Санкт-Петербургский государственный университет
**Кафедра технологии программирования**

**Шилкин Евгений Владимирович**

**Выпускная квалификационная работа бакалавра**

# Анализ технологий распознавания автомобильных номеров

Направление 010400
Прикладная математика и информатика

Научный руководитель,
ст. преподаватель
Стученков А.Б.

Санкт-Петербург
2017

# Содержание

# Введение

Массовая интеграция информационных технологий во все аспекты современной жизни вызвала спрос на обработку транспортных средств в качестве ресурсов в информационных системах. Поскольку автономная информационная система без каких-либо данных не имеет смысла, возникла также потребность в преобразовании информации о транспортных средствах между реальным миром и информационной системой. Это может быть достигнуто с помощью человека, или специального программного обеспечения, которое способно распознавать автомобили по их номерным знакам. Для решения этой задачи были разработаны различные способы распознавания и системы распознавания номеров. Сегодня они используются в различных системах для контроля трафика и безопасности движения, например на парковочных стоянках, контроле границы, системах отслеживания похищенных автомобилей.

На парковке номерные знаки используются для вычисления времени, проведенного на стоянке. Когда машина въезжает на парковку, систем распознает ее номер и вносит в базу данных, а когда выезжает, разница во времени используется для вычисления платы. Во многих компаниях подобные системы используются для предоставления доступа только своему персоналу. В некоторых странах, программа автоматического распознавания номеров, установленная на границе, отслеживает и контролирует все ее пересечения автотранспортными средствами.

# Постановка задачи

В большинстве случаев, транспортное средство определяется по его номерному знаку, который легко может прочитать человек, но не машина. Для программы, номерной знак это всего лишь изображение в оттенках серого, определенное как $f(x, y)$, где $x$ и $y$ - это дискретные координаты, а $f$ это функция яркости изображения в точке. Поэтому необходимо построить аппарат, который будет извлекать характерные признаки из дискретного изображения. Такие системы являются используют методы машинного обучения, распознавания образов и обработки изображения. Таким образом, целью моей работы является изучение этих методов и их сравнение между собой.

Для изучения алгоритмов, были поставленны следующие задачи:

1. Подготовка тестового датасета
2. Алгоритмы поиска региона с номерным знаком
   - 2.1 Метод прямоугольников
   - 2.2 Выделение высокочастотных областей
   - 2.3 Сверточные нейронные сети
3. Сегментация номерного знака
4. Распознование символов
   - 4.1 Метод K-ближайших соседей
   - 4.2 Распознование символов с помощью нейронных сетей
5. Сравнение алгоритмов

# Обзор литературы

Алгоритмы распознавания номерных знаков довольно подробно описывались в литературе. К примеру, довольно полный обзор на эвристические методы был представлен в [7]. В этой статьи описывается применение алгоритмов, основанных на поиске ключевых точках изображения.

Так как основной целью своей работы я ставил исследование использования нейронных сетей для решения этой задачи, ключевыми источниками информации стали [9] и [2]. В этих книгах подробно рассказано про историю нейронных сетей, их ключевые особенности, алгоритмы обучения и типовые архитектуры.

В процессе работы я так же столкнулся с проблемой сегментации изображения, поэтому полезная оказалась статья [6], в которой описываются основные методы сегментации изображения и выделения объектов.

# Глава 1. Подготовка данных

Эффективность работы системы распознавания номерных знаков зависит не только от программных компонентов, но и от средств, с помощью которых выполняется съемка. Специализированные камеры способны делать снимки в высоком разрешении, и с помощью инфракрасных датчиков просвечивать слой грязи, что сильного облегчает распознавание номера. В случае если система установлена на высокоскоростной магистрали, к камере предъявляются дополнительные требования, такие как высокая скорость захвата, иначе изображение будет расплывчатым.

Для объективной оценки работоспособности алгоритма, тестовое множество должно содержать элементы, затрагивающие как можно большее число возможных критических моментов. Для задачи распознавания номерных знаков такими являются:

- Погнутые номерные таблички
- Неравномерная освещенность
- Дефекты символов
- Разнообразные шумы на изображении

Для составления тестового датасета использовались материалы с сайта http://avto-nomer.ru/ru/gallery. Извлечение данных производилось с помощью специального кроулера, который обходит страницы сайта и асинхронно загружает изображения.

Пример полученных изображений:


TOYOTA
С100КВ199
2016-11-01


У008ХМ76


95
95
92
ДТ
V-Power

# Глава 2. Эвристические алгоритмы

Первым шагом в процессе автоматического распознавания номеров является выделение области с номерным знаком. Его задача состоит в том чтобы алгоритм мог найти прямоугольную зону с номерном на исходном изображении. Для человека номерная табличка это «небольшая табличка прикрепленная к транспортному средству», но компьютер не понимает такое определение, так как он не знает что такое «транспортное средство», ни что либо еще. Поэтому нам нужно найти подходящее определение, описывающее номерной знак, так, чтобы оно было понятно программе.

## 2.1. Выделение границ

Для того чтобы выделить различные границы на изображении мы можем использовать операцию свертки функции $f$ с различными матрицами $\mathbf{m}$:

$$f'(x,y) = f(x,y)\tilde{*}\mathbf{m}[x,y] = \sum_{i=0}^{w-1}\sum_{j=0}^{h-1} f(x, y * \mathbf{m}[mod_w(x-i), mod_h(y-j],$$

где $w$ и $h$ - количество столбцов и строк матрицы $\mathbf{m}$ соответственно.

**Детектор вертикальных и горизонтальных границ** Для того, чтобы выделить вертикальные и горизонтальные границы можно использовать сверточные матрицы $\mathbf{m}_{he}$ и $\mathbf{m}_{ve}$. Обычно используются матрицы свертки намного меньшие, чем исходное изображение, например размера 3x3. Однако, для нахождения более грубых границы можно использовать матрицы и большей размерности

$$\mathbf{m}_{he} = \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}, \mathbf{m}_{ve} = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix}$$

**Детектор границ Собеля** Детектор границ Собеля использует пару матриц размерностью 3x3. Первая из них используется для выделения вертикальных границ, а вторая для горизонтальных.

$$\mathbf{G}_x = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}, \mathbf{G}_y = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

Итоговое значение пикселя считается по формуле $|\mathbf{G}| = \sqrt{\mathbf{G}_x^2 + \mathbf{G}_y^2}$. Однако на практике, намного быстрее будет использовать приближенное значение как: $|\mathbf{G}| = |\mathbf{G}_x| + |\mathbf{G}_y|$

## 2.2. Метод прямоугольников

Теперь, когда мы можем выделять контуры объектов на изображении, мы можем приступить к разбору алгоритмов поиска области с номерным знаком. Как известно, номерные таблички стандартизированны, а именно они имеют конкретные размеры, прописанные в ГОСТе.

$W_{plate} = 520$.

$H_{plate} = 112$.

Таким образом, мы можем перебрать все контуры, чьи ограничивающие прямоугольники попадают под ограничение:

$$4.64 - \delta \leq \mathbf{W}_{contour}/\mathbf{H}_{contour} \leq 4.64 + \delta,$$

где $\delta$ - варируемый параметр, который влияет на точность поиска. Чем он меньше, тем меньше ложных срабатываний, но тогда высока вероятность того, что мы пропустим искомый контур. Данный метод хорошо работает на изображениях, сделанных инфракрасной камерой, которая выделяет светлые участки, и приглушает темные. Однако для приведен-

ного алгоритма очень высок порог ложных срабатываний, из-за чего на изображениях с большим числом объектов он работает крайне медленно.

# Глава 3. Метод проекций

**Горизонтальные и вертикальные ранговые фильтры** Горизонтальные и вертикальные ранговые фильтры часто используются для поиска областей, с ярко выраженными границами, например область номерного знака. Ширина горизонтального рангового фильтра намного больше чем его высота ($w \gg h$), и наоборот, для вертикального рангового фильтра высота намного больше чем ширина ($h \gg w$).

Для сохранения средней яркости изображения, необходимо чтобы каждый пиксель исходного изображения был заменен средним значением пикселей, покрываемых фильтром. В общем случае это означает, что матрицу фильтра накладывается следующе условие:

$$\sum_{i=0}^{w-1} \sum_{j=0}^{h-1} \mathbf{m}_{hr}[i,j] = 1$$

## 3.1. Горизонтальные и вертикальные проекции

После применения операций свертки, мы можем попробовать найти область номерного знака, используя статистическую информацию, полученную из снимка. Существует множество различных способов статистического анализа. Одним из них является поиск вертикальной и горизонтальной проекции изображения на оси $x$ и $y$.

Вертикальная проекция изображения это график, представляющий суммарное значение пикселей вдоль оси $y$. Если мы посчитаем вертикальную проекцию после применения детектора горизонтальных границ, то каждое значение на графике будет представлять собой частоту встречаемости границы на рисунке. Таким образом, вертикальная проекция может быть использована для локализации области номерного знака. Аналогично, горизонтальная проекция, представляет собой сумму пикселей вдоль оси $x$

Пусть наше входное изображение определено как дискретная функция $f(x, y)$. Тогда, вертикальная проекция $p_y$ функции $f$ в точке $y$ это сумма всех пикселей в $y$ ряду входного изображения. Соответственно, горизонтальная проекция в точке $x$ этой функции это сумма значений пикселей в столбце $x$.

$$p_x(x) = \sum_{j=0}^{h-1} f(x,j);\ p_y(y) = \sum_{i=0}^{w-1} f(i,y)$$

где $w$ и $h$ - ширина и высота изображения.

## 3.2. Двухфазовый статистический анализ

Статистический анализ изображения состоит из двух фаз: в первую фазу мы ищем более широкую область положения номерного знака, которая сужается во второй фазе. В результате мы довольно точно получаем регион с номером. Эти две фазы основываются на одинаковых принципах, но различаются параметрами, которые используются для поиска границ области.

Первая фаза - поиск «полосы» на изображении, операция, которую мы используем чтобы найти и обрезать вертикальную границу номера с помощью вертикальной проекции изображения. Эта полоса используется для выделения номерной таблички путем исследования ее горизонтальной проекции, а не целого изображения.

## 3.3. Вертикальный поиск

Выделение полосы, содержащей номер, происходит в соответствии с анализом вертикальной проекции изображения. Если $h$ - высота исследуемого изображения, соответствующая вертикальная проекция $p_h^r(y)$ содержит $h$ значений, при $y \in \langle 0; h-1 \rangle$

График проекции обычно бывает слишком «острый» из-за высокой дисперсии значений $p_h^r(y)$. Существует два пути решения этой проблемы. Во-первых, мы можем применить сглаживающий фильтр на исходном изображении, но это довольно дорогая по времени операция. Во-вторых, мы можем уменьшить статистическую дисперсию $p_h^r(y)$, путем свертки ее с ранговым вектором:

$$p_y(y) = p_h^r(y) \tilde{*} \mathbf{m}_{hr}[y],$$

где $\mathbf{m}_{hr}$ - это ранговый вектор, по аналогии с ранговой матрицей из пункта 2.1. Обычно ширину $\mathbf{m}_{hr}$ составляет 9 элементов.

Главной проблемой анализа является поиск пиковых значений на графике вертикальной проекции. Эти значения соответствуют возможным кандидатам на нахождение номерного знака. Максимальное значение $p_y(y)$ можно найти как:

$$y_{bm} = \arg\max_{0 \le y \le h-1} \{p_y(y)\}$$

Соответсвующие границы полосы $y_{b0}$ и $y_{b1}$ можно получить как:

$$y_{b0} = \max_{0 \le y \le y_{bm}} \{y | p_y(y) \le c_y * p_y(y_{bm})\}$$
$$y_{b1} = \min_{y_{bm} \le y \le h-1} \{y | p_y(y) \le c_y * p_y(y_{bm})\},$$

где $c_y$ - параметр, с помощью которого определяется ширина полосы.

Этот метод применяется итеративно, что найти несколько возможных областей. Координаты $y_{b0}$ и $y_{b1}$ вычисляются на каждом шаге процесса, после этого значения $p_y(y)$ в области $\langle y_{b0}; y_{b1} \rangle$ зануляются, чтобы исключить ее из поиска на следующем шаге.

После того, как мы нашли несколько возможных полос, мы проводим аналогичную операцию для горизонтальной проекции каждой из них.

# Глава 4. Нейронные сети

Для лучшего понимания сущности искусственных нейронных сетей, в начале следует рассмотреть структуру и функциональность биологического нейрона. Человеческий мозг - это нейронная сеть, состоящая из более 10 миллиардов соединенных нейронов. Каждый нейрон - это клетка, использующая химические процессы чтобы обрабатывать и передавать информацию. У каждой клетки есть тело, размером несколько микрометров, содержащее тысячи «входов», называемых дендритами. У нее также есть одно выходное соединение, называемое аксоном, которое может быть несколько метров в длину. Поток данных в биологической нейронной сети представим как череда электрических сигналов, которые передаются по аксону.

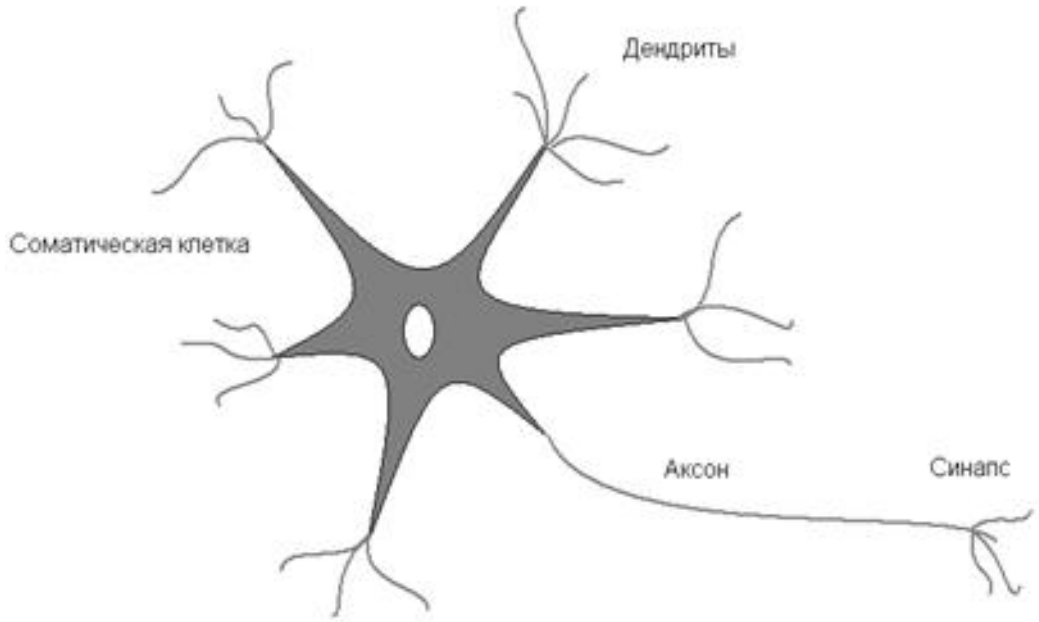


Рис. 1: Биологический нейрон

Биологическая нейронная сеть содержит два типа синаптических связей. Первый - это соединение, которое усиливает проходящий сигнал. Второй - сдерживающий, который подавляет сигнал. Поведение соединения представленое его «весом». В нейронной сети существует механизм, который позволяет регулировать веса соединений. Таким образом, система синаптических весов является представление человеческой памяти - так как веса постоянно меняются, старая информация постепенно забывается.

Так как модель биологического нейрона весьма сложна, математики разработали несколько упрощенных моделей, таких как нейрон МакКаллока-Питтса или персептрон.

## 4.1. Нейрон МакКаллока-Питтса

Нейрон МакКаллока-Питтса был первой моделью искусственного нейрона, представленной в 1943 году. У этого нейрона всего два возможных выходных значения (0 или 1) и только два типа синаптических весов: полностью усиливающий и полностью подавляющий. Усиливающий вес не влияет на входной сигнал ($w = 1$), в то время как подавляющий меняет его знак ($w = -1$).

Взвешенные входы обрабатываются нейроном следующим образом:

$$y = g(\sum_{j=0}^{J-1} w_{i,j} * x_j - \nu_i)$$

$$g(\xi) = \begin{cases} 1 & \text{if } \xi \geq 0 \\ 0 & \text{if } \xi < 0 \end{cases}$$

Этот нейрон может выполнять логические операции И, ИЛИ, НЕ. Более того МакКаллок и Питтс доказали, что совокупность таких нейронов способна реализовывать произвольные вычислительные функции, подобно машине Тьюринга. Однако поскольку биологические нейроны имеют не бинарный отклик, эта модель нейрона не подходит для аппроксимации.

## 4.2. Персептрон

Другой моделью нейрона является перспетрон. Было доказано, что сети МакКаллока-Питтса с измененными синаптическими изменениями способны обучаться для задач распознавания и классификации. Обучение заключается в изменении весов нейрона, в зависимости от реакции нейрона. Если нейрон неактивен, когда он должен быть активен - мы увеличиваем веса нейрона. Ксли он активен, когда должен быть неактивен - уменьшаем. Этот принцип был использован в первой модели классификатора на основе адаптивного линейного нейрона. Главной проблемой таких сетей было то, что они не способны решать линейно неразделимые задачи.

Эта проблема была решена математиками Румельхарт, Хилтон и Вильямс, разработавшими метод обратного распространения ошибки для обучения многослойных нейронных сетей. Простой бинарный нейрон МакКаллок Питтса был заменен нейроном с непрерывным входом/выходом.

У персептрона несколько числовых входов и один числовой выход. Пусть $x_0 \dots x_{j-1}$ - входные значения, соответствующие весам $w_{i,0} \dots w_{i,j-1}$. Взвешенные входы суммируются, и подсчитываются следующим образом:

$$y = g(\sum_{j=0}^{J-1} w_{i,j} * x_j - \nu_i)$$
$$g(\xi) = \frac{1}{1+e^{-\xi}}$$

где $g(\xi)$ - сигмоидальная функция активации (3), а $\nu_i$ - пороговое значение. Иногда пороговая величина реализовывается как еще один вход с постоянным весом -1. Также, функция нейрона может быть преобразована к виду $g(\mathbf{x} * \mathbf{w})$, где $\mathbf{x}$ - это вектор входных данных(включая пороговое значение), а $\mathbf{w}$ - вектор весов нейрона (включая постоянный вес порогового значения).

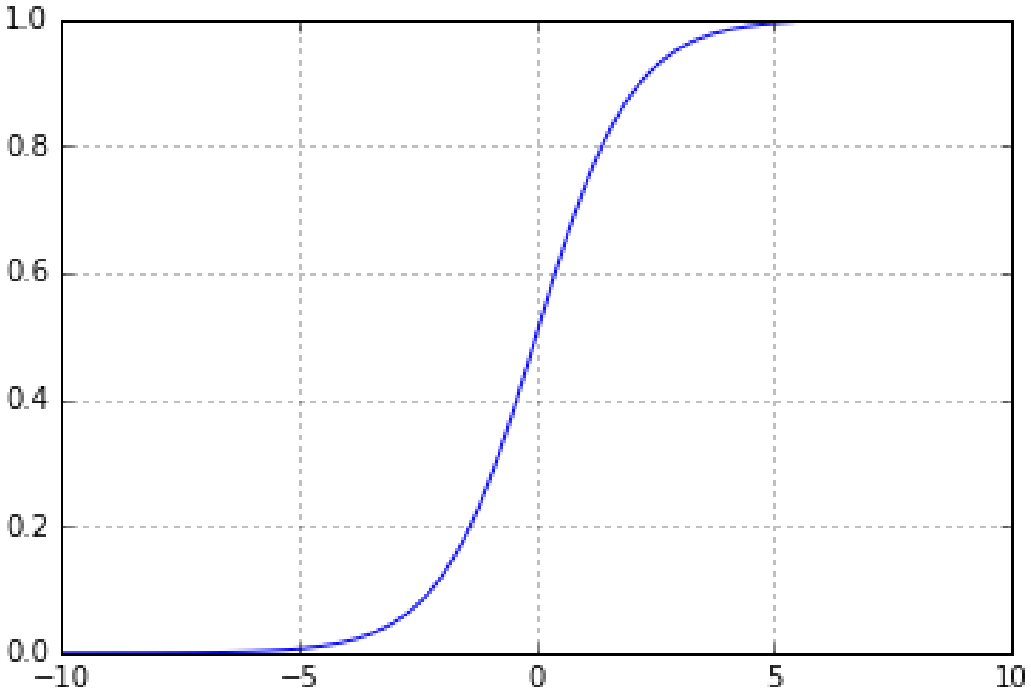


Рис. 2: Сигмоидальная функция активации

## 4.3. Сети с прямым распространением

Формально, нейронную сеть можно рассматривать как ориентированный граф $G = (N, E)$, где $N$ - не пустое множество нейронов, а $E$ - множество ориентированных связей между ними. Связь $e(n, n') \in E$ это бинарное отношение между $n$ и $n'$. Множество нейронов $N$ можно разбить на непересекающиеся подмножества $N_0, N_1, N_2$, где $N_i$ - совокупность нейронов i-го слоя.

$$N = N_0 \cup N_1 \cup N_2$$

J-ый вес i-го нейрона в слое k обозначим как $w_{i,j}^{(k)}$, а порог i-го нейрона в слое k обозначим как $\nu_i^{(k)}$. Число нейронов во входном (0), скрытом(1), и выходном слое обозначим как $m$, $n$, $o$ соответственно. Количество нейронов во входном слое соответствует длинне вектора $\mathbf{x}$, так что каждому элементу соответствует один нейрон. Нейроны входного слоя не выполняют никаких вычислительных действий, а просто передают значения нейронам скрытого слоя.

Число нейронов скрытого слоя $(n)$, может разниться, но оно влияет на способность сети к распознаванию в целом. Слишком малое число нейронов приведет к тому, что сеть не сможет запомнить новые образцы. Если же нейронов будет слишком много, нейронная сеть переобучится и потеряет способность к обобщению.

Информация в сетях с прямым распространением передается от нижних слоев к верхним в одну сторону. Связи существуют только между нейронами в соседних слоях, поэтому в таких сетях нет соединений между нейронами одного слоя или не соседними слоями.

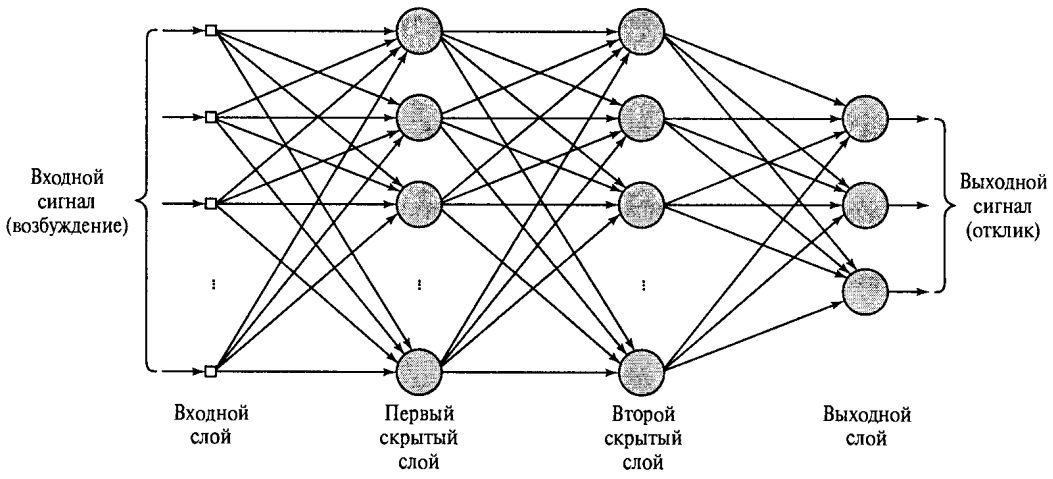


Рис. 3: Нейронная сеть с прямым распространением

## 4.4. Обучение нейронной сети с прямым распространением

Доказано, что многослойная нейронная сеть, состоящая из перспетронов с сигмоидальной функцией активации способна решать даже нелинейные задачи. С точки зрения математики это значит, что для каждой функции $F : \mathbb{R}^m \to \mathbb{R}^o$ существует многослойная нейронная сеть, которая способна аппроксимировать эту функцию. Доказательство основано на теореме Колмогорова, которая гласит, что любая непрерывно возрастающая функция, определенная на интервале $\langle 0; 1\rangle^m$ может быть выражена как:

$$f(x_0, \ldots, x_{m-1}) = \sum_{i=0}^{2-m} \alpha_i \Big( \sum_{j=0}^{m-1} \varphi_{i,j}(x_j) \Big)$$

Пусть перед нами стоит задача построить нейронную сеть, соответствующую заданной нелинейной функции. Для начала мы должны подобрать архитектуру сети. Количество нейронов во входном и выходном слое определяется размерностью входных и выходных данных, в то время как число нейронов скрытого слоя необходимо подобрать эмпирически.

Обучение нейронной сети заключается в поиске оптимальных параметров $\mathbf{w}_+$ нейронной сети. Определим две функции ошибки, для оценки качества параметра $w$:

$$E_t = \tfrac{1}{2} \sum_{i}^{A_t} (F(\mathbf{x}_i, \mathbf{w}) - \widetilde{\mathbf{x}}_\mathbf{i})^2$$
$$E_x = \tfrac{1}{2} \sum_{i}^{A_x} (F(\mathbf{x}_i, \mathbf{w}) - \widetilde{\mathbf{x}}_\mathbf{i})^2,$$

где маленькая $t$ означает тренировочное множество, а маленький $x$ - тестовое. $E_t$ - значение ошибки, определенная на значениях тренировочного множества, а $E_x$ - значение функции ошибки на тестовом множестве соответственно.

Целью процесса обучения является поиск оптимальных значений весов и порогов, чтобы минимизировать функцию ошибки $E_t$. Это итера-

тивный процесс, на каждом шаге которого сети предъявляется образец $\mathbf{x}$, и ее ответ $\mathbf{y}$ сравнивается с желаемым значением $\hat{\mathbf{x}}$. Разница между полученным и желаемым значениями используется для изменения весов. Этот процесс продолжается до тех пор, пока значение функции ошибки не становится меньше некоторого, наперед заданного значения $\epsilon$

## 4.5. Сверточные нейронные сети

Сверточные нейронные сети очень похожи на обычные нейронные сети, рассмотренные выше. Они состоят из нейронов с настраиваемыми весами и порогами. Каждый нейрон получает входные данные, скалярно умножает на вектор весов, и вычисляет значение активационной функции. Таким образом всю нейронную сеть можно представить в виде одной функции. Однако сверточные сети предполагают что входными данными являются изображения, что позволило явно задать часть параметров. Это допущение позволяет проще вычислять значение функции активации и уменьшает общее число параметров сети.

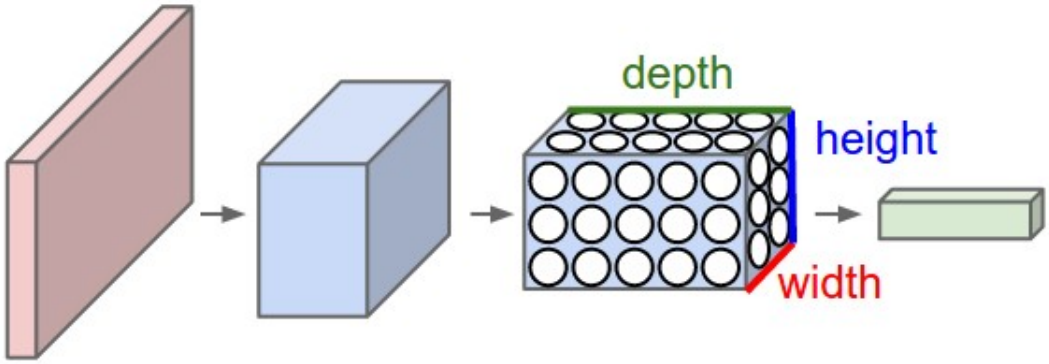


Рис. 4: Сверточная нейронная сеть

В отличии от обычной нейронной сети, сверточная сеть организует нейроны в 3 измерения: высота, ширина и глубина (4). Стоит отметить, что «глубина» относится только к третьему измерению, а не к глубине всей нейронной сети, которая характеризует общее количество слоев.

В сверточной нейронной сети обычно используется 3 типа слоев:

1. Convolution layer - слой, который будет вычислять выход нейронов, которые соединены с локальными областями входа. Каждый вычисляет произведение между весами нейрона и областью, с которой он соединен.

2. Pooling layer - слой, выполняющий операцию снижения рамерности по ширине и высоте.

3. Fully-Connected layer - полносвязный слой, как в обычной нейронной сети.

**Сверточный слой** Параметры сверточного слоя состоят из набора обучаемых фильтров. Каждый фильтр имеет небольшую размерность, например $5 \times 5$. Во время прямого прохода мы "скользим" фильтром по изображению и вычисляем произведение элементов фильтра и области, затронутой им. В результате мы получаем двухмерную карту активации, каждый элемент которой соответствует значению фильтра в определенной позиции. При обучении сеть будет подбирать фильтры, которые активируются, когда они "видят" какой-то тип визуальных объектов, будь то линия или пятно какого-то цвета на первом слое, или более сложные узора на более высоких уровнях сети. На каждом слое у нас есть целый набор фильтров, каждый из которых на выходе дает отдельную двухмерную карту. Объединив эти карты в одну матрицу, мы получим выходное значение слоя.

**Пространственное расположение** Мы объяснили организацию сверточного слоя, но не рассмотрели размерность выходного сигнала. Ей управляют три гиперпараметра: глубина, шаг, и заполнение.

1. Глубина выходного сигнала соответствует числу фильтров, каждый из которых учится искать какие-либо признаки на входе. К пример, если первый сверточный слой принимает на принимает на вход необработанное изображение, то различные нейроны могут активироваться при наличии различных участков границ.
2. Шаг фильтра. Когда шаг равен единице, мы перемещаем фильтры по одному пикселю за раз. Когда шаг два (или три и более, что встречается достаточно редко), тогда при движении фильтров, они смещаются по два пикселя за раз. Это приводит к уменьшению размерности выходного значения.
3. Иногда бывает удобно дополнить входное изображение нулями по границе. Хорошая особенность этой операции заключается в том, что она позволяет управлять размеров выходного сигнала, например мы можем использовать ее, для сохранения пространственных размеров входных данных.

Мы можем посчитать размер выходного изображения в зависимости от входного изображения($\mathbf{W}$), рамера фильтра ($\mathbf{F}$), шага, с которым он применяется ($\mathbf{S}$), и размера нулевого заполнения границы ($\mathbf{P}$). Можно доказать, что искомая формула

$$N = \frac{W-F+2P}{S} + 1$$

К примеру, для входа размером $7 \times 7$ и фильтром $3 \times 3$ с шагом 1 и заполнением 0 мы получим выход размером $5 \times 5$. Если мы возьмем шаг равный 2, то размерность будет $3 \times 3$.

**Pooling layer** Обычно pooling слой вставляют между последовательными сверточными слоями. Его функция заключается в постепенном уменьшении размерности сигнала для уменьшения числа параметров, времени вычисления, а также контроля переобучения. Pooling слой действует независимо на каждую матрицу во входном сигнале, и изменяет ее размер используя операцию MAX. Наиболее распространенными являются pooling слоя с фильтрами размером $2 \times 2$, примененные с шагом 2. Они уменьшают ширину и высоту матрицы в 2 раза, отбрасывая 75% вычислительных операций на следующем уровне. В общем случае pooling layer:

- Принимает на вход сигнал размером $W_1 \times H_1 \times D_1$
- Требует два гиперпараметра: размер фильтра **F** и шаг **S**
- На выходе получается сигнал размером $W_2 \times H_2 \times D_2$, где:
  $W_2 = \frac{W_1 - F}{S} + 1$
  $H_2 = \frac{H_1 - F}{S} + 1$
  $D_2 = D_1$
- Имеет нулевое число параметров, так как вычисляет фиксированную функцию над входными данными

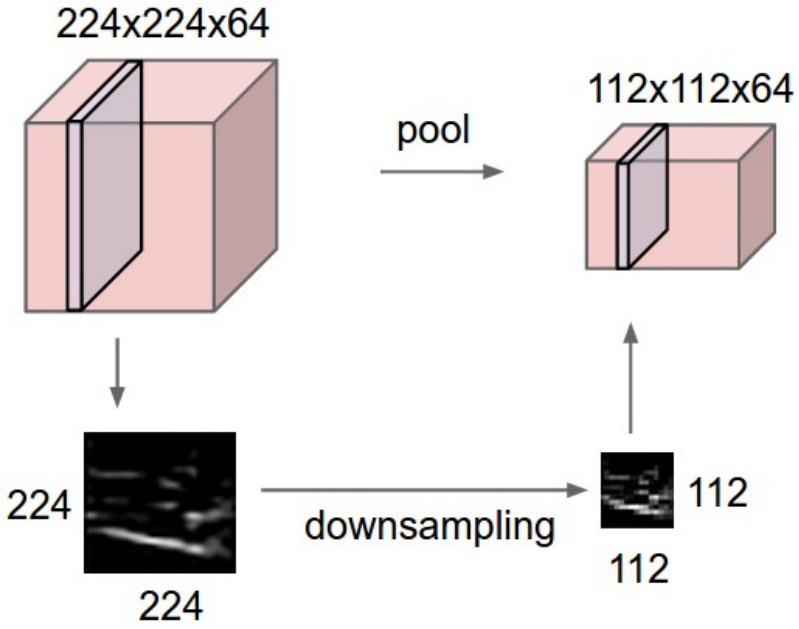


Рис. 5: Принцип работы pooling layer

# Глава 5. Детектор номерного знака на основе нейронной сети

У рассмотренных выше эвристических алгоритмов поиска области номерного знака есть один существенный недостаток: все они имеют ряд параметров, которые сильно влияют на точность поиска и требуют тонкой настройки. Более того, набор параметров, который хорошо справляется с некоторыми изображениями, может совершенно не подходить для других. Таким образом, использование нейронных сетей, которые способны самостоятельно подбирать оптимальные параметры в заданном контексте, выглядит целесообразно. Более того, используя способность нейронной сети к обобщению, мы можем рассчитывать на то, что наш алгоритм будет хорошо работать на данных, не представленных в обучающем множестве.

## 5.1. YOLO network

YOLO (You Only Look Once) - нейронная сеть, основной задачей которой является поиск и классификация объектов на изображении. Главными ее преимуществами являются:

- Простота - она имеет крайне заурядную архитектуру. В этой сети используются только стандартные сверточные, pooling и полносвязные слои.
- Скорость работы. YOLO способна работать с видеопотоком с частотой 45 кадров в секунду, что идеально подходит для realtime распознования.
- YOLO работает со целым изображением. В отличии от рекурентных нейронных сетей, которые анализируют изображение по частям, YOLO получает признаки из целостного контекста, что позволяет более точно находить искомые элементы

**Архитектура** Данная сеть содержит 24 сверточных слоя, за которыми следуют 2 полносвязных. Подробнее можно посмотреть в таблице 1

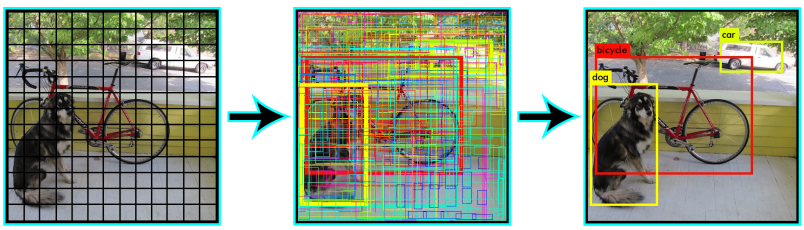


Рис. 6: Пример работы YOLO network

Таблица 1: Архитектура

| № | Layer type | Parameters |
|---|---|---|
| 1 | Conv. Layer 7x7x64 | Stride 2 |
| 2 | Maxpool Layer 2x2 | Stride 2 |
| 3 | Conv.Layer 3x3x192 | Stride 1 |
| 4 | Maxpool Layer 2x2 | Stride 2 |
| 5 | Conv. Layer 1x1x128 | |
| 6 | Conv. Layer 3x3x256 | |
| 7 | Conv. Layer 1x1x256 | |
| 8 | Conv. Layer 3x3x512 | |
| 9 | Maxpool Layer 2x2 | Stride 2 |
| 10 | Conv. Layer 1x1x256 | |
| 11 | Conv. Layer 3x3x512 | |
| | Repeat 10-11 3 times | |
| 18 | Conv. Layer 1x1x512 | |
| 19 | Conv. Layer 3x3x1024 | |
| 20 | Maxpool Layer 2x2 | Stride 2 |
| 21 | Conv. Layer 1x1x512 | |
| 22 | Conv.Layer 3x3x1024 | |
| | Repeat 21-22 1 time | |
| 25 | Conv. Layer 3x3x1024 | |
| 26 | Conv. Layer 3x3x1024 | Stride 2 |
| 27 | Conv. Layer 3x3x1024 | |
| 28 | Conv. Layer 3x3x1024 | |
| 29 | Dense layer 4096 | |
| 30 | Dense Layer 7x7x30 | |

## 5.2. Адаптация сети YOLO к задаче

Сеть YOLO преполагает решение более сложной задачи, чем поиск объекта на изображении, поэтому для увеличения скорости работы программы, в ее архитектуру были внесены некоторые изменения:

- Было уменьшено общее число сверточных слоев с 24 до 14.
- Был изменен формат выходных данных - так как нам нет необходимости классифицировать найденные объекты, мы вычисляем только вероятность нахождения объекта в регионе, и его ограничивающую область.
- В силу проведенных изменений, использовать уже обученную часть не представляется возможным, поэтому новая нейронная сеть был полностью заново обучена на собранном датасете.

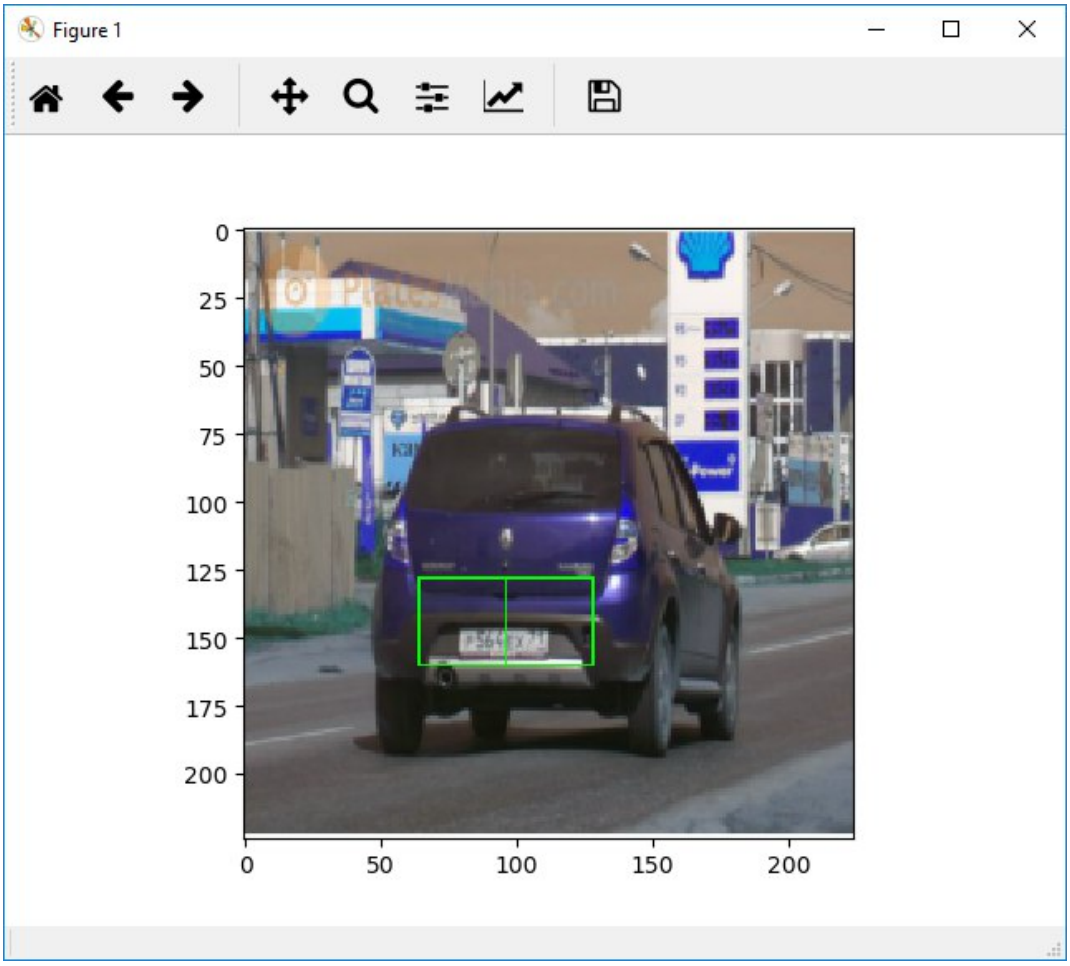


Рис. 7

Как видно из таблицы 2 полученная нейронная сеть успешно справляется с задачей выделения области номерного знака, даже на изображениях, которые не участвовали в процессе обучения.

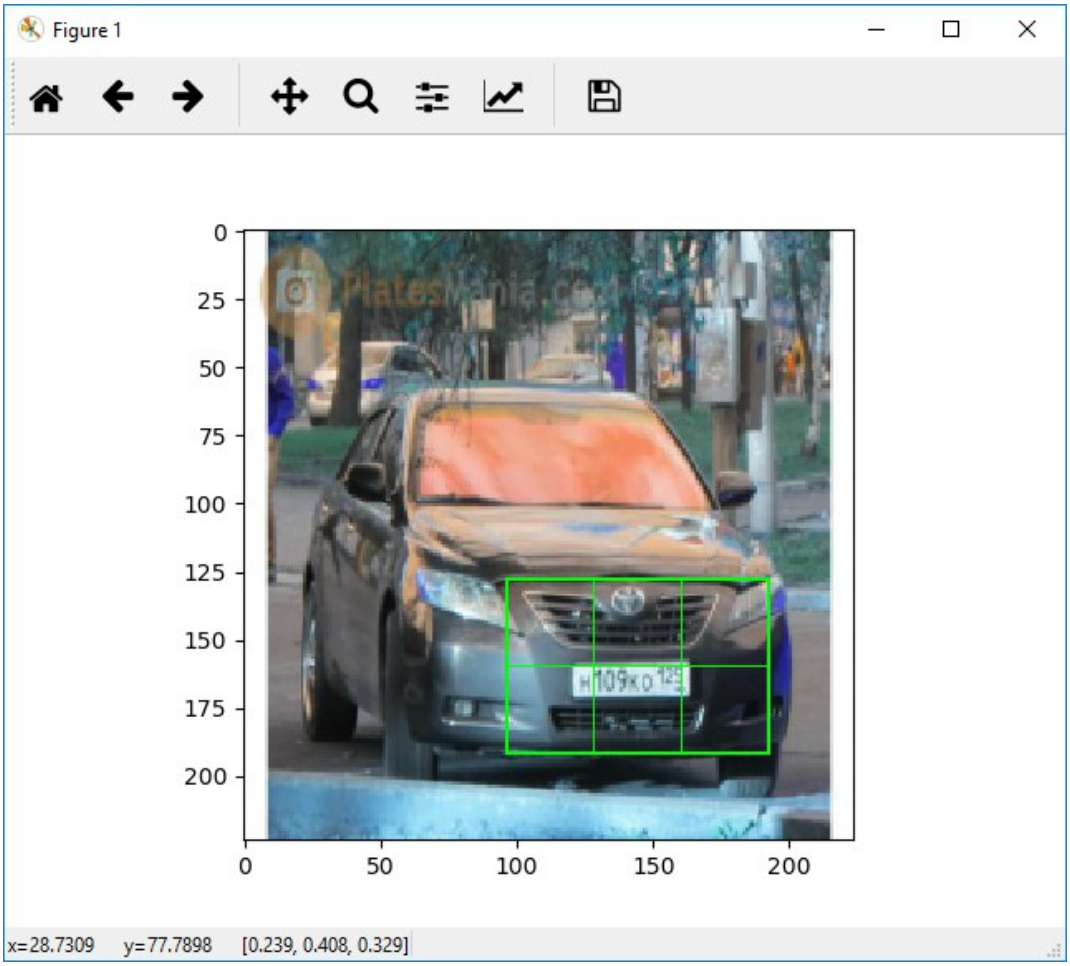


Рис. 8

Таблица 2: Результаты обучения сети

| | Image count | Succesfull results |
|---|---|---|
| Train set | 622 | 584 |
| Test set | 200 | 161 |

# Глава 6. Сегментация номерного знака

Следующим шагом после нахождения участка с номерной табличкой является ее сегментация. Сегментация - один из самых важных этапов в процессе распознавания номера, потому что все последующие действия зависят от ее результатов.

Мы можем использовать горизонтальную проекцию изображения для поиска пропусков между символами. Кроме того, кроме символов знак может содержать нежелаемые элементы, такие как черточки и точки, от которых желательно избавиться и извлечь только символы.

## 6.1. Удаление шумов

Для начала избавимся от лишних элементов на изображении. Для этого можно использовать различные медианные фильтры. Принцип работы медианного фильтра заключается в следующем: значения пикселей внутри окна фильтра сортируются в порядке возрастания, после этого выбирается значение, стоящее в середине полученной последовательности. Выбранное значение и будет выходным результатом фильтра. Медианные фильтры эффективно справляются с удалением различных шумов с изображения.

## 6.2. Морфологические операции на бинарных изображениях

**Наращивание** Операция наращивания изображения $A$ маской $B$ обозначается как $A \oplus B$ и задается выражением:

$$A \oplus B = \bigcup_{b \in B} A_b$$

Маска $B$ применяется ко всем элементам изображения $A$. При этом между элементами маски и затронутой ей областью выполняется операция И, а после, к полученным значениям применяется операция ИЛИ. Таким образом, результирующим значением фильтра будет 1, если в изображении хотя бы под одним ненулевым элементом фильтра стоит 1, и 0 в противном случае.

**Эрозия** Эрозия изображения $A$ с маской $B$ обозначается как $A \ominus B$ и определяется как:

$$A \ominus B = \{z \in A | B_z \subseteq A\}.$$

При выполнении операции эрозии мы также проходим маской по всем пикселям изображения. Если в некоторой позиции каждый единичный пиксел маски совпадет с единичным пикселем бинарного изображения, то выполняется логическое сложение центрального пиксела структурного элемента с соответствующим пикселем выходного изображения.

**Замыкание** Замыкание бинарного изображения $A$ с маской $B$ обозначается как $A \bullet B$ и определяется как:

$$A \bullet B = (A \oplus B) \ominus B.$$

При выполнении операции наращивания мы избавляемся от различных дыр и щелей в изображении, но при этом увеличиваем контуры объектов. Чтобы избежать этого, можно применить эрозию с той же маской. В результате мы получим объект с заполненными пропусками.

## 6.3. Выделение символов

После того как мы очистили изображение от шумов, его следует бинаризировать с помощью адаптивного порога. Это необходимо для того чтобы верно разделить светлый и темный фон в случае неравномерного освещения. Теперь, когда у нас есть бинаризированное изображение, с помощью операции замыкания мы можем выделить контуры букв.

Вычислим горизонтальную проекцию $p_x(x)$ замкнутого изображения $f(x, y)$. Очевидно, что пики на графике проекции будут соответствовать пропускам между символами. С помощью алгоритма, описанного в пункте 3.3, мы можем выделить контуры символов.

# Глава 7. Распознавание символов

Теперь, когда у нас есть выделенные символы, мы можем приступить к их распознаванию. Стоит заметить, что некоторые символы достаточно трудно отличимы друг от друга, например O и 0. Однако, так как мы знаем что на изображении представлен автомобильный номер, который имеет строгую структуру, мы сможем разделить эти два символа.

## 7.1. K-nearest neighbors

Одним из алгоритмов, которым можно определить изображенный символ, является метод $K$ ближайших соседей. K-nearest neighbors (k-NN) метрический метод, используемый для задач классификации и регрессии. Тренировочные элементы представлены векторами в многомерном пространстве, каждому из них соответствует метка класса. Во время обучения алгоритма, мы просто сохраняем эти вектора и их метки.

При классификации, нашему запросу присваивается тот класс, который наиболее часто встречается среди $K$ элементов тренировочного множества, наиболее близких к нему. Для вычисления расстояния можно использовать различные метрики, например для непрерывных величин используют Евклидово расстояние, а для бинарных можно воспользоваться расстоянием Хэмминга.

Выбор параметра $K$ сильно зависит от обучающих данных. Обычно, большие значения $K$ снижают влияние шумов на результат классификации, но делают границы между классами менее различимыми. Точность $k - NN$ может значительно пострадать от наличия в обучающей выборке зашумленных или нерелевантных данных, однако, если у нас имеется достаточно большая база, охватывающая многие параметры,например в нашем случае это символы, отснятые под разными углами и в разном масштабе, метод $K$ ближайших соседей показывает хорошие результаты.

## 7.2. Распознавание символов с помощью нейронной сети

Существует множество решений для задачи оптического распознавания изображения с использованием нейронной сети. Множество символов, на автомобильных номерах имеет мощность 22 элемента - 12 букв и 10 цифр. Для упрощения задачи классификации, символы $O$ и 0, а также $B$ и 8 были объединены в один класс. Так последовательность букв и цифр на номерной табличке строго определена, мы всегда сможем определить, какой из двух этих знаков следует использовать, при этом сеть

будет лучше обучаться, так как различия между такими символами не позволяют разделить их в разные классы.

# Глава 8. Реализация

Все приведенные выше алгоритмы были реализованы мной на языке программирования Python, в виде прикладных модулей. Каждый из них, предоставляет функцию, которая на вход принимает изображения, а на выходе выдает множество пар вида $\langle BoundingRect; PlateNumber \rangle$. Для операций с изображениями, таких как применения фильтра, дилатация, расширения, использовалась библиотека компьютерного зрения opencv, как самая популярная и удобная в использовании. Для реализации и обучения нейронных сетей использовался стек tensorflow и keras. Tensorflow - фреймворк для математических вычислений на GPU, что значительно ускорило процесс обучения (около двадцатикратного прироста скорости, по сравнению с обучением на CPU). Keras - это библиотека позволяющая быстро создавать, изменять, обучать и сохранять нейронные сети. Ее преимуществом является высокоуровневый API, который позволяет нам манипулировать сразу слоями сети, а не низкоуровневыми объектами, такими как матрицы и вектора.

# Выводы

Для того, чтобы определить качество каждого алгоритма, и выяснить, какой из них лучше работает, был проведен ряд тестов(9). Для этого, из общего датасета, были выбраны 200 изображений, не участвоваших в обучении нейронных сетей.

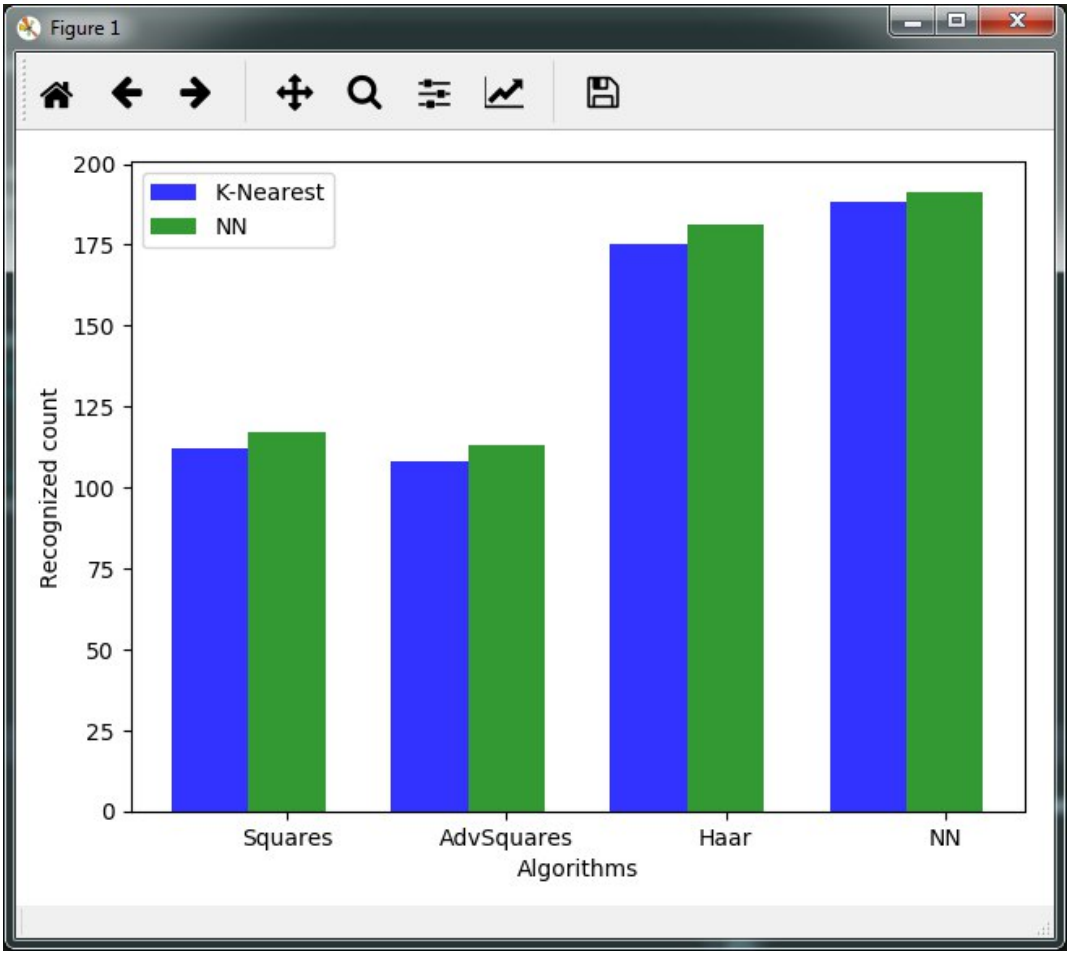


Рис. 9

В полученных результатах, стоит заметить, что нейронные сети оказались лучшим решением как для задачи поиска области номерного знака, так и для распознавания отдельных символ. Также, я предполагал что статистические методы покажут лучший результат, чем простой поиск контуров, однако это оказалось нет. Этот факт я могу объяснить тем, что на многих изображения присутствует высокочастотный фон: вывески, деревья, группы людей, которые сильно влияли на точность выделения региона.

# Заключение

В результате проделанной работы, были реализовны некоторые алгоритмы распознавания автомобильных номеров и проведено их сравнение между собой. Данное исследование показало эффективность использования нейронных сетей в задачах машинного обучения и реализации компьютерного зрения. В качестве улучшения этих алгоритмов можно рассмотреть создание end-to-end сети, которая будет самостоятельно выделять область и распознавать символы. Создание и обучение такой сети может стать объектом магистерской диссертации.

# Список литературы


[1] Kuo-Ming Hung and Ching-Tang Hsieh A Real-Time Mobile Vehicle License Plate Detection and Recognition, Tamkang Journal of Science and Engineering, Vol. 13, No. 4, pp. 433 -442 (2010)

[2] Fraser N.: Introduction to Neural Networks, http://www.virtualventures.ca/ neil/neural/neuron.h tml

[3] Convolutional Neural Networks (CNNs / ConvNets), http://cs231n.github.io/convolutional-networks/#conv

[4] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection

[5] Gonzalez R., Woods R.: Digital Image Processing, Prentice Hall, Upper Saddle River, New Jersey, 2002

[6] Zhang Y., Zhang C.: New Algorithm for Character Segmentation of License Plate, Intelligent Vehicles Symposium, IEEE, 2003

[7] Ondrej Martinsky, ALGORITHMIC AND MATHEMATICAL PRINCIPLES OF AUTOMATIC NUMBER PLATE RECOGNITION SYSTEMS, BRNO UNIVERSITY OF TECHNOLOGY, 2007

[8] Потапов А. А., Пахомов А. А., Никитин С. А., Гуляев Ю. В., Новейшие методы обработки изображений. — М.: Физматлит, 2008. — 496 с.

[9] Мак-Каллок У. С., Питтс В. Логическое исчисление идей, относящихся к нервной активности // Автоматы / Под ред. К. Э. Шеннона и Дж. Маккарти. — М.: Изд-во иностр. лит., 1956. — С. 363—384. (Перевод английской статьи 1943 г.)

[10] Graham, Benjamin (2014-12-18), "Fractional Max-Pooling", https://arxiv.org/abs/1412.6071

[11] Convolutional Neural Networks (LeNet), http://deeplearning.net/tutorial/lenet.html

[12] Viola and M. Jones. Robust real-time object detection. International Journal of Computer Vision, 4:34–47, 2001.

[13] LaTeX on Wikibooks. http://en.wikibooks.org/wiki/LaTeX

# Приложение

Исходный код кроулера:

```
import asyncio
import lxml.html
from os.path import basename, join
from aiohttp import ClientSession

api_url = 'http://avto-nomer.ru/ru/gallery-{0}'


async def fetch(url, session: ClientSession):
    async with session.get(url) as response:
        return await response.text()


async def crawl_page(page):
    with ClientSession() as session:
        page = await fetch(api_url.format(page), session)
        tree = lxml.html.fromstring(page)
        urls = tree.xpath('//img[@class=\"img-responsive cent
        tasks = (asyncio.ensure_future(fetch_image(url, sessi
        await asyncio.gather(*tasks)

async def fetch_image(image_url, session: ClientSession):
    async with session.get(image_url) as response:
        dump = await response.read()
        with open(join('data', basename(image_url)), 'wb') as
            f.write(dump)


async def crawl():
    for i in range(30):
        await crawl_page(i)

if __name__ == '__main__':
    loop = asyncio.get_event_loop()
    loop.run_until_complete(asyncio.ensure_future(crawl()))
    loop.close()
```

Подготовка нейронной сети:

```
import numpy as np
```

```
import pickle
from keras.models import Sequential
from keras.layers import MaxPooling2D, Convolution2D, UpSampl
from keras.layers import Dense, Reshape
from keras.layers.advanced_activations import LeakyReLU
from keras import backend as K
from keras.callbacks import EarlyStopping, ModelCheckpoint


def create_model():
    lrelu = LeakyReLU(alpha=.1)
    lrelu.__name__ = "LeakyReLU"

    model = Sequential()
    model.add(Convolution2D(64, (7, 7), input_shape=(224, 224
                            activation=lrelu, strides=2))
    model.add(MaxPooling2D())
    model.add(Convolution2D(192, (3, 3), activation=lrelu))
    model.add(MaxPooling2D())
    model.add(Convolution2D(128, (1, 1), activation=lrelu))
    model.add(Convolution2D(256, (3, 3), activation=lrelu))
    model.add(Convolution2D(256, (1, 1), activation=lrelu))
    model.add(Convolution2D(512, (3, 3), activation=lrelu))
    model.add(MaxPooling2D())
    model.add(Convolution2D(256, (1, 1), activation=lrelu))
    model.add(Convolution2D(512, (3, 3), activation=lrelu))
    model.add(Convolution2D(256, (1, 1), activation=lrelu))
    model.add(Convolution2D(512, (3, 3), activation=lrelu))
    model.add(Convolution2D(256, (1, 1), activation=lrelu))
    model.add(Convolution2D(512, (3, 3), activation=lrelu))
    model.add(Convolution2D(512, (1, 1), activation=lrelu))
    model.add(Convolution2D(1024, (3, 3), activation=lrelu))
    model.add(MaxPooling2D())
    model.add(Flatten())
    model.add(Dense(4096, activation='linear'))
    model.add(Dense(512, activation='linear'))
    model.add(Dense(49, activation='sigmoid'))
    model.add(Reshape((7, 7)))
    model.compile(loss='mean_squared_error', optimizer='sgd')
    return model
```

```
if __name__ == '__main__':
    model = create_model()
    with open('data2.dt', 'rb') as f:
        data, labels = pickle.load(f)

    filepath = 'weights/weights.{epoch:02d}-{loss:.2f}.h2f5'
    model.fit(data, labels, batch_size=4, epochs=20)
    model.save_weights("weights/weights_1.h2f5", True)
```