

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «КубГУ»)

Факультет компьютерных технологий и прикладной математики
Кафедра информационных технологий

КУРСОВАЯ РАБОТА

**РАСПОЗНАВАНИЕ АВТОМОБИЛЬНЫХ НОМЕРОВ ПРИ ПОМОЩИ
ИСКУССТВЕННОЙ НЕЙРОННОЙ СЕТИ**

Работу выполнил _____ С.О. Брезицкий
(подпись)

Направление подготовки 01.03.02 Прикладная математика и информатика

Направленность Программирование и информационные технологии

Научный руководитель, доц.
канд. техн. наук, доц. _____ А.А. Полупанов
(подпись)

Нормоконтролер
канд. пед. наук, доц. _____ А.В. Харченко
(подпись)

Краснодар
2022

РЕФЕРАТ

Курсовая работа 39 с., 22 рис., 12 источников.

СВЁРТОЧНЫЕ НЕЙРОННЫЕ СЕТИ, РАСПОЗНАВАНИЕ, СВЁРТКА, СИСТЕМА, ОБУЧЕНИЕ, МОДЕЛЬ

Цель работы: спроектировать систему для распознавания автомобильных номеров.

В процессе работы были изучены: основные понятия свёрточных нейронных сетей, средства разработки и обучения свёрточных нейронных сетей, архитектура и процесс разработки свёрточных нейронных сетей.

В практической части был разработана система, позволяющая распознать автомобильный номер с использованием свёрточных нейронных сетей. Представлена программа, которая реализует систему.

Средства разработки: язык программирования Python, библиотеки OpenCV, NumPy, Scikit-image, matplotlib, TensorFlow, PyTesseract.

СОДЕРЖАНИЕ

Введение.....	4
1 Методика распознавания объектов.....	5
1.1 Постановка задачи.....	6
2 Свёрточные нейронные сети (СНС).....	7
2.1 Структура свёрточных нейронных сетей.....	8
2.1.1 Свёрточный слой и операция свёртки.....	9
2.1.2 Субдискретизирующий слой.....	19
2.1.3 Полносвязный слой.....	21
3 Программная реализация.....	22
3.1 Выбор языка программирования.....	22
3.2 Используемые пакеты.....	22
3.3 Выделение номера с помощью свёрточной сети.....	24
3.3.1 Построение сети.....	24
3.3.2 Обучение сети.....	28
3.4 Распознавание номера.....	32
3.4.1 Выбор модели для выделения номера.....	32
3.4.2 Подготовка изображения.....	33
3.4.3 Распознавание символов.....	36
Заключение.....	37
Список использованных источников.....	38

ВВЕДЕНИЕ

Проблема автоматизированного оперативного распознавания текстовой информации является актуальной задачей, связанной с широким классом практических задач. Одной из таких задач является распознавание автомобильных номеров. Создание алгоритма, распознающего автомобильные номера, позволяет:

- автоматизировать контроль въезда и перемещения транспортных средств на объектах с ограниченным доступом и закрытых территориях;

- на автомагистралях обеспечить контроль транспортных потоков и осуществлять автоматическое трассирование угнанных транспортных средств и тех, за которыми числятся правонарушения;

- автоматизировать сбор статистики для муниципальных служб;

- автоматизировать контроль выезда оплаченных или неоплаченных транспортных средств на станциях технического обслуживания и автокомбинатах, контролировать загрузку зоны обслуживания;

- отслеживать въезд и выезд на автостоянках, осуществлять автоматический подсчет стоимости предоставленных услуг, контролировать свободное место;

- отслеживать въезд, выезд и время нахождения транспортных средств на территории склада и терминала, предотвращать возможные хищения.

Проблема становится актуальнее, если учесть, что на сегодняшний день в России, как и во всём мире, существует тенденция на увеличение количества автомобилей на душу населения. Следовательно, с увеличением числа автомобилей на дорогах возрастает необходимость контроля за ними.

1 Методика распознавания объектов

Процесс распознавания автомобильного номера изображен на рисунке 1.

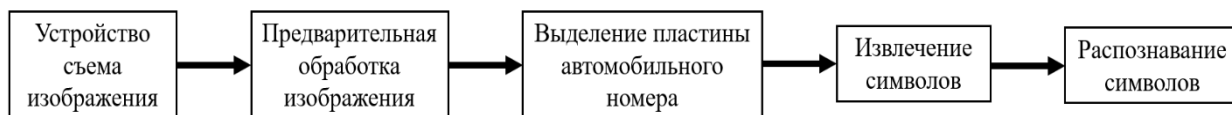


Рисунок 1 – Схема распознавания автомобильного номера

- 1) устройство съема изображения – видеокамера;
- 2) предварительная обработка полученного изображения включает следующие этапы: коррекция изображения – выравнивание, ограничение экстремальных значений яркости, устранение эффекта смазывания изображения, возникающего в связи с тем, что скорость автомобиля больше, чем скорость регистрации, устранение избыточной информации – использование бинаризации (перевод изображения из цветного в черно-белое);
- 3) выделение пластины номера выполняется при помощи сегментации;
- 4) на 4 и 5 этапах выполняется извлечение символов с изображения и их распознавание.

Одним из наиболее эффективных методов распознавания образов (номеров) является свёрточные нейронные сети CNN (Convolutional Neural Network), которые являются логическим развитием идей таких архитектур нейронной сети как когнитрона и неокогнитрона.

Использование свёрточных нейронных сетей для распознавания изображений обусловлено двумя основными факторами:

- 1) снижение сложности нейронной сети и сложности её обучения, в сравнении с классическим многослойным персептроном, что актуально в области обработки и анализа изображений;

2) повышение устойчивости распознавания к различным к искажениям символов в сравнении с классическими нейронными сетями и другими методами классификации изображений.

1.1 Постановка задачи

В данной курсовой работе исследован основной метод, который позволяет осуществить распознавание автомобильных номеров, а именно свёрточные нейронные сети. Рассмотрена структура и свойства этой сети. С помощью этого метода была спроектирована система для распознавания автомобильных номеров, а также были исследованы особенности его работы при различных входных данных.

2 Свёрточные нейронные сети (СНС)

Свёрточные нейронные сети (convolutional neural networks, CNN) — это весьма широкий класс архитектур, основная идея которых состоит в том, чтобы переиспользовать одни и те же части нейронной сети для работы с разными небольшими, локальными участками входов. Как и многие другие нейронные архитектуры, сверточные сети известны довольно давно, и в наши дни у них уже нашлось много самых разнообразных применений, но основным приложением, ради которого люди когда-то придумали сверточные сети, остается обработка изображений.

Именно зрительная кора, а точнее исследования Дэвида Хьюбела и Торстена Визеля на рубеже 50—60-х годов XX века мотивировали многие идеи, которые сегодня используются в СНС.

Идея свёрточных нейронных сетей появилась очень давно по меркам машинного обучения. Можно сказать, что первой настоящей сверточной сетью, позаимствовавшей для информатики воплощенные природой в зрительной коре идеи, был Неокогнитрон (Neocognitron) Кунихико Фукусимы, появившийся в 1979–1980 годах. Впрочем, Фукусима не использовал методы оптимизации, используемые в нейронных сетях, а именно градиентный спуск и вообще обучение с учителем, а его работы были довольно прочно забыты. Снова сверточные сети в уже вполне современной форме появились только в работах группы Яна ЛеКуна в конце 1980-х годов, и с тех пор и до наших дней они вполне успешно применяются для распознавания изображений и многих других задач [1].

Главная идея, лежащая в основе свёрточных нейронных сетей, состоит в том, что вполне достаточно локального осмысления изображения. [2].

Основными преимуществами свёрточных нейронных сетей являются:

– удобное распараллеливание вычислений, а следовательно, возможность реализации алгоритмов работы и обучения сети на графических процессорах;

- относительная устойчивость к повороту и сдвигу распознаваемого изображения;
- обучение при помощи классического метода обратного распространения ошибки;
- по сравнению с полносвязной нейронной сетью (типа персептрона) — гораздо меньшее количество настраиваемых весов, так как одно ядро весов используется целиком для всего изображения, вместо того, чтобы делать для каждого пикселя входного изображения свои персональные весовые коэффициенты. Это подталкивает нейросеть при обучении к обобщению демонстрируемой информации, а не попиксельному запоминанию каждой показанной картинке в мириадах весовых коэффициентов, как это делает персептрон [3].

2.1 Структура свёрточных нейронных сетей

СНС состоит из разных видов слоёв: свёрточные (convolutional) слои, субдискретизирующие (subsampling, подвыборка) слои и слои полносвязной нейронной сети – персептрона. Схема СНС представлена на рисунке 2.

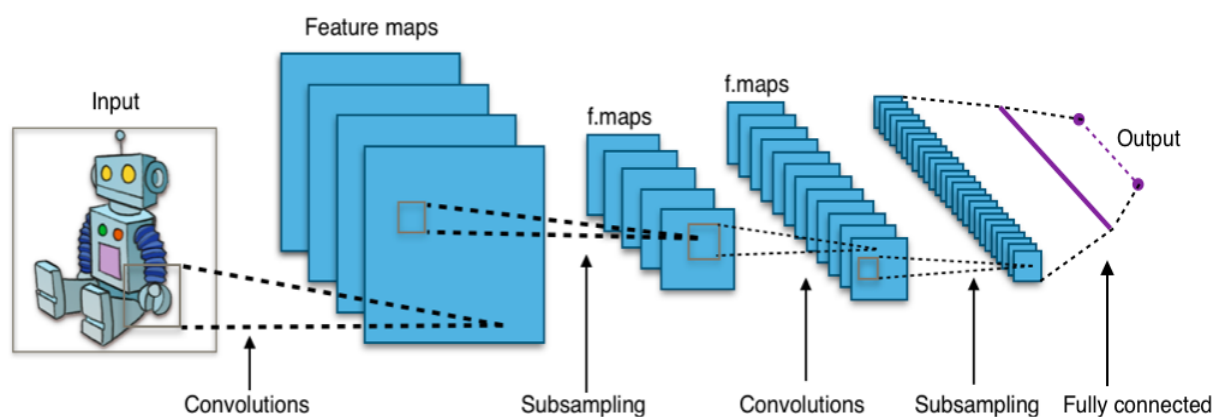


Рисунок 2 – Типовая архитектура свёрточной нейронной сети

Первые два типа слоев (convolutional, subsampling), чередуясь между собой, формируют входной вектор признаков для многослойного персептрона [4].

2.1.1 Свёрточный слой и операция свёртки

Краеугольным камнем всех нейронных сетей являются аффинные преобразования. В каждом слое полносвязной сети повторяется одна и та же операция: на вход подается вектор, который умножается на матрицу весов, а к результату добавляется вектор свободных членов; только после этого к результату применяется некая нелинейная функция активации. И во всех полносвязных сетях, такой подход используется постоянно, независимо от структуры или происхождения данных. Будь то изображения, текст или музыка, вновь и вновь применяется аффинное преобразование в каждом слое такой сети, предварительно приведя данные к векторной форме.

Однако многие типы данных имеют свою собственную внутреннюю структуру, которая отлично известна нам заранее. Главный пример такой структуры — изображение, которое обычно представляют как массив векторов чисел: если изображение черно-белое, то это просто массив интенсивностей, а если цветное, то массив векторов из трех чисел, обозначающих интенсивности трех основных цветов (красного, зеленого и черного в стандартном RGB, синего, зеленого и красного в трех типах колбочек в человеческом глазе и т. д.). Если же обобщить такую внутреннюю структуру до максимальной общности, описание получится такое:

1) исходные данные представляют собой многомерный массив («тензор»);

2) среди размерностей этого массива есть одна или более осей, порядок вдоль которых играет важную роль; например, это может быть расположение пикселей в изображении, временная шкала для музыкального произведения, порядок слов или символов в тексте;

3) другие оси обозначают «каналы», описывающие свойства каждого элемента по предыдущему подмножеству осей; например, три компонента для изображений, два компонента (правый и левый) для стереозвука и т. д.

Когда мы обучаем полносвязные нейронные сети, это дополнительное знание о структуре задачи никак не используется. Аффинные преобразования никак не учитывают структуру картинки, топологию данных.

Но ведь она не просто присутствует, а играет определяющую роль: конечно же, взаимное расположение пикселей в картинке важно для распознавания символов. Получается, что полносвязной сети приходится на основе интенсивностей отдельных пикселей выучить не только то, какая форма соответствует какому символу, но заодно и вообще понятие формы, тот факт, что некоторые компоненты входного вектора представляют собой соседние пиксели, а значит, они сильно скоррелированы, и так далее.

Основная идея сверточной сети состоит в том, что обработка участка изображения очень часто должна происходить независимо от конкретного расположения этого участка. Грубо говоря, если мы хотим распознать регистрационный номер автомобиля, совершенно не важно, на 200 или на 200 пикселей левый край номерной пластины отстоит от левого края фотографии. Узнать номер можно и на обрезанной фотографии, где нет ничего, кроме пластины; это локальная задача, которую можно решать локальными средствами. Конечно, взаимное расположение объектов играет важную роль, но сначала их нужно в любом случае распознать, и это распознавание — локально и независимо от конкретного положения участка с объектом внутри большой картинки.

Поэтому свёрточная сеть попросту делает это предположение в явном виде: давайте покроем вход небольшими окнами (скажем, 5×5 пикселей) и будем выделять признаки в каждом таком окне небольшой нейронной сетью. Причем — и тут ключевое соображение — признаки будем выделять в каждом окне одни и те же, то есть маленькая нейронная сеть будет всего одна, входов

у нее будет всего $5 \times 5 = 25$, а из каждой картинке для нее может получиться очень много разных входов.

Затем результаты этой нейронной сети опять можно будет представить в виде «картинки», заменяя окна 5×5 на их центральные пикселы, и на ней можно будет применить второй свёрточный слой, с уже другой маленькой нейронной сетью, и т. д [1].

Основное отличие полносвязного слоя от сверточного заключается в следующем: слои полносвязной сети изучают глобальные шаблоны в пространстве входных признаков, тогда как свёрточные слои изучают локальные шаблоны: в случае с изображениями — шаблоны в небольших двумерных окнах во входных данных. Пример работы с шаблонами показан на рисунке 3.

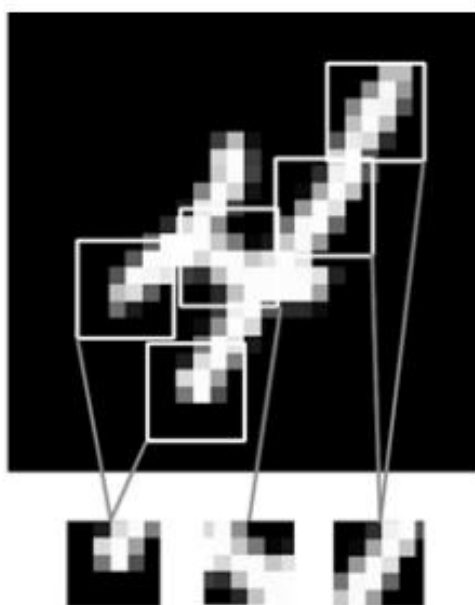


Рисунок 3 – Изображение можно разбить на локальные шаблоны, такие как края, текстуры и т. д.

Эта ключевая характеристика наделяет сверточные нейронные сети двумя важными свойствами:

– шаблоны, которые они изучают, являются инвариантными в отношении переноса. После изучения определенного шаблона в правом

нижнем углу картинке сверточная нейронная сеть сможет распознавать его повсюду: например, в левом верхнем углу. Полносвязной сети пришлось бы изучить шаблон заново, если он появляется в другом месте. Это увеличивает эффективность сверточных сетей в задачах обработки изображений: таким сетям требуется меньше обучающих образцов для получения представлений, обладающих силой обобщения;

– они могут изучать пространственные иерархии шаблонов. Первый сверточный слой будет изучать небольшие локальные шаблоны, такие как края, второй — более крупные шаблоны, состоящие из признаков, возвращаемых первым слоем, и т. д. Это позволяет свёрточным нейронным сетям эффективно изучать все более сложные и абстрактные визуальные представления [5]. Это свойство представлено на рисунке 4.

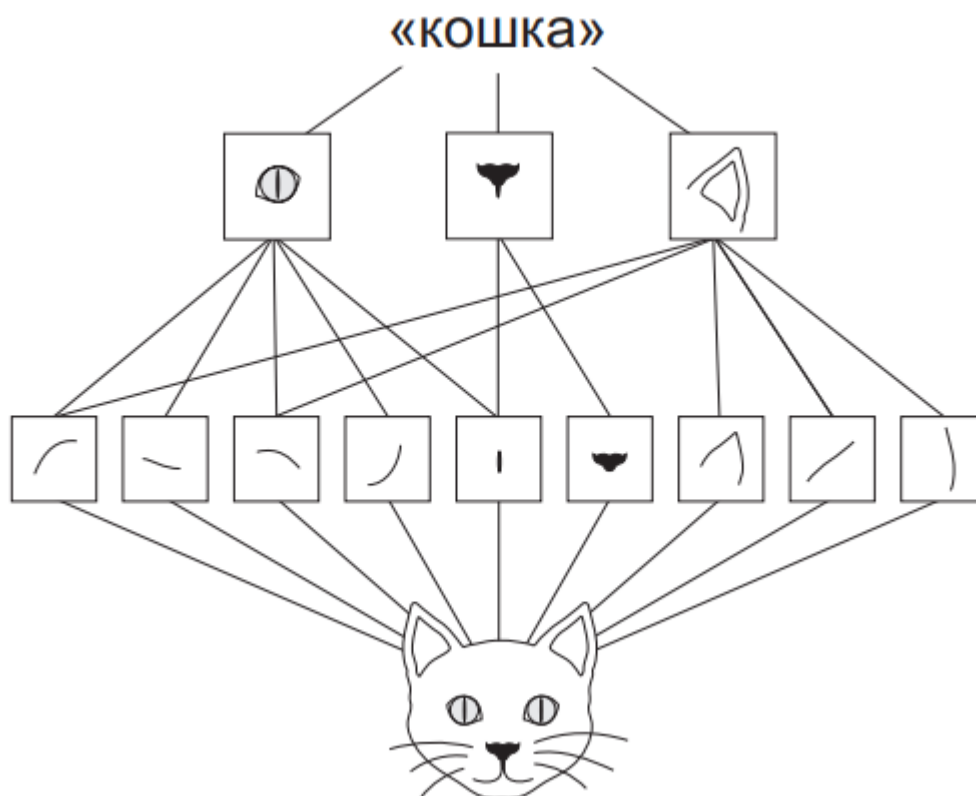


Рисунок 4 – Видимый мир формируется пространственными иерархиями видимых модулей: гиперлокальные края объединяются в локальные объекты, такие как глаза или уши, которые, в свою очередь, объединяются в понятия еще более высокого уровня, такие как «кошка»

Прежде чем перейти непосредственно к определениям операции свёртки нужно разобраться с понятием канала в изображении. Обычно цветные картинки, подающиеся на вход нейронной сети, представлены в виде нескольких прямоугольных матриц, каждая из которых задает уровень одного из цветовых каналов в каждом пикселе изображения. Картинка размером 200×200 пикселей — это на самом деле 120000 чисел, три матрицы интенсивностей размером 200×200 каждая. Если изображение черно-белое, то такая матрица будет одна. А если это не простая картинка, а, скажем, результат изображающей масс-спектрометрии, когда в каждом пикселе находится целый спектр, то матриц может быть очень много. Но в любом случае мы будем предполагать, что в каждом пикселе входного изображения стоит некоторый тензор (обычно одномерный, то есть вектор чисел), и его компоненты называются каналами (channels).

Такие же матрицы будут получаться и после свёрточного слоя: в них по-прежнему будет пространственная структура, соответствующая исходной картинке (но не в точности такая же), однако каналов теперь может стать больше. Значения каждого признака, которые выделяются из окон в исходном изображении, будут представлять собой целую матрицу. Каждая такая матрица называется картой признаков (feature map).

Осталось только формально определить, что же такое свертка и как устроены слои сверточной сети. Свертка — это всего лишь линейное преобразование входных данных особого вида. Если x^l — карта признаков в слое под номером l , то результат двумерной свёртки с ядром размера $2d + 1$ и матрицей весов W размера $(2d + 1) \times (2d + 1)$ на следующем слое будет так:

$$y_{i,j}^l = \sum_{-d \leq a, b \leq d} W_{a,b} x_{i+a, j+b}^l \quad (1)$$

где

$y_{i,j}^l$ — результат свёртки на уровне l ;

$x_{i,j}^l$ – её вход, то есть выход всего предыдущего слоя.

Иначе говоря, чтобы получить компоненту (i, j) следующего уровня, мы применяем линейное преобразование к квадратному окну предыдущего уровня, то есть скалярно умножаем пиксели из окна на вектор свертки. Это проиллюстрировано на рисунке 5: мы применяем свертку с матрицей весов W размера 3×3 к матрице X размера 5×5 [1].

$$\begin{pmatrix} 0 & 1 & 2 & 1 & 0 \\ 4 & 1 & 0 & 1 & 0 \\ 2 & 0 & 1 & 1 & 1 \\ 1 & 2 & 3 & 1 & 0 \\ 0 & 4 & 3 & 2 & 0 \end{pmatrix} * \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 2 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 9 & 5 & 4 \\ 8 & 8 & 10 \\ 8 & 15 & 12 \end{pmatrix} .$$

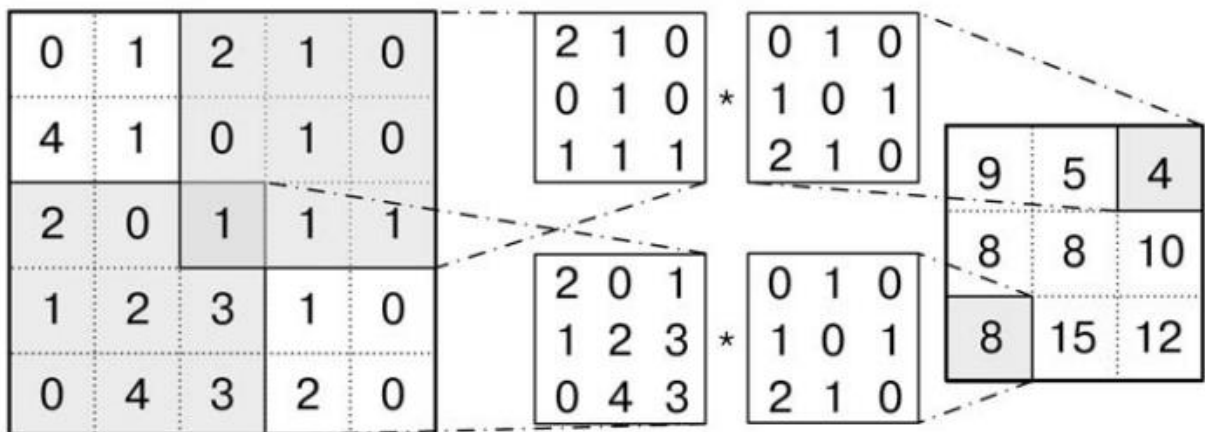


Рисунок 5 – Пример подсчёта результата свёртки: два примера подматрицы и общий результат

Свёртки определяются двумя ключевыми параметрами:

- размер шаблонов, извлекаемых из входных данных, — обычно 3×3 или 5×5;
- глубина выходной карты признаков — количество фильтров, вычисляемых сверткой.

Таким образом, свёртка работает методом скользящего окна: она двигает окно с размером 3×3 или 5×5 по трехмерной входной карте признаков, останавливается в каждой возможной позиции и извлекает трехмерный

шаблон окружающих признаков (с формой (высота_окна, ширина_окна, глубина_входа)). Каждый такой трехмерный шаблон затем преобразуется (путем умножения тензора на матрицу весов, получаемую в ходе обучения, которая называется ядром свертки) в одномерный вектор с формой (выходная глубина,). Все эти векторы затем собираются в трехмерную выходную карту с формой (высота, ширина, выходная глубина). Каждое пространственное местоположение в выходной карте признаков соответствует тому же местоположению во входной карте признаков (например, правый нижний угол выхода содержит информацию о правом нижнем угле входа). Полный процесс изображен на рисунке 6 [5].

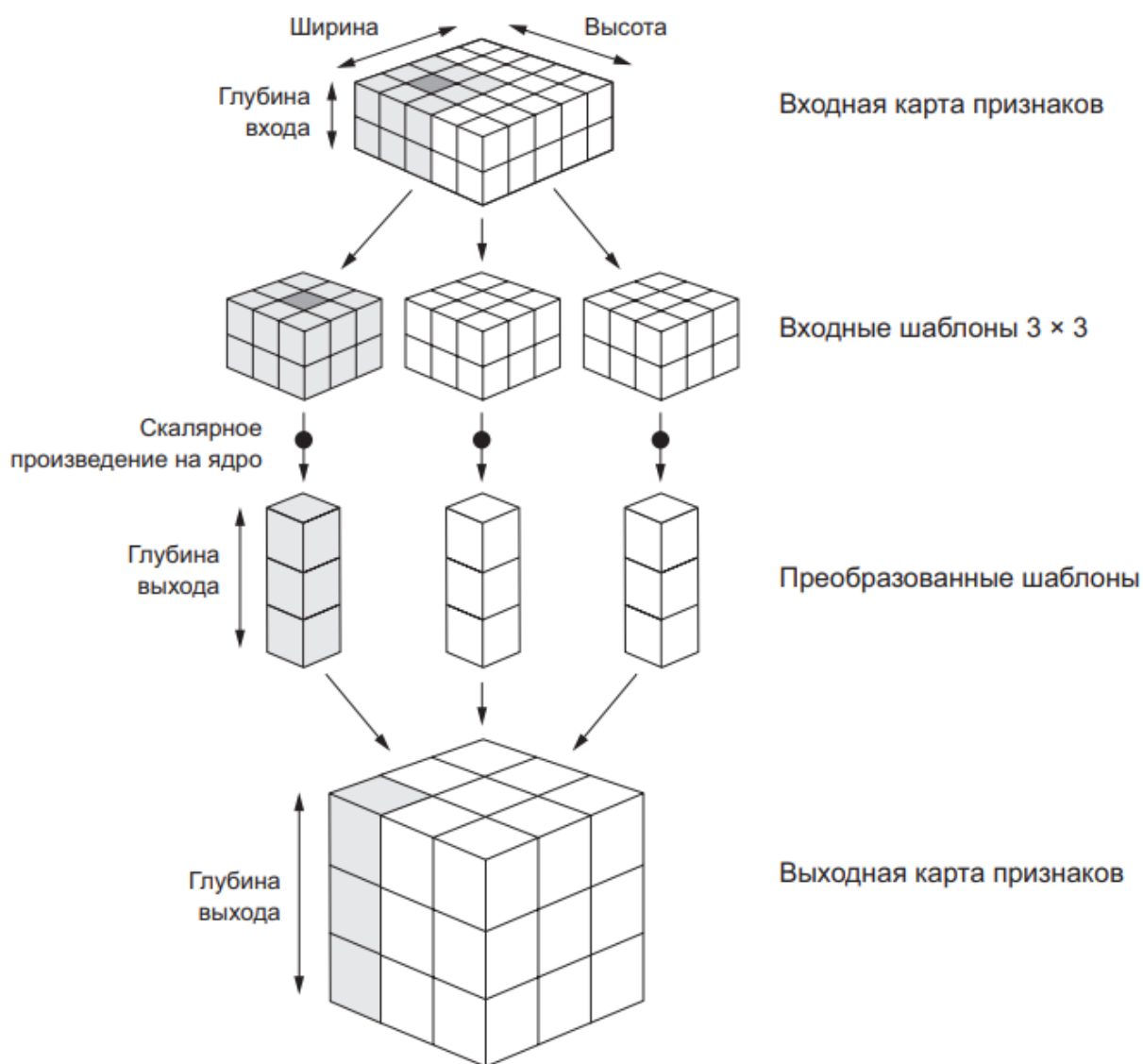


Рисунок 6 – Принцип действия свёртки

Свёртка обладает свойствами, о которых было сказано выше:

– свёртка сохраняет структуру входа (порядок в одномерном случае, взаимное расположение пикселей в двумерном и т. д.), так как применяется к каждому участку входных данных в отдельности;

– операция свёртки обладает свойством разреженности, так как значение каждого нейрона очередного слоя зависит только от небольшой доли входных нейронов (а, например, в полносвязной нейронной сети каждый нейрон зависел бы от всех нейронов предыдущего слоя);

– свёртка многократно переиспользует одни и те же веса, так как они повторно применяются к различным участкам входа [1].

В упрощённом виде свёрточный слой можно описать формулой:

$$z^l = f(y^l + b^l) \quad (2)$$

где

z^l – выход слоя l ;

y^l – её вход (результат свёртки);

$f()$ – функция активации;

b^l – коэффициент сдвига слоя l [4].

Также нужно обратить внимание на то, что выходные ширина и высота могут отличаться от входных. На то есть две причины:

– эффекты границ, которые могут устраняться дополнением входной карты признаков;

– использование шага свертки.

—Эффекты границ и дополнение

Рассмотрим карту признаков 5×5 (всего 25 клеток). Существует всего 9 клеток, в которых может находиться центр окна 3×3 , образующих сетку 3×3 , это показано на рисунке 7. Следовательно, карта выходных признаков будет

иметь размер 3×3 . Она получилась немного сжатой: ровно на две клетки вдоль каждого измерения. Можно увидеть, как проявляется эффект границ на более раннем примере: изначально у нас имелось 28×28 входов, количество которых после первого сверточного слоя сократилось до 26×26 .

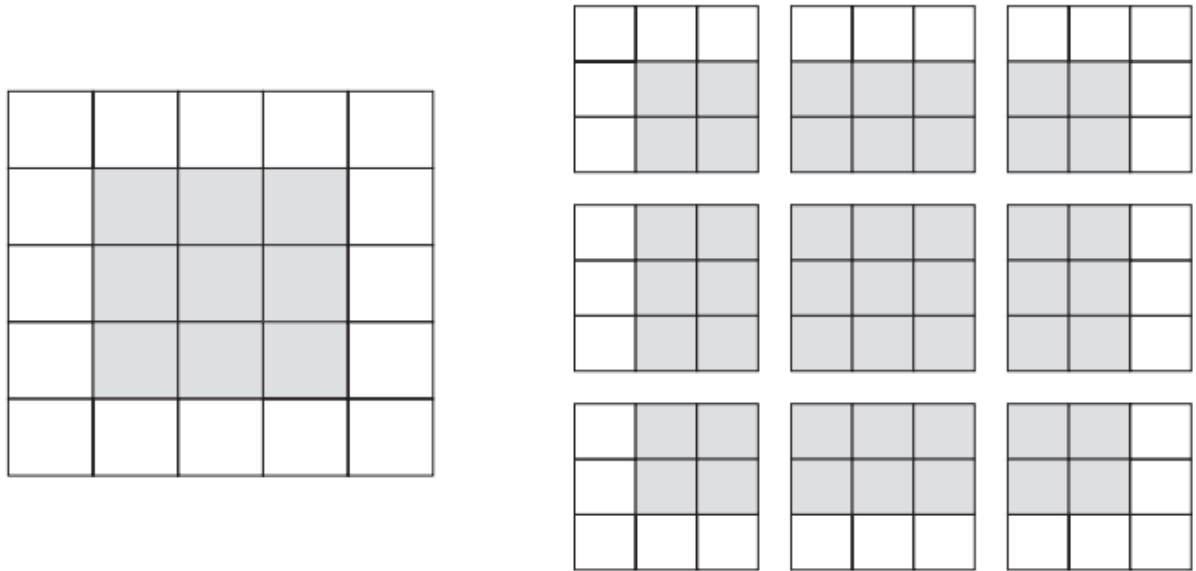


Рисунок 7 – Допустимые местоположения шаблонов 3×3 во входной карте признаков 5×5

При этом за счёт эффекта границ размер исходных матриц уменьшается, формула (3):

$$z_j^l = f(\sum_i y_i^l + b_j^l) \quad (3)$$

где

z_j^l – карта признаков j (выход слоя l);

y_i^l – её вход (результат свёртки);

$f()$ – функция активации;

b_j^l – коэффициент сдвига слоя l для карты признаков j [4].

Чтобы получить выходную карту признаков с теми же пространственными размерами, что и входная карта, можно использовать дополнение (padding). Дополнение заключается в добавлении соответствующего количества строк и столбцов с каждой стороны входной карты признаков, чтобы можно было поместить центр окна свертки в каждую входную клетку. Для окна 3×3 нужно добавить один столбец справа, один столбец слева, одну строку сверху и одну строку снизу. Для окна 5×5 нужно добавить две строки. Такое добавление проиллюстрировано на рисунке 8.

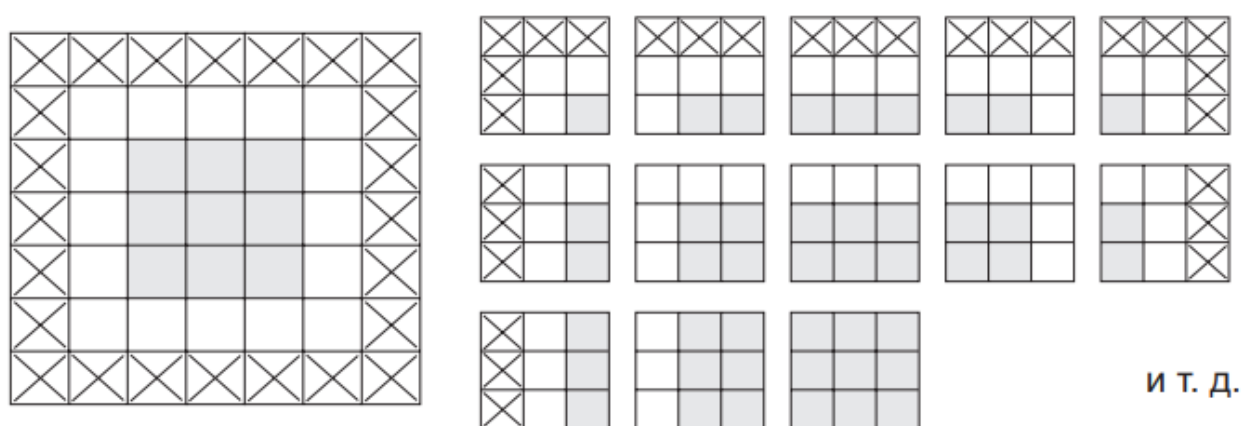


Рисунок 8 – Дополнение входной карты признаков 5×5 , чтобы получить 25 шаблонов 3×3

—Шаг свёртки

Другой фактор, который может влиять на размер выходной карты признаков, — шаг свертки. До сих пор в объяснениях выше предполагалось, что центральная клетка окна свертки последовательно перемещается в смежные клетки входной карты. Однако в общем случае расстояние между двумя соседними окнами является настраиваемым параметром, который называется шагом свертки и по умолчанию равен 1. Также имеется возможность определять свертки с пробелами (strided convolutions) — свертки с шагом больше 1. На рисунке 9 можно видеть, как извлекаются шаблоны 3×3 сверткой с шагом 2 из входной карты 5×5 (без дополнения).

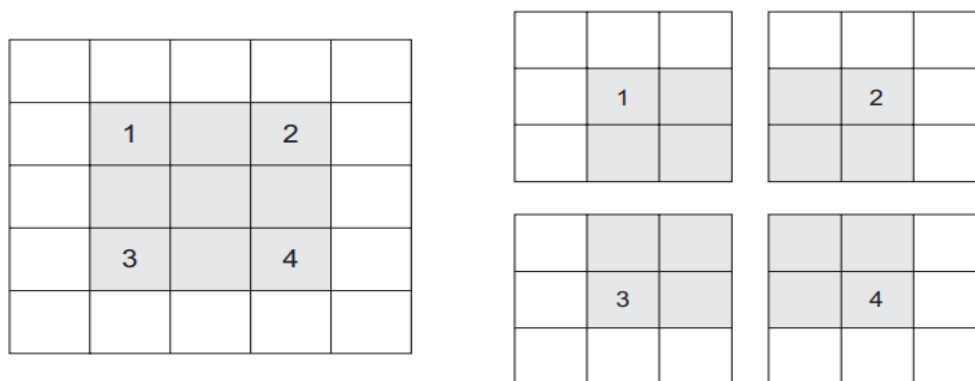


Рисунок 9 – Шаблоны 3×3 свёртки с шагом 2×2

Использование шага 2 означает уменьшение ширины и высоты карты признаков за счет уменьшения разрешения в два раза (в дополнение к любым изменениям, вызванным эффектами границ). Свертки с пробелами редко используются на практике [5].

2.1.2 Субдискретизирующий слой

В свёрточных нейронных сетях со свёрточными слоями чередуются субдискретизирующие слои (первым слоем обязательно должен быть свёрточный), в котором используется операция субдискретизации.

Смысл этой операции заключается в следующем: в свёрточных сетях обычно исходят из предположения, что наличие или отсутствие того или иного признака гораздо важнее, чем его точные координаты. Например, при распознавании номерных пластин свёрточной сетью нам гораздо важнее понять, есть ли на фотографии пластина, чем узнать, с какого конкретно пиксела она начинается и в каком заканчивается. Поэтому можно позволить себе «обобщить» выделяемые признаки, потеряв часть информации об их местоположении, но зато сократив размерность.

Обычно в качестве операции субдискретизации к каждой локальной группе нейронов применяется операция взятия максимума (max-pooling). Формально определим субдискретизацию так:

$$y_{i,j}^{l+1} = \max_{-d \leq a \leq d, -d \leq b \leq d} z_{i+a, j+b}^l \quad (4)$$

где

$y_{i,j}^{l+1}$ – результат субдискретизации на уровне $l+1$;

$z_{i,j}^l$ – выход слоя l .

Здесь d – это размер окна субдискретизации. Как правило, нас интересует случай, когда шаг субдискретизации и размер окна совпадают, то есть получаемая на вход матрица делится на непересекающиеся окна, в каждом из которых мы выбираем максимум; для $d = 2$ эта ситуация проиллюстрирована на рисунке 10.

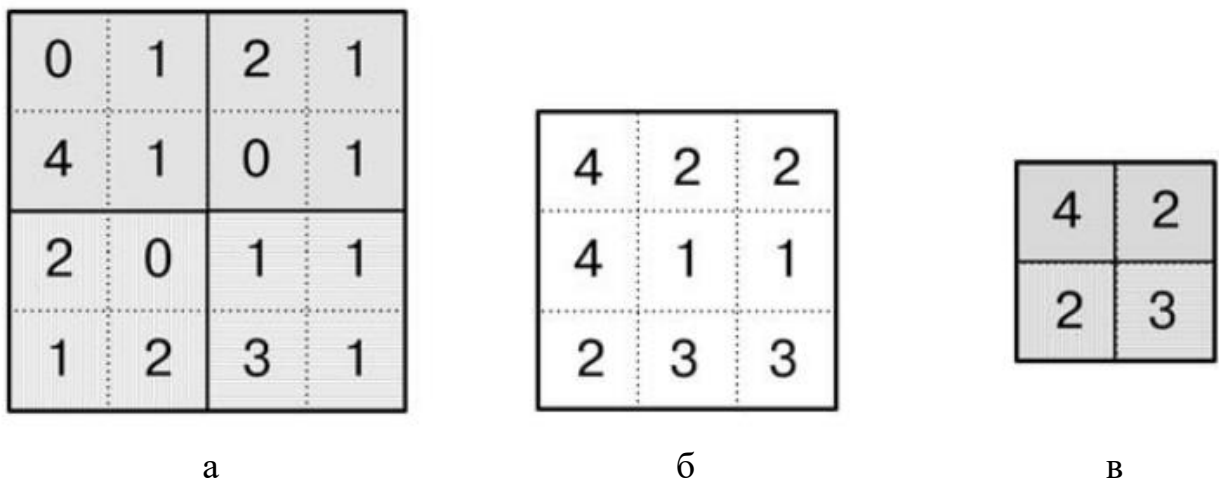


Рисунок 10 – Пример субдискретизации с окном размера 2×2 : а — исходная матрица; б — матрица после субдискретизации с шагом 1; в – матрица после субдискретизации с шагом 2.

Хотя в результате субдискретизации действительно теряется часть информации, сеть становится более устойчивой к небольшим трансформациям изображения вроде сдвига или поворота [1].

Субдискретизирующий слой можно описать формулой:

$$z^{l+1} = f(a^{l+1}y^{l+1} + b^{l+1}) \quad (5)$$

где

z^{l+1} – выход слоя $l+1$;

y^{l+1} – её вход (результат субдискретизации);

$f()$ – функция активации;

a^{l+1}, b^{l+1} – коэффициенты сдвига слоя $l+1$.

2.1.3 Полносвязный слой

После чередования сверточных слоёв и слоёв субдискретизации в конце идёт слой обычного многослойного персептрона. Цель слоя – классификация, он моделирует сложную нелинейную функцию, оптимизируя которую, улучшается качество распознавания.

Нейроны каждой карты предыдущего субдискретизирующего слоя связаны с одним нейроном скрытого слоя. Таким образом число нейронов скрытого слоя равно числу карт субдискретизирующего слоя, но связи могут быть не обязательно такими, например, только часть нейронов какой-либо из карт субдискретизирующего слоя быть связана с первым нейроном скрытого слоя, а оставшаяся часть со вторым, либо все нейроны первой карты связаны с нейронами 1 и 2 скрытого слоя. Вычисление значений нейрона можно описать формулой (6):

$$z_j^l = f(\sum_i z_i^{l-1} w_{i,j}^{l-1} + b_j^{l-1}) \quad (6)$$

где

z_j^l – карта признаков j (выход слоя l);

$f()$ – функция активации;

b_j^l – коэффициент сдвига слоя l ;

$w_{i,j}^l$ – матрица весовых коэффициентов слоя l [4].

3 Программная реализация

3.1 Выбор языка программирования

В качестве языка для моей программы был выбран высокоуровневый объектно-ориентированный язык общего назначения Python. Python – это интерпретируемый язык программирования. Исходный код преобразуется в машинный частями по мере его выполнения специальной программой, которая называется интерпретатор. Python отличается своим уникальным синтаксисом, он ориентирован на повышение читаемости кода и производительности программиста. Его отличительными особенностями являются: интроспекция, динамическая типизация, автоматическое управление памятью, механизм обработки исключений, удобные в использовании высокоуровневые структуры данных. Язык был создан в 1989—1991-х годах Гвидо Ван Россумом, и с того момента быстро стал популярен и востребован у программистов. Разработка и поддержка языка продолжается до сих пор. На данном этапе поддерживаются две ветки языка Python 3.x и Python 2.x. Этот язык создавался под влиянием других языков программирования, так, например, от языка АВС он позаимствовал отступы и высокоуровневые структуры данных, от Fortran появились срезы массивов, от C/C++ – некоторые синтаксические конструкторы и т. д. [6][7]

3.2 Используемые пакеты

— OpenCV

Это библиотека компьютерного зрения и машинного обучения с открытым исходным кодом, которая реализована на C++, однако имеет оболочку для Python, Java и других языков. Библиотека содержит более 2500 оптимизированных алгоритмов, включая полный набор как классических, так и современных алгоритмов компьютерного зрения и машинного обучения.

— NumPy

Это основной пакет, использующийся для научных вычислений при написании программ на языке Python. Он позволяет выполнять основные операции над n-мерными массивами и матрицами. Благодаря механизму векторизации NumPy повышает производительность и, соответственно, ускоряет выполнение операций.

— Scikit-image

Библиотека, предназначенная для обработки изображений.

— matplotlib

Это библиотека для визуализации обработанных данных двумерной и трёхмерной графикой. Она является основным инструментом для визуализации данных на языке Python, поддерживается различными платформами и IDE (iPython, Jupyter и пр.).

— TensorFlow

Открытая программная библиотека для машинного обучения, разработанная компанией Google для решения задач построения и тренировки нейронной сети с целью автоматического нахождения и классификации образов. Библиотека использует многоуровневую систему узлов для обработки большого количества данных, что расширяет сферу ее использования далеко за пределы научной области.

— PyTesseract

Это инструмент оптического распознавания символов (OCR) для Python. Является оболочкой для Tesseract OCR – свободной компьютерной программы для распознавания текстов.

3.3 Выделение номера с помощью свёрточной сети

3.3.1 Построение сети

Для выделения прямоугольной области, содержащей номерную пластину, с помощью библиотеки TensorFlow была разработана свёрточная нейронная сеть, схема которой показана на рисунке 11. А на рисунке 12 изображена схема программной реализации данной сети.

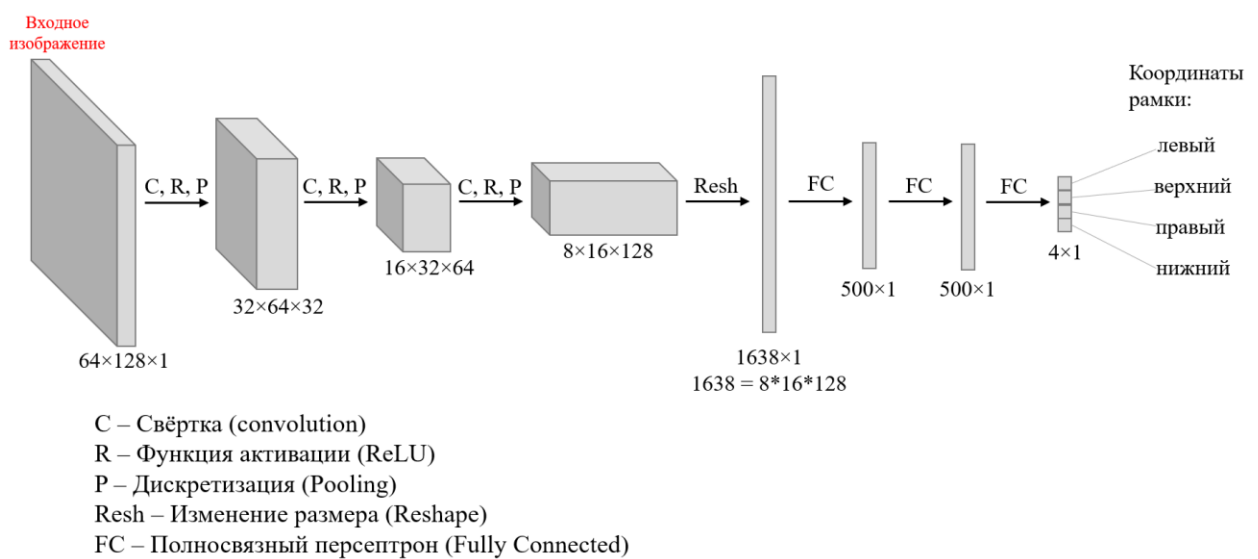


Рисунок 11 – Схема модели СНС

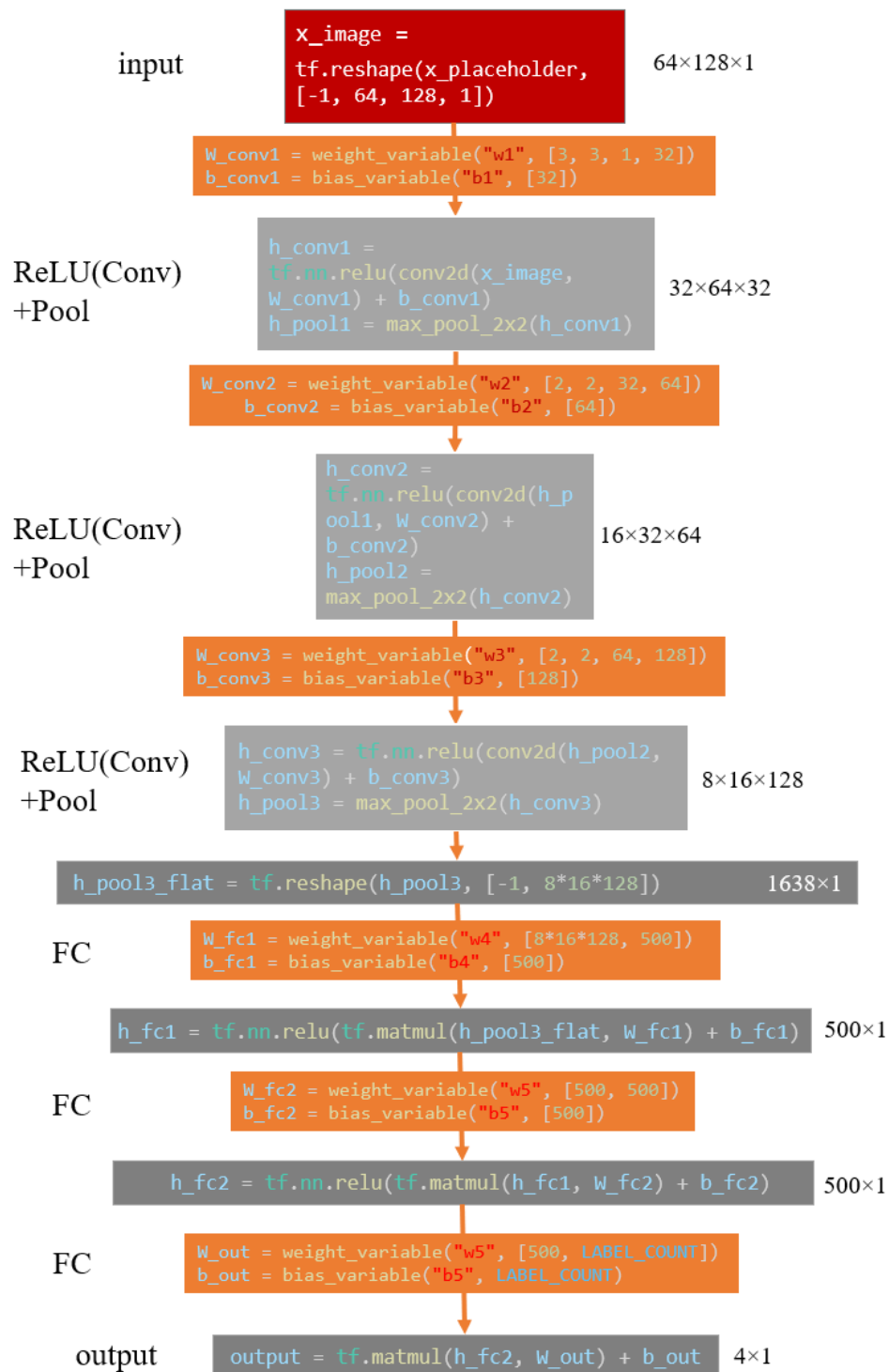


Рисунок 12 – Программная реализация СНС

Как видно из схемы, сеть состоит из трёх пар слоёв свёртки и субдискретизации, а также трёх слоёв полносвязного персептрона.

На вход сети подается переформатированный тензор в виде двумерного массива размером 64×128 пикселей (форматирование изображения в нужный тензор в TensorFlow делается с помощью функции `reshape`):

```
x_image = tf.reshape(x_placeholder, [-1, 64, 128, 1]).
```

Нужно обратить внимание на то, что формально массив теперь четырехмерный: первая размерность -1 соответствует заранее неизвестному размеру мини-батча, а в четвертой размерности указано, что в каждом пикселе стоит только одно число.

Далее идёт первый свёрточный слой. Для него было выбрано ядро размером 3×3 , число фильтров $- 32$ и число каналов $- 1$ (так как изображение чёрно-белое). Переменные для весов свёртки:

```
W_conv1 = weight_variable("w1", [3, 3, 1, 32]),
```

где `weight_variable` – функция инициализации веса:

```
def weight_variable(name, shape):  
    initial = tf.truncated_normal(shape, stddev=0.1)  
    return tf.get_variable(name, initializer=initial),
```

а массив из четырех чисел на входе — это форма тензора, который мы инициализируем с помощью нормального распределения. Первые два параметра задают размер ядра, третий отвечает за число входных каналов, а четвертый определяет собственно число выходных каналов. По сути, двигая наше окно-фильтр по исходному изображению, мы на выходе получаем вместо одного значения столбик из 32 значений, что можно представить себе как применение 32 разных фильтров. После весов задаются свободные члены (`biases`); для них достаточно отвести всего 32 переменные: для каждого из 32 фильтров:

```
b_conv1 = bias_variable("b1", [32]),
```

где `bias_variable` – функция инициализации свободных членов:

```
def bias_variable(name, shape):  
    initial = tf.constant(0.1, shape=shape)  
    return tf.get_variable(name, initializer=initial).
```

Осталось только применить свёрточные фильтры и использовать функцию активации:

```
h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1),
```

где `conv2d` – функция для свёрточных фильтров:

```
def conv2d(x, W):  
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1],  
padding='SAME'),
```

а `tf.nn.relu()` – функция активации.

Стоит заметить, что в сети используется выпрямленная линейная функция активации (Rectified Linear Unit, ReLU), так как она лишена ресурсоёмких операций, а также отсекает ненужные детали [8]. Также стоит отметить, что эта функция активации будет использоваться во всей сети.

Чтобы соблюсти стандартную архитектуру свёрточной сети, осталось только добавить слой субдискретизации:

```
h_pool1 = max_pool_2x2(h_conv1),
```

где `max_pool_2x2` – функция субдискретизации с размером окна 2×2 :

```
def max_pool_2x2(x):  
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1], strides=[1, 2, 2,  
1], padding='SAME').
```

В итоге, после первой пары слоёв выходит тензор размером $32 \times 64 \times 32$.

Далее создаётся вторая пара слоёв, в которой слой свёртки с ядром 2×2 , числом фильтров равным 64 и числом каналов равным 32. В конце выходит тензор размером $16 \times 32 \times 64$.

И последним создаётся третья пара слоёв. В ней свёрточный слой имеет ядро такого же размера, как и в предыдущей паре (2×2), только число фильтров равно 128, а число каналов – 64. В итоге получается тензор размером $8 \times 16 \times 128$.

Как правило, в глубоких нейронных сетях за свёрточными слоями следуют полносвязные, задача которых состоит в том, чтобы «собрать вместе» все признаки из фильтров и, собственно, перевести их в самый последний

слой, который выдаст ответ. Но для начала нам нужно из трёхмерного слоя сделать плоский; также будет использована функция `reshape`:

```
h_pool3_flat = tf.reshape(h_pool3, [-1, 8*16*128]).
```

Число $8*16*128$ возникло из-за того, что мы трижды применили субдискретизацию и при этом в последнем слое использовали 128 фильтров.

И теперь осталось только добавить полносвязные слои. Добавляем первый слой из 500 нейронов:

```
W_fc1 = weight_variable("w4", [8*16*128, 500])
b_fc1 = bias_variable("b4", [500])
h_fc1 = tf.nn.relu(tf.matmul(h_pool3_flat, W_fc1) + b_fc1),
```

где `tf.matmul(a, b)` – функция TensorFlow, которая умножает матрицу `a` на матрицу `b`. Далее добавляется слой так же из 500 нейронов. В конце добавляется третий, самый последний слой с четырьмя выходами:

```
W_out = weight_variable("w6", [500, LABEL_COUNT])
b_out = bias_variable("b6", [LABEL_COUNT])
output = tf.matmul(h_fc2, W_out) + b_out
```

В итоге эта сеть должна предсказать координаты точек в углах номерной пластины.

3.3.2 Обучение сети

Обучение сети происходило с помощью набора данных `Licence plates`, составленного сервисом `Supervisely` [9]. Этот набор составлен из искусственно созданных чёрно-белых случайных изображений с наложенными на них номерами. Каждое изображение имеет размер 64×128 пикселей. На рисунке 13 представлен пример изображений, находящийся в этом наборе данных.

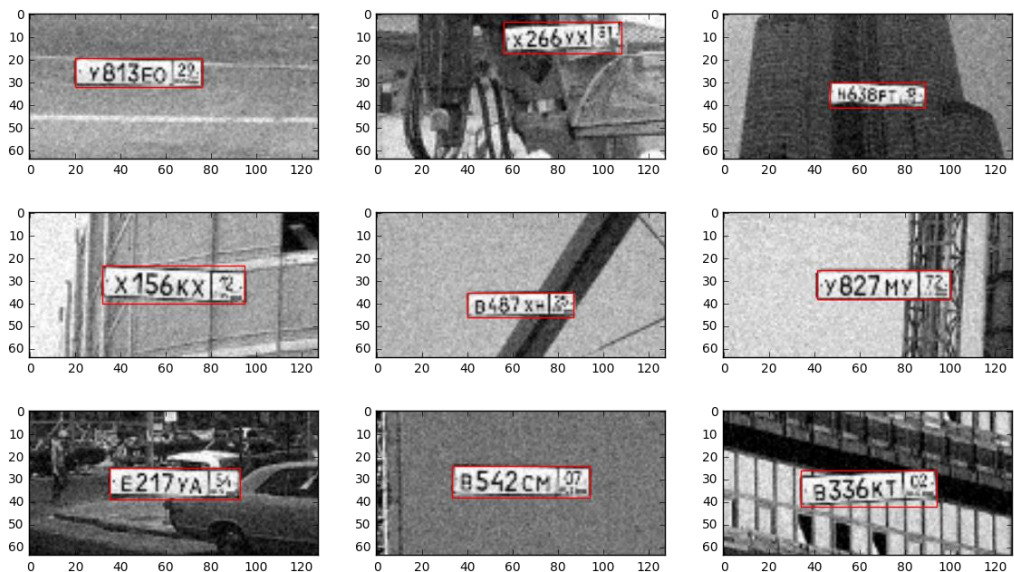


Рисунок 13 – Выборка из набора данных

Создаём переменную для тренировочных данных:

```
x_placeholder = tf.placeholder(tf.float32, shape=[None,
PIXEL_COUNT]).
```

В данном случае `x_placeholder` — это не какой-то заранее заданный тензор, а так называемая заглушка (`placeholder`), которую заполним, когда попросим TensorFlow произвести вычисления.

Для того чтобы обучить модель, нужно также зафиксировать некий способ оценки качества предсказаний (именно эту оценку мы и будем в конечном счете оптимизировать). Опишем в терминах TensorFlow функцию потерь. Для исходной разметки нам понадобится заглушка:

```
y_placeholder = tf.placeholder(tf.float32, shape=[None,
LABEL_COUNT]).
```

Записываем функцию потерь:

```
loss = mse(model.y_placeholder, model.output),
```

где `mse()` – функция потерь:

```
def mse(expected, predicted):
    se = tf.square(expected - predicted)
    return tf.reduce_mean(se).
```

В качестве функции потери была использована среднеквадратичная ошибка (Mean Squared Error, MSE) [10], которая вычисляется по формуле:

$$MSE = \frac{1}{n} * \sum_{i=1}^n (y_i - x_i) \quad (7)$$

где

n – размер выборки;

y_i – фактическое значение данных;

x_i – прогнозируемое значение данных.

Для обучения модели использован алгоритм оптимизации Adam – адаптивный вариант градиентного спуска, являющийся модификацией Adagrad, но использует сглаженные версии среднего и среднеквадратичного градиентов [11]:

$$m_t = \beta_1 m + (1 - \beta_1) g_t \quad (8)$$

$$v_t = \beta_2 m + (1 - \beta_1) g_t^2 \quad (9)$$

$$u_t = \frac{\eta}{\sqrt{v_t + \epsilon}} \quad (10)$$

```
train_step = tf.train.AdamOptimizer().minimize(loss).
```

Осталось только запустить обучение и дождаться результата. Ход обучения представлен на рисунке 14.

```
session = tf.InteractiveSession()
session.run(tf.global_variables_initializer())
#saver.restore(session, os.path.join(MODEL_PATH, "model"))

last_epoch = -1
while dataset.epoch_completed() < EPOCH:
    (batch_x, batch_y) = dataset.next_batch(20)
    train_step.run(feed_dict={model.x_placeholder: batch_x, model.y_placeholder: batch_y})
    if dataset.epoch_completed() > last_epoch:
        last_epoch = dataset.epoch_completed()
        score_test = loss.eval(feed_dict={model.x_placeholder: X2_test, model.y_placeholder: Y2_test})
        if score_test < best_score:
            best_score = score_test
            saver.save(session, os.path.join(MODEL_PATH, "model"))
        if dataset.epoch_completed() % 1 == 0:
            epm = 60 * dataset.epoch_completed() / (time.time()-start_time)
            print('Epoch: %d, Score: %f, Epoch per minute: %f' % (dataset.epoch_completed(), score_test, epm))
print('Finished in %f seconds.' % (time.time()-start_time))

session.close()
```

Output exceeds the [size limit](#). Open the full output data [in a text editor](#)

```
Epoch: 0, Score: 1255.765381, Epoch per minute: 0.000000
Epoch: 1, Score: 0.028268, Epoch per minute: 7.074798
Epoch: 2, Score: 0.020439, Epoch per minute: 7.662754
Epoch: 3, Score: 0.010363, Epoch per minute: 7.901268
Epoch: 4, Score: 0.007444, Epoch per minute: 8.075397
Epoch: 5, Score: 0.005275, Epoch per minute: 8.170623
Epoch: 6, Score: 0.004199, Epoch per minute: 8.228775
Epoch: 7, Score: 0.003650, Epoch per minute: 8.267812
Epoch: 8, Score: 0.002536, Epoch per minute: 8.292307
Epoch: 9, Score: 0.002299, Epoch per minute: 8.310922
Epoch: 10, Score: 0.001979, Epoch per minute: 8.319011
Epoch: 11, Score: 0.001762, Epoch per minute: 8.317294
Epoch: 12, Score: 0.001669, Epoch per minute: 8.316186
Epoch: 13, Score: 0.001734, Epoch per minute: 8.356227
Epoch: 14, Score: 0.001490, Epoch per minute: 8.330120
Epoch: 15, Score: 0.001197, Epoch per minute: 8.336434
Epoch: 16, Score: 0.001681, Epoch per minute: 8.359461
Epoch: 17, Score: 0.001736, Epoch per minute: 8.382718
Epoch: 18, Score: 0.001123, Epoch per minute: 8.389469
Epoch: 19, Score: 0.001067, Epoch per minute: 8.395933
Epoch: 20, Score: 0.001518, Epoch per minute: 8.409458
Epoch: 21, Score: 0.001117, Epoch per minute: 8.420229
Epoch: 22, Score: 0.001122, Epoch per minute: 8.415671
Epoch: 23, Score: 0.001175, Epoch per minute: 8.409289
Epoch: 24, Score: 0.001589, Epoch per minute: 8.409819
...
Epoch: 28, Score: 0.000905, Epoch per minute: 8.422991
Epoch: 29, Score: 0.000923, Epoch per minute: 8.425583
Epoch: 30, Score: 0.000958, Epoch per minute: 8.433249
Finished in 213.441181 seconds.
```

Рисунок 14 – Обучение модели

Всего было задано 30 эпох. Обучение прошло всего за 213 секунд и общие потери составляли около 0.00096.

После обучения модель можно проверить на тестовой выборке. Результат работы показан на рисунке 15.

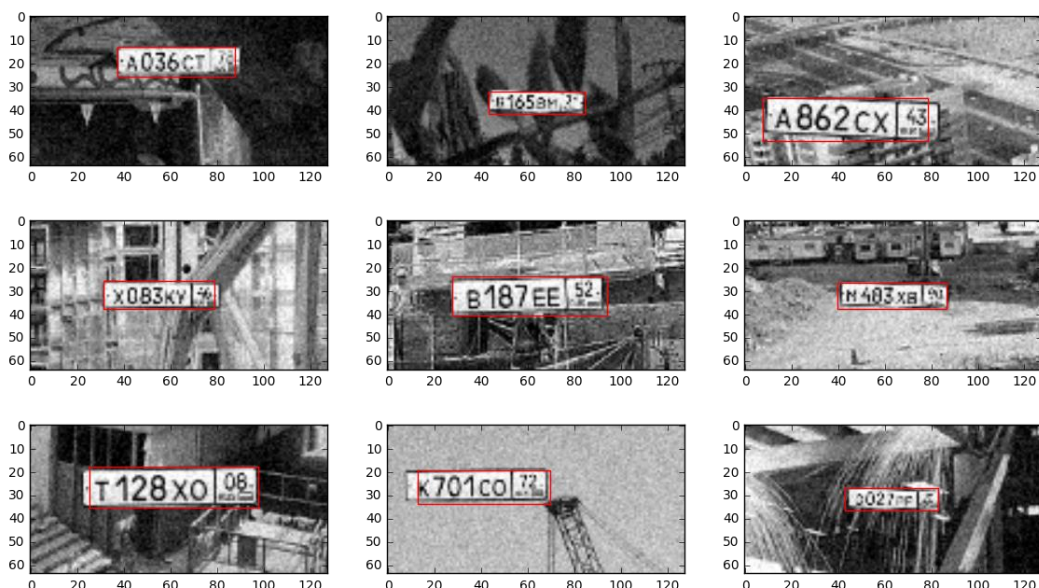


Рисунок 15 – Результат тестирования модели

Из рисунка видно, что модель достаточно чётко определила местонахождения номеров на изображениях.

3.4 Распознавание номера

3.4.1 Выбор модели для выделения номера

Однако для системы распознавания номеров будет использована уже готовая модель SSD ResNet50 V1 FPN 1024x1024, так как созданная модель имеет серьёзные ограничения в выборе изображений (чёрно-белые с разрешением 64×128 пикселей), а используемая модель работает с цветными изображениями размером 1024×1024 пикселей. Модель была дотренирована с помощью набора данных о парковке в китайском городе CCPD [12]. Результат работы модели продемонстрирован на рисунке 16.

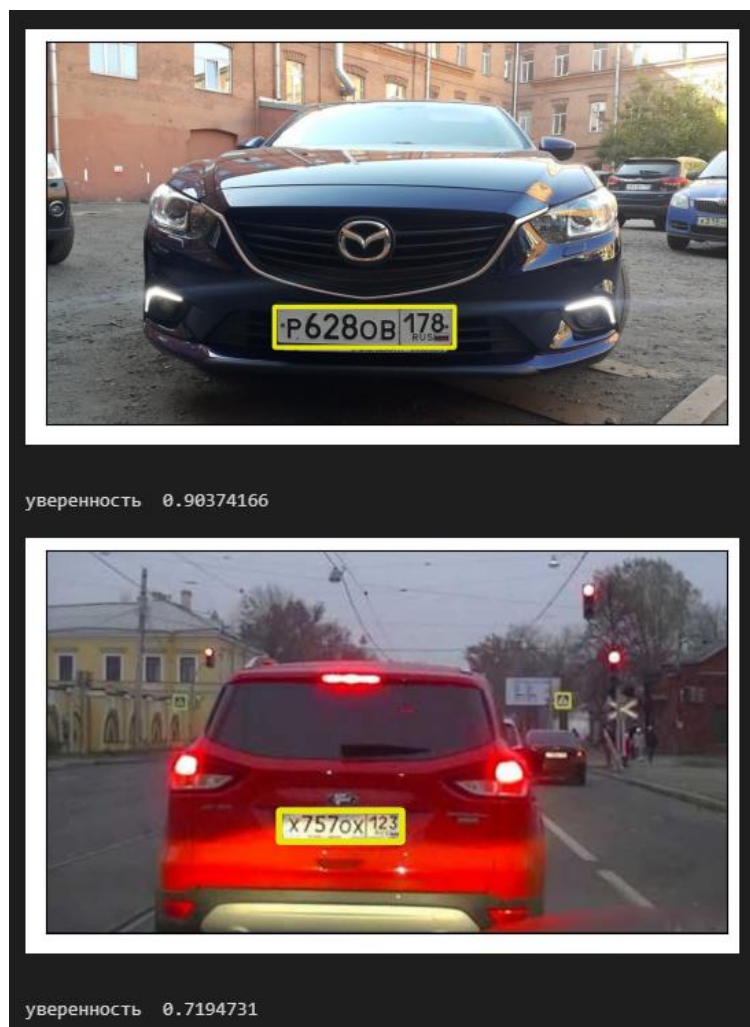


Рисунок 16 – Результат тестового запуска

3.4.2 Подготовка изображения

Прежде чем распознавать символы из выделенной области, нужно предварительно обработать изображение для более точного результата. Обработка изображений будет производиться с помощью библиотеки OpenCV и Scikit-Image.

Обработку изображения можно представить следующей последовательностью этапов:

1) с помощью модели, описанной выше, оставляется только номер. Результат этого этапа продемонстрирован на рисунке 17;

2) изображение выравнивается с помощью преобразования Хафа. Результат этого этапа продемонстрирован на рисунке 18;

3) увеличивается контрастность изображения. Результат этого этапа продемонстрирован на рисунке 19;

4) изображение переводится из цветного в чёрно-белое. Результат этого этапа продемонстрирован на рисунке 20;

5) с помощью двустороннего фильтра убирается лишний шум. Результат этого этапа продемонстрирован на рисунке 21.

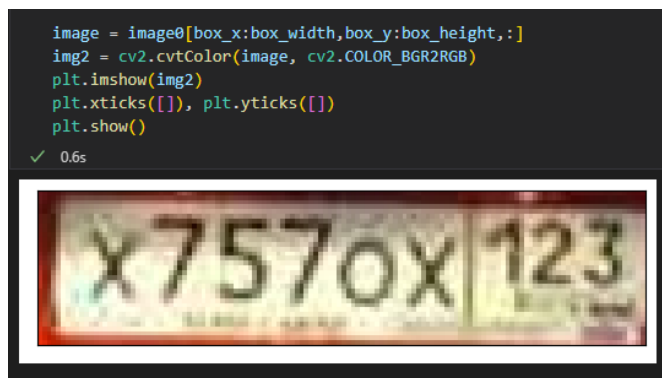


Рисунок 17 – Обрезка номерной пластины



Рисунок 18 – Выравнивание изображения

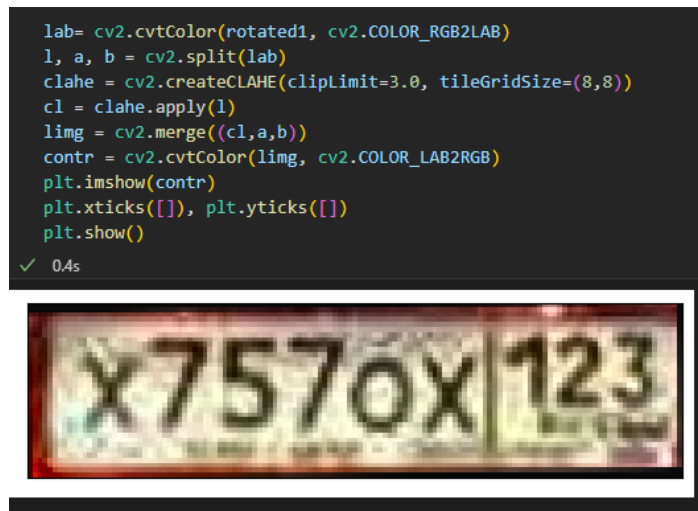


Рисунок 19 – Увеличение контрастности

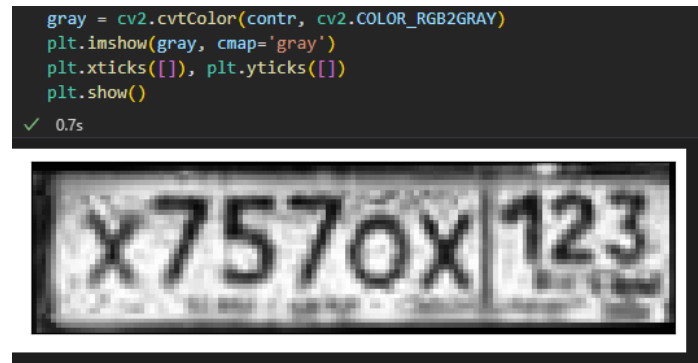


Рисунок 20 – Перевод из цветного в чёрно-белое

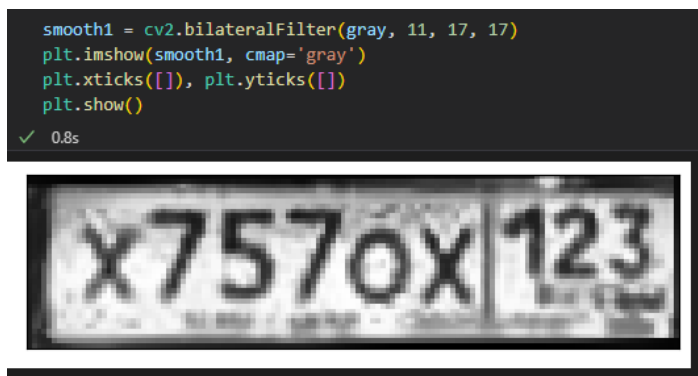


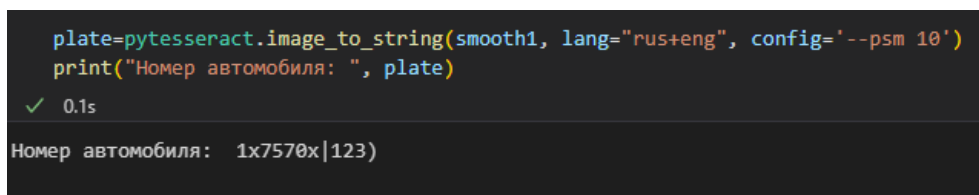
Рисунок 21 – Избавление от лишнего шума

3.4.3 Распознавание символов

Последним этапом в распознавании регистрационного номера – это фактическое считывание информации о номере с сегментированного изображения. Для распознавания символов из изображения использовалась программа Tesseract-OCR и оболочка для работы с ней на языке Python – PyTesseract:

```
plate=pytesseract.image_to_string(smooth1, lang="rus+eng",  
config='--psm 10')  
print("Номер автомобиля: ", plate)
```

Результат выполнения представлен на рисунке 22.



```
plate=pytesseract.image_to_string(smooth1, lang="rus+eng", config='--psm 10')  
print("Номер автомобиля: ", plate)  
✓ 0.1s  
Номер автомобиля: 1x7570x|123)
```

Рисунок 22 – Результат работы программы

По итогу данная программа была протестирована на 10 изображениях, вероятность обнаружения автомобильного номера составила 100%, а точность распознавания символов составила 70%

ЗАКЛЮЧЕНИЕ

В рамках данной курсовой работы был исследован метод, позволяющий выполнить распознавание регистрационных автомобильных номеров на изображении. Рассмотрены библиотеки, которые делают это возможным, а также на основе полученной информации реализован подход к распознаванию номеров (свёрточная нейронная сеть). Были выявлены преимущества и недостатки данного алгоритма, а также проведено тестирование из данных, полученных в условиях реального транспортного движения.

Несмотря на то, что на сегодняшний день существует достаточно большое количество систем, основанных на свёрточных нейронных сетях, данная область не прекращает развиваться и охватывает всё больше сфер человеческой деятельности, что позволяет её быть востребованной в течение долгого периода времени.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Николенко, С. Глубокое обучение / С. Николенко, А. Кадурин, Е. Архангельская. – Санкт-Петербург: Питер, 2018. – 480 с. – ISBN 978-5-496-02536-2.
2. Шакла, Н. Машинное обучение и TensorFlow / Н. Шакла. – Санкт-Петербург: Питер, 2019. – 336 с. – ISBN 978-5-4461-0826-8
3. Convolutional neural network // Wikipedia, the free encyclopedia: [сайт]. – URL: https://en.wikipedia.org/wiki/Convolutional_neural_network (дата обращения: 01.12.2022)
4. HybridTech: Свёрточная нейронная сеть, часть 1: структура, топология, функции активации и обучающее множество // Habr: [сайт]. – URL: <https://habr.com/ru/post/348000/> (дата обращения: 01.12.2022)
5. Chollet, F. Deep Learning with Python / F. Chollet. – New York: Manning Publications, 2017. – 384 с.
6. Шакирьянов, Э. Д. Компьютерное зрение на Python. Первые шаги / Э. Д. Шакирьянов. – Москва: Лаборатория знаний, 2021. – 163 с. – ISBN 978-5-00101-944-2.
7. Постолиит, А. В. Основы искусственного интеллекта в примерах на Python. Самоучитель / А. В. Постолиит. – Санкт-Петербург: БХВ-Петербург, 2021. – 448 с. – ISBN 978-5-9775-6765-7
8. Koul, A. Practical Deep Learning for Cloud, Mobile and Edge / A. Koul, S. Ganju, M. Kasam. – Sebastopol: O`Reilly Media, Inc., 2020. – 958 с.
9. Supervisely: unified OS/Platform for computer vision: сайт – Tallinn. – URL: <https://supervise.ly/> (дата обращения: 05.12.2022)
10. Mean Squared Error (MSE) // Introduction to probability, statistics, and random processes: [сайт]. – URL: https://www.probabilitycourse.com/chapter9/9_1_5_mean_squared_error_MSE.php (дата обращения: 07.12.2022)

11. Kingma, D. Adam: A Method for Stochastic Optimization / D. Kingma, J. Ba // 3rd International Conference for Learning Representations (San Diego; 2014). – San Diego, 2015 – 15 с.

12. CCPD (Chinese City Parking Dataset, ECCV) // Github: [сайт]. – URL: <https://github.com/detectRecog/CCPD> (дата обращения: 08.12.2022)