, мы фильтруем блоки на основе их вероятности. Как правило, поля с оценками ниже порогового значения (например, ниже 0,5) игнорируются. Далее, Non-Maximum Suppression (NMS) [25] решает проблему множественных обнаружений одного и того же объекта. NMS – это функция подавления не-максимумов, которая использует очень важную метрику в детектирование объектов , коэффициент Жаккара изображенная на рисунке 3.1.6.

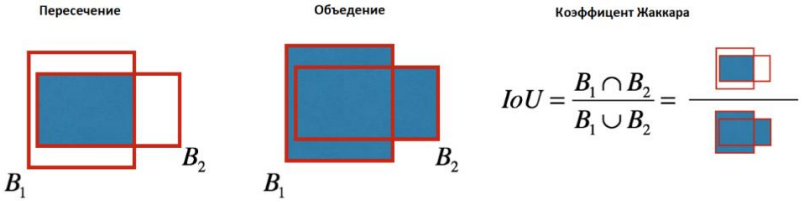


Рис.3.1.6. Коэффициент Жаккара

Алгоритм подавления не-максимумов состоит из следующих этапов:

1. Выбираем рамку с максимальной вероятностью
2. Вычислите его перекрытие со всеми другими блоками и удалите блоки, которые перекрывают его больше, чем установленное пороговое значение.
3. Вернитесь к шагу 1 и выполняйте итерацию до тех пор, пока не останется больше рамок с более низким показателем, чем текущий выбранный.

В итоге, наша модель должна работать следующим образом:

1. Подаем входное изображение (608, 608, 3).
2. Входное изображение проходит через Darknet-53, что приводит к (19,19,5,85) размерному выводу.
3. После выравнивания двух последних измерений на выходе получается объем формы (19, 19, 54):
   3.1. Каждая ячейка в сетке 19x19 над входным изображением дает 51 чисел.
   3.2. 51 = 9 x 6, потому что каждая ячейка содержит прогнозы для 5 блоков, соответствующих 9 якорям.
   3.3. 6 = 5 + 1, где 5, потому что (pc, bx, by, bh, bw) имеет 5 чисел, а 1 — это количество классов, которые мы хотели бы обнаружить
4. Затем мы выбираем только несколько рамок на основе:
   4.1. Оценка пороговых значений: фильтруем поля, в которых обнаружен класс с оценкой ниже порогового значения.
   4.2. Подавления не-максимумов.

5. Получаем результаты в формате: номер класса, вероятность, координаты.

Вот и все, что касается теории. В следующей главе будет описана реализация YoloV3.

3.2 Детектор ключевых точек

Данная нейронная сеть была призвана решить проблему, которая возникает при использование скользящего окна в предыдущей нейронной сети для детектирования. Дело в том, что если номерные знаки находятся под большим углом, то нейронная сеть для распознавания не может распознать символы на нем в связи с архитектурными особенностями, про которые будет рассказано в следующей подглаве. Для решения данной задачи было решено использовать детектор ключевых точек, которая должна выдать четыре ключевых точек номерного знака, как показан на рисунке 3.2.1.

Рис. 3.2.1. Ключевые точки

Так как мы используем детектор ключевых точек после детектора объектов, то это сильно упрощает нашу задачу, к тому же сами ключевые точки находятся на самом простом месте для обнаружения, на угловых точках светоотражающей части номерного знака. Это позволяет нам создать достаточно простую свёрточную нейронную сеть, как на рисунке 3.2.2, чтобы справиться с этой задачей.

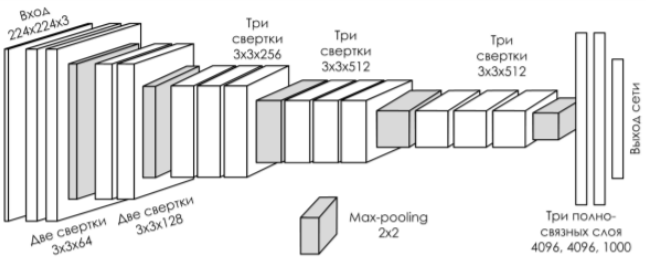


Рис.3.2.2. Архитектура LeNet

Мы будем использовать классическую архитектуру свёрточной нейронной сети, которая состоит свёртки, функции максимума и полносвязных слоев. Такая архитектура называется LeNet[27]. LeNet – это архитектура сверточной нейронной сети, предложенная Yann LeCun et al. в 1998 году.

3.3 Распознавание символов

OCR расшифровывается как «Оптическое распознавание символов». Это широко распространенная технология распознавания текста внутри изображений, таких как отсканированные документы и фотографии. Технология

OCR используется для преобразования практически любого вида изображений, содержащих письменный текст (набранный, рукописный или напечатанный), в машиночитаемые текстовые данные.
Для решения данной задачи с помощью нейронных сетей существует два основных метода:

1. Использование детектора объектов, чтобы найти каждый символ.
2. Использование комплексного решения, которое объединяет свёрточные и рекуррентные нейронные сети.

Первый метод хорошо подходит, если у вас небольшой набор данных и фразы, которые распознаются не имеют ярко выраженного паттерна. Для этого метода нужно разметить каждый символ (обвести прямоугольником и обозначить принадлежность к одному из классов). Данный метод плохо подходит, если у вас большой набор данных, как в нашем случае. Мы имеем 95 000 изображений номерных знаков, в которых в среднем 8 символов. Это значит, что нам бы пришлось разметить около 760000 символов. Поэтому было решено отказаться от данного метода распознавания символов.
Для решения проблемы разметки символов, было решено найти решения, при котором не нужно размечать текущий набор данных. Одним из таких методов является использование свёрточно-рекуррентных нейронных сетей.
Рекуррентные нейронные сети (RNN) – это один из видов нейронных сетей, где нейронная сеть обучает работать с последовательностью данных и находит паттерны в данных. Просто так комбинировать последовательно свёрточные и рекуррентные сети нельзя, они обязательно должны быть связаны с друг-другом в пределах одной модели. Другие решения, которые используют рекуррентные нейронные сети, в которых предварительная обработка не зависит от последующих компонентов в

конвейере, поэтому существующие системы на основе RNN не могут быть обучены и оптимизированы сквозным образом.

Такая модель называется свёрточно-рекуррентная нейронная сеть(CRNN). Данная является комбинацией свёрточной и рекуррентной нейронных сетей. Для CRNN не нужно размечать каждый элемент последовательности отдельно.

Архитектура CRNN, как показано на рисунке 3.3.1, состоит из трех компонентов, включая сверточные уровни, рекуррентные уровни и уровень транскрипции, снизу вверх.

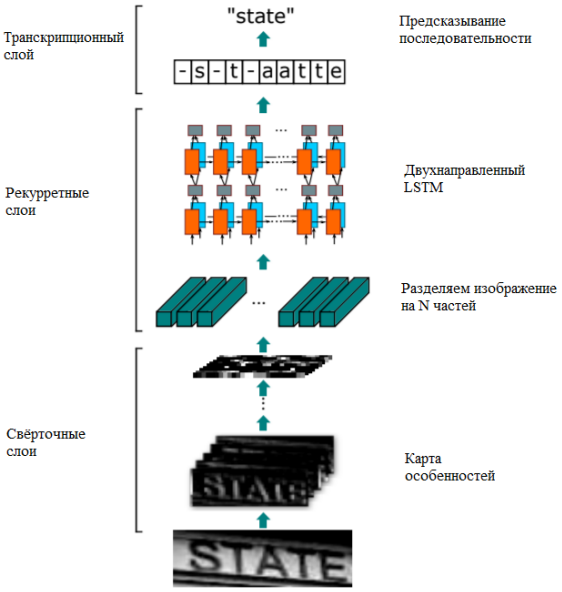


Рис.3.3.1. Архитектура CRNN

Поверх сверточной сети построена рекуррентная сеть для прогнозирования каждого кадра последовательности признаков, выводимого сверточными уровнями. Преимуществом CRNN является то, что сеть обучается одной функцией потери, поэтому все части сети связаны с

друг-другом. Далее будет более подробно описан каждый этап CRNN.

3.3.1 Получения дескрипторов областей изображения

В модели CRNN компонент сверточных слоев строится путем извлечения сверточных слоев и слоев с максимальным объединением из стандартной модели CNN без полносвязных слоёв. После прохождения свёрточной части, дескриптор изображения делиться на N частей. Как показано на рисунке 3.3.2, каждый вектор в последовательности признаков связан с рецептивным полем и может рассматриваться как дескриптор изображения для этой области. Это помогает нам избавиться от процесса разметки каждого изображения, достаточно знать, что написано на изображении.

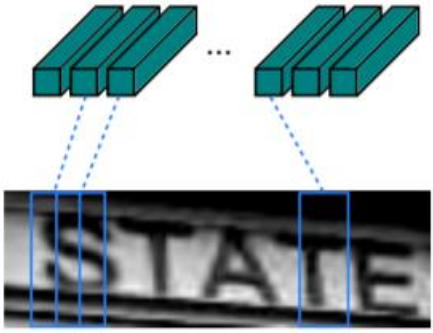


Рис. 3.3.2. Получения дескрипторов областей

3.3.2 Обработка последовательности дескрипторов

Глубокая двунаправленная рекуррентная нейронная сеть является одним из основных видов рекуррентных нейронных сетей. Кроме того, некоторые неоднозначные символы легче различить при наблюдении их контекста, чем распознавая каждого из них в отдельности. Во-вторых, RNN может распространять обратно ошибки

дифференциала на свой вход, то есть сверточный уровень, что позволяет нам совместно обучать рекуррентные уровни и сверточные уровни в единой сети. В-третьих, RNN может работать с последовательностями произвольной длины, проходящими от начала до конца.

У обычных RNN сетей имеется проблема с затухающими градиентами, поэтому такие сети не могут обрабатывать большие последовательности данных. LSTM блоки (показанный на рисунке 3.3.2) состоит из ячейки памяти и трех мультипликативных вентилей.

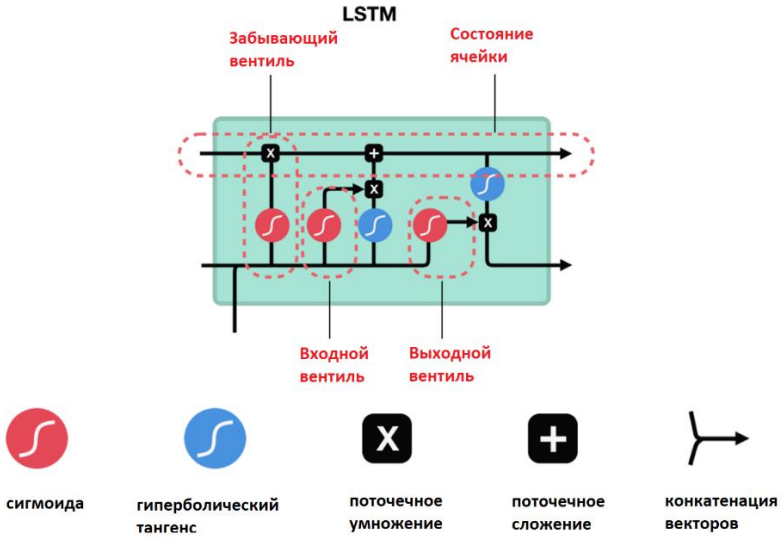


Рис.3.3.2. Архитектура ячейки LSTM

В нашем случаем мы будем использовать двунаправленные LSTM блоки, где блок состоит из двух блоков направленный против друг друга.

3.3.3 Транскрипция

Транскрипция — это метод превращение последовательности полученного из RNN в осмысленную фразу .

Коннекционная временная классификация [29].Ссуть данного метода заключается нахождения всех возможных последовательностей сырых данных в осмысленную фразу, на основе некоторых правил. Функция отображения последовательности в последовательность B определена на последовательности $\pi \in L_0 T$, где T - длина. Сначала удаляя повторяющиеся метки, а затем удаляем пробелы. Например, в примере с «--hh-e-l-ll-oo--» (’-’ представляет «пустой») преобразовывается на «hello». Тогда условная вероятность определяется как сумма вероятностей всех π:

$$p(1|y) = \sum_{\pi:\mathrm{B}(\pi)=1} p(\pi|y)$$

3.4 Архитектура системы детектирования и распознавания номерных знаков

Собрав все вышеперечисленные этапы детектирования и распознавания номерных знаков, мы получаем алгоритм, изображенный на рисунке 3.4.1.

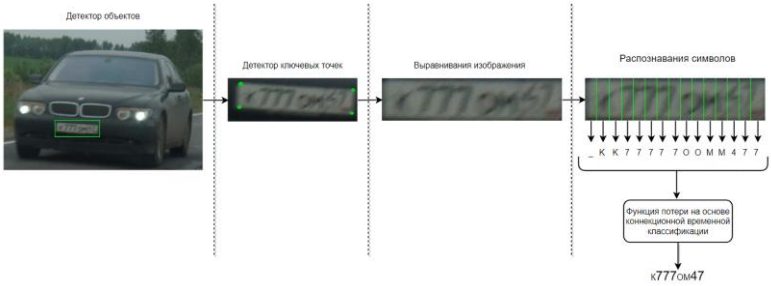


Рис.3.4.1. Архитектура системы детектирования и распознавания номерных знаков

Данная архитектура позволяет нам детектировать российские номерные знаки транспортных при разных условиях. Детектор ключевых точек скорее является опциональным дополнением системы, чтобы увеличить

точность распознавания при нахождения номерных знаков под большим углом. При нормальном использовании данный не требуется и его исключения из архитектуры можно ускорить его работу. В следующей главе будут описаны особенности реализации данной архитектуры.

Глава 4. Реализация системы

Программа была написана на языке Python 3.6, с использованием библиотеки OpenCV для работы с изображениями, NumPy для работы с большими данными и Pytorch для обучения и запуска нейронных сетей. При разработке было важно добиться идеального баланса между точностью и быстродействием.

4.1 Детектор объектов

В качестве функции потери использовалась GIoU [30], которая работает на основе коэффициента Жаккара.

$$\mathrm{GIoU} = |\mathrm{A} \cap \mathrm{B}||\mathrm{A} \cup \mathrm{B}| - |\frac{\mathrm{C} \backslash (\mathrm{A} \cup \mathrm{B})}{\mathrm{C}}|$$
$$= \mathrm{IoU} - |\frac{\mathrm{C} \setminus (\mathrm{A} \cup \mathrm{B})}{\mathrm{C}}|$$

А в качестве метода оптимизации использовалась функция Adam [31]. Это алгоритм оптимизации, который можно использоваться вместо классического стохастического градиентного спуска для итеративного обновления весов сети на основе обучающих данных. В отличие от стохастического градиентного спуска имеет моментум, которая лучше справляется с преодолением локальных минимумов. Для этого использовалась встроенная функция в Pytorch torch.optim.Adam() с адаптивной скоростью обучения, которая менялась каждые 30 эпох.

В отличие от оригинальной реализации YoloV3, вместо использования одного разрешения, например 416x416, было решено использовать 3 разных разрешения при обучении, а именно 320, 416 и 608. Обучения производилось сначала на разрешении 320, а потом на 608 и в конце при разрешении 416. А также было решено отказаться от соотношения сторон 1:1, в сторону 16:9. Это связано с тем, что при конвертировании большинство изображении, которые по умолчанию работают при

соотношении сторон 16:9 в 416x416, изображения искажается и часть информации теряется. А так как наша задачи напрямую связана с детектированием вытянутого прямоугольного объекта, то в некоторых случаях это может сильно повлиять на результаты работы алгоритма. Наша выборка состоит в большинстве случаях из фотографии автомобилей из близкого расстояния, что усложняет детектирования. Так же всегда есть риск переобучения нейронной сети, чтобы решить все эти проблемы было решено использовать аугментацию обучающей выборки данных. В нашем случае мы использовали следующие функции:

- поворот от 10 до 180 градусов
- масштабирование от -1.5x до 2.0x
- случайное вырезание кусочек изображения
- накладывание разноцветных прямоугольников на некоторые части номерных знаков.
- накладывание шумов на изображение

Для увеличения скорости обучения изображения отправлялись в пакетах из 64 изображений. Такое количество изображении занимал всю память Nvidia Tesla V100 16 ГБ. Использования пакетов является одним из сильных инструментов при обучении нейронных сетей, так как процесс переноса данных из оперативной памяти в память графического процессора занимает много времени из-за архитектурных особенностей обычных компьютеров. Поэтому нужно, как можно чаще инициализировать процесс обмена данными. Для увеличения размета было решено использовать библиотеку Apex от Nvidia, которая позволяет использовать числа половинной точности(16 бит) для хранения и обработки весов, а также дает доступ к тензорным ядрам графического процессора. При этом качество детектирования не страдает.

Во время обучения выборка данных была разделена на 2

части: тренировочная (75 000) и тестовая (20 000). Для тестирования использовались следующие 3 метрики: точность, полнота и F-мера.

Точность — это отношение правильно предсказанных положительных наблюдений к общему количеству предсказанных положительных наблюдений.

Полнота — это отношение правильно спрогнозированных положительных наблюдений ко всем наблюдениям в реальном классе.

F-мера - это средневзвешенное значение точности и отзыва. Следовательно, этот показатель учитывает как ложные срабатывания, так и ложные отрицания.

$$F_1 = 2 * \frac{\text{полнота} * \text{точность}}{\text{полнота} + \text{точность}}$$

Вся модель вместе с весами будет храниться в файле формата pt.

4.2. Детектор ключевых точек

В отличие от детектора объектов функция потери была заменена на среднеквадратическая ошибка. $MSE = \frac{1}{n}\sum_{i=1}^{n}(X - Y)^2$

Это было связано с тем, что теперь мы детектируем 4 пары точек. После детектирования изображения номерных знаков расширялись на 15% по высоте и 10% по ширине, так как после детектирования некоторые углы номерных знаков могли бы не попасть в наш входной кадр. На вход подавались изображения с шириной 128 и высотой 72. Применялись все те же аугментации, что и в детекторе объектов. Набор данных состоял из 5000 изображении отмеченных вручную. В это раз в качестве метрики использовалась только точность, так как этого достаточно для этой задачи.

4.3. Распознавание символов

На вход изображения подавались в разрешении 96x32.

Обучающая выборка была вырезана с набора данных для детектирования, а так изначально в названии файлов содержал символы номерных знаков, что в свою очередь упростило нашу задачу. В качестве функции потери было использовано CTC Loss, подробно о нем описано в подглаве 3.3.3. Из-за архитектурных особенностей, а именно из-за того, что изображения разделяется на N=20 частей, то невозможно применить некоторые аугментации(поворот, масштабирования и т.д.), потому что изображения не имеют точной разметки.

4.4 Компоновка нейронных сетей в единую систему

Была реализована система обработки изображений и видео файлов в заданной директории и обработка видеопотоков с любых камер в реальном времени. А также из-за обучения нейронной сети для детектирования есть возможность подавать картинки трёх разных разрешений. Для запуска нашей системы требуется мощная видеокарта, поэтому была реализована система серверной обработки видеопотоков, которая будет описана далее.

4.5 Серверная обработка видеопотоков

Данная часть программы будет реализована на сокетах [32]. Архитектура системы представлена на рисунке 4.5.1.

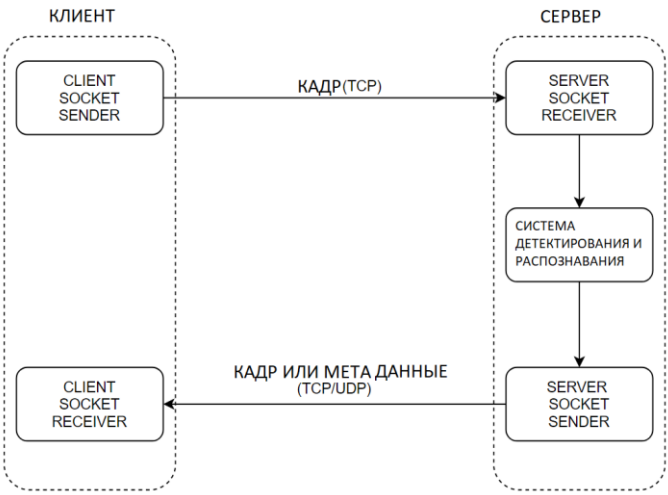


Рис.4.5.1. Архитектура системы при облачной обработке

Требуется выделить системе свободный порт для работы по протоколу TCP/UDP. На первом этапе система обрабатывает и передает изображения на сервер, где каждый кадр прогоняется через систему детектирования и распознавания номерных знаков. После этого этапа мы получаем метаданные (координаты номеров, символы номеров, точность распознавания и т.д.), затем мы можем изобразить эти данные на кадре и отправить кадр обратно на клиент или же отправить только метаданные. Все изображения отправляются в формате JPG для экономии трафика. Системные требования для клиента зависят от качества видео, которое мы передаем. Больше всего времени на клиенте занимает процесс сжатие изображения в формат JPG для дальнейшей отправки, благодаря этом сжатию мы можем очень сильно уменьшить сетевой трафик. Для серверной части требования более жесткие, обязательно требуется графический процессор с архитектурой CUDA для работы нейронных сетей, а также

соответствующий процессор для обеспечения достаточной пропускной способности для видеокарты и предварительной обработки поступающих кадров.

Глава 5. Тестирование и анализ результатов

5.1 Детектор объектов

Полное обучение детектора объекта занимает около 12 часов на видеокарте Nvidia Tesla V100 16 ГБ. После 75 эпох на тестовой части нашей выборки данных, который состоит из 20 000 изображений были получены следующие результаты:

- Полнота (Recall) – 99.7 %
- Точность (Precision) – 97.8 %
- F-мера – 98.7 %

Данные результаты были получены при входном разрешении 416x234. Исходя из результатов можно сказать, наша цель была достигнута. Судя по точности модели можно сказать, что в выборке данных имеется небольшое количество неправильно размеченных областей из-за автоматической разметки. Эту проблему можно решить с помощью простого бинарного классификатора, который сможет отличить номерные знаки от других объектов.

5.2 Детектор ключевых точек

Обучения детектора ключевых из-за использования достаточно простого архитектуры заняло около 1 часа. После обучения была достигнута точность 91 %. Это связано с тем, что выборка данных состоял из 4000 изображений для обучения и 1000 для тестов.

5.3 Распознавание символов

Полное обучение заняло около 2 часов или 80 эпох. После обучения была достигнута точность 95%. Как и ранее было упомянуто, выборка данных не была очищена от ложных данных, поэтому это повлияло на конечные результаты обучения. А также из-за того, что символы автоматическим образом распознаются, то те номера, которые находятся под

большим углом наклона, так же влияют отрицательно на конечные результаты.

5.4 Результаты работы системы

Благодаря использованию разного рода аргументационных функции, была достигнута возможность распознать номерные знаки с высоты камер видеонаблюдение, как показана на рисунке 5.4.1.

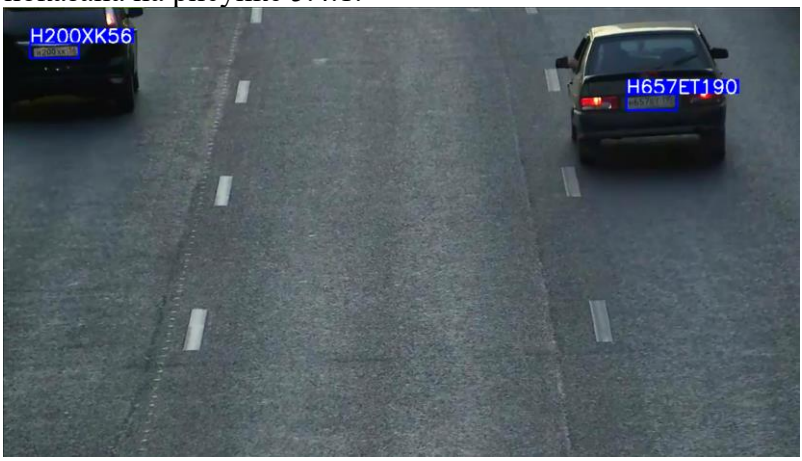


Рис. 5.4.1. Кадры с камеры видеонаблюдения трёх полосной дороги

А из-за многоуровневой архитектуры системы появилась возможность распознать номерные знаки при достаточно сложных ситуациях, как показана на рисунке 5.4.2.

Рис. 5.4.2. Результаты работы системы

Были достигнуты следующие результаты по производительности:

- 25 к/с (608x342), Nvidia GTX 1070ti, i7 6700k, FP32, batch_size = 1
- 40 к/с (608x342) AWS P3.2xlarge, 8 ядер, Nvidia Tesla V100, FP16, batch_size = 1

ЗАКЛЮЧЕНИЕ

В данной работе был разработана система детектирования и распознавания номерных знаков по изображениям.

Во время разработки системы было проанализировано различные методы для детектирования и распознавания номерных знаков, а также различные готовые решения от других разработчиков.

Для обучения системы было собрана больше 100 тысяч изображений с помощью собственного веб-парсера. А набор данных был размечен с помощью метода псевдо-разметки.

Разработанный алгоритм состоит из трёх частей. Сначала происходит детектирования расположения номерного знака с помощью нейронной сети YoloV3, а затем вырезается изображения номерного знака и передается на детектор ключевых точек. Затем мы выравниваем изображения по этим четырем точкам и подаем кадр на нейронную сеть по распознаванию символов(CRNN).

Система была разработана на языке Python 3.6. Для обучения и запуска нейронных сетей использовалась библиотека PyTorch, а для работы с изображения библиотека OpenCV. При подготовке данных для работы с большими данных было использована библиотека NumPy.

Программа была протестирована на разных видео кадрах и отдельных изображениях. В обычных сценариях эксплуатации, система справилась со всеми примерами.

Чтобы улучшить систему можно предпринять следующие шаги:

- Добавить в выборку данных номерных знак других стран.
- Перевести модели на более низкоуровневые библиотеки для нейронных. сетей
- Переписать систему на C++.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. R. Laroca, E. Severo, L.A. Zanlorensi, L.S. Oliveira, G.R. Goncalves A Robust Real-Time Automatic License Plate Recognition Based on the YOLO Detector. International Joint Conference on Neural Networks (IJCNN), 2018. P. 1-10.
2. S.M. Silva, C.R. Jung License Plate Detection and Recognition in Unconstrained Scenarios. Computer Vision – ECCV 2018. Ch.36. DOI: 10.1007/978-3-030-01258-8_36
3. Дейлид И.А., Молодяков С.А Применение методов машинного обучения для определения препятствий с помощью стереозрения // Железнодорожный транспорт. 2019. № 12. С. 27-29.
4. Nobuyuki Otsu (1979). A threshold selection method from gray-level histograms".IEEE Trans. Sys. Man. Cyber. 9 (1):62–66. doi:10.1109/TSMC.1979.4310076
5. Lienhart, R. and Maydt, J., An extended set of Haar-like features for rapid object detection, ICIP02, pp. I: 900—903, 2002
6. R. Polikar, Ensemble Based Systems in Decision Making, IEEE Circuits and Systems Magazine, vol.6, no.3, pp. 21-45, 2006
7. Nello Cristianini, John Shawe-Taylor. An Introduction to Support Vector Machines and Other Kernel-based Learning Methods. — Cambridge University Press, 2000. — ISBN 978-1-139-64363-4.
8. Altman, Naomi S. (1992). An introduction to kernel and nearest-neighbor nonparametric regression The American Statistician. 46 (3): C175–185. doi:10.1080/00031305.1992.10475879. hdl:1813/31637.
9. Rosenblatt, F. (1958). "The Perceptron: A Probabilistic Model For Information Storage And Organization In The Brain". Psychological Review. 65 (6): 386–408.

10. Ciresan, Dan; Meier, U.; Schmidhuber, J. Multi-column deep neural networks for image classification. 2012 IEEE Conference on Computer Vision and Pattern Recognition : journal. — 2012. — June. — P. 3642—3649. — doi:10.1109/cvpr.2012.6248110.
11. Taddy, Matt (2019). "Stochastic Gradient Descent". Business Data Science: Combining Machine Learning and Economics to Optimize, Automate, and Accelerate Business Decisions. New York: McGraw-Hill. pp. 303–307. ISBN 978-1-260-45277-8.
12. Haykin, Simon S. (1999). Neural Networks: A Comprehensive Foundation. Prentice Hall. ISBN 978-0-13-273350-2.
13. OpenALPR. Электронный ресурс. /. Последнее обращение https://www.openalpr.com/ 15.05.2020
14. Tesseract OCR. Электронный ресурс. /. Последнее обращение https://github.com/tesseract-ocr/ 15.05.2020
15. Sepp Hochreiter; Jürgen Schmidhuber. Long short-term memory . Neural Computation: journal. — 1997. — Vol. 9, no. 8. — P. 1735—1780. — doi:10.1162/neco.1997.9.8.1735. — PMID 9377276
16. Graves, A.; Liwicki, M.; Fernandez, S.; Bertolami, R.; Bunke, H.; Schmidhuber, J. A Novel Connectionist System for Improved Unconstrained Handwriting Recognition // IEEE Transactions on Pattern Analysis and Machine Intelligence: journal. — 2009. — Vol. 31, no. 5.
17. Plate Recognizer. Электронный ресурс. /. Последнее обращение https://platerecognizer.com/ 10.05.2020
18. Avto-nomer. Электронный ресурс. /. Последнее обращение https://avto-nomer.ru / 10.05.2020
19. Dong-Hyun Lee. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural Dong-Hyun Lee. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In

Workshop on Challenges in Representation Learning, ICML 2013, 2013.
20. S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In NIPS, 2015.
21. K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In CVPR, 2016.
22. Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. CoRR, abs/1804.02767, 2018.
23. W. Liu, D. Anguelov, D. Erhan, C. Szegedy, and S. E. Reed.SSD: single shot multibox detector. CoRR, abs/1512.02325, 2015.
24. K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. ICLR, 2015.
25. J. Hosang, R. Benenson, and B. Schiele. Learning non-maximum suppression. In ICCV, 2017.
26. Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. Generalized intersection over union: A metric and a loss for bounding box regression. In IEEE Conference on Computer Vision and Pattern Recognition, 2019.
27. LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P.Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):P 2278–2324, November 1998.
28. B. Shi, X. Bai, and C. Yao. An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. IEEE Trans. Pattern Analysis and Machine Intelligence, 2016.
29. A. Graves, S. Fernandez, F. Gomez, and J. Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In ICML, 2006.
30. Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In ICLR, 2015.

31. Xue M and Zhu C The Socket Programming and Software Design for Communication Based on Client/Server Pacific-Asia Conference on Circuits, Communications and Systems. 2009.