Санкт–Петербургский государственный университет

Ефимов Анатолий Алексеевич

Выпускная квалификационная работа

Методы оптического распознавания символов на основе нейронных сетей

Уровень образования: бакалавриат
Направление 01.03.02 «Прикладная математика и информатика»
Основная образовательная программа СВ.5005.2016 «Прикладная математика, фундаментальная информатика и программирование»

Научный руководитель:
доцент, кафедра технологий программирования
к.т.н. Блеканов Иван Станиславович

Рецензент:
профессор, кафедра информационных систем
д.ф. - м.н. Матросов Александр Васильевич

Санкт-Петербург
2020 г.

Содержание

Введение

Автоматизированное оптическое распознавание символов получило распространение во многом благодаря его применению в области компьютерного зрения, приложений распознавания текста. Подход, принятый для решения проблемы распознавания, был основан на характиристиках сиволов, воспринимаемых людьми, то есть рассматривались геометрические особенности символа.

Позже был применен подход сопоставления шаблонов, который включал сравнение входных символов с заранее определенными шаблонами. Этот метод распознает символы либо как точное совпадение, либо как отсутствие совпадения вообще. Он также не учитывал такие эффекты, как наклоны и вариации стиля, которые не предполагали серьезных изменений формы.

Другой подход, а именно распознавание с использованием коэффициентов корреляции, был основан на перекрестной корреляции входных символов или их преобразований с шаблонами базы данных, чтобы учесть незначительные различия. Он ввел ложное или ошибочное распознавание среди символов, очень похожих по форме, таких как "I" и "J", "B" и "8", "O", "Q" и "0" и т. д.

Нейронные сети позволяют решить эту проблему, так как они могут воспринимать и распознавать символ на основе его топологических особенностей, таких как форма, симметрия, закрытые или открытые области и количество пикселей. Преимущество нейронных сетей заключается в том, что они могут быть обучены на "образцах", а затем использованы для распознавания символов, имеющих сходный (не точный) набор признаков.

Нейронные сети принимаю на вход вектор признаков. Это означает, что каждому обекту должно быть поствлено в соотношение набор чиловых значений. Таким образом, для обучения сети используется векторная база данных, позволяющая ей эффективно распознавать каждый символ на основе его топологических свойств.

Постановка задачи

Целью данной работы является разработка эффективной системы, позволяющей распознавать текст на изображении при помощи нейронных сетей. Для достижения цели поставлены следующие задачи:

- Сделать обзор предметной области
- Выбрать и подготовить набор данных для обучения
- Обучить свёерточную нейронную сеть распознавать английские символы
- Реализовать систему распознавания в виде web-приложения

Глава 1. Нейронные сети

Искусственная нейронная сеть (ИНС) (англ. Artificial neural network (ANN)) — вычислительная модель, вдохновленная биологической нейронной сетью, представляющая собой совокупность искусственных нейронов, взаимодействующих между собой.

Основные принципы работы нейронных сетей были описаны еще в 1943 году Уорреном Мак-Каллоком и Уолтером Питтсом. В 1957 году нейрофизиолог Фрэнк Розенблатт разработал первую нейронную сеть, а в 2010 году большие объемы данных для обучения открыли возможность использовать нейронные сети для машинного обучения.

В настоящее время нейронные сети широко используются в задачах машинного обучения и решают проблемы различной сложности.

Каждый нейрон выполняет относительно простую работу: получает входные данные от соседних нейронов или извне и использует его для вычисления выходного значения, которое является входом в другие нейроны. Еще одна задача — это корректировка весов, используемых для входных значений.

Различают три типа нейронов:

- Входные нейроны – нейроны, которые получают входные значение извне.
- Скрытые нейроны, чьи входные и выходные нейроны остаются внутри нейросети.
- Выходные нейроны

1.1 Математическая модель нейронной сети

Строго говоря, искуссивенный нейрон — это функция $f : \mathbb{R}^n \to \mathbb{R}$, то есть функция преобразующая несколько входных параметров в один выходной [4].

Рассмотрм нейрон с n входами x_i и соответсвующими весами w_{ij}. Взвешенные значения $w_{ij} \cdot x_i$ и смещение θ_j поступают на вход так называемому

сумматору.

$$net_j = \sum_i w_{ij} \cdot x_i + \theta_j = w_j^T \cdot x + \theta_j$$

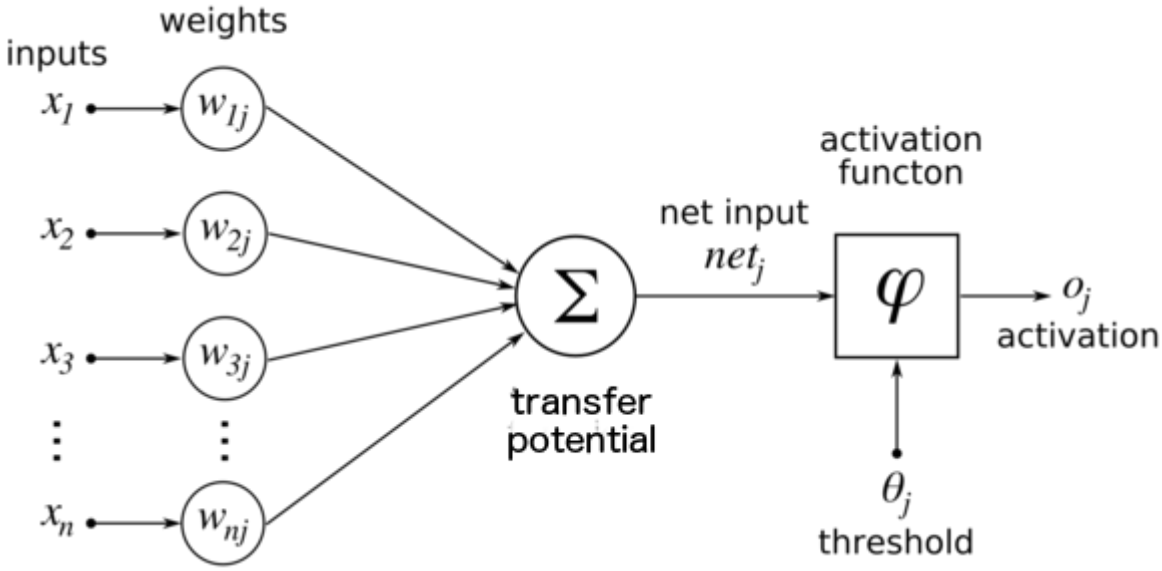


Рис. 1: Модель искуственного нейрона

После этого значение net_j передается в функцию активации $\varphi(net)$, которая преобразует взвешенную сумму в значение, являющееся выходом нейрона.

Некоторые функции активации:

- Пороговая функция

$$\varphi(net) = \begin{cases} 0 & net \geq 0 \\ 1 & net < 0 \end{cases}$$

- Кусочно-линейная функция

$$\varphi(net) = \begin{cases} 1 & net > 1 \\ net & 0 \leq net \leq 1 \\ 0 & net < 0 \end{cases}$$

- Логистическая функция

$$\varphi(net) = \frac{1}{1 + e^{-net}}$$

- Гиперболический тангенс

$$\varphi(net) = \tanh(\frac{1}{2} \cdot net) = \frac{1 - e^{-net}}{1 + e^{-net}}$$

1.2 Обучение нейронной сети

Обучение — это адаптация сети для лучшего решения задачи путем рассмотрения выборочных наблюдений. Обучение подразумевает под собой корректировку весов сети для повышения точности получаемого результата. Это делается путем минимизации наблюдаемых ошибок. Обучение завершается, когда изучение дополнительных наблюдений не приводит к значительному снижению частоты ошибок. Даже после обучения частота ошибок обычно не достигает 0. Практически это делается путем определения функции потерь, которая периодически оценивается во время обучения. Целью процесса обучения является уменьшение общего количества различий между наблюдениями. Большинство моделей обучения можно рассматривать как простое применение теории оптимизации и статистической оценки.

В машинном обучении обратное распространение ошибки (*backpropogation*) — это широко используемый алгоритм обучения нейронных сетей. Во время исполнения алгоритма обратного распостранения ошибки вычисляется градиент функции потерь относительно весов сети для конкретного примера тестовой выборки. Вычисление производится более эффективным образом, в отличие от наивного прямого вычисления градиента по отношению к каждому весу в отдельности. Такая эффективность делает возможным использование градиентных методов для обучения многослойных сетей. Обычно используется градиентный спуск или такие варианты, как стохастический градиентный спуск. Алгоритм обратного распространения работает, вычисляя градиент функции потерь относительно каждого веса по рекуррентному правилу, вычисляя градиент по одному слою за итерацию. Данная идея является примером динамического программирования.

Итак, обзначим вес ребра, соединяющего нейроны i и j, через w_{ij}, o_i — выходное значение нейрона. Также определим некторую функцию потерь

$$E = L(t, y)$$

где t — значение из тестовой выборки, y — значение, вычисленное нейросетью. E опреляет качечство предскзаания нейрости. Часто в качетсве фукнции поетрь выбирают *средний квадрат ошибки*:

$$E = \frac{1}{k} \sum_{k \in outputs} (t_k - o_k)^2$$

Веса модифицируются с помощью стохастического градиентного спуска, то есть мы будем, рассматрвиая примеры тестовой выборки, "спускаться" в сторону противоложную напралению градиента фукнции потерь, тем самым минимизируя её.

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$$

η — коэффицент задаюзий шаг обучения или *learning rate*.

Частную производную будев высиалять следующим образом:

$$\frac{\partial E}{\partial wij} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}}$$

Последнюю частную производную можно упростить:

$$\frac{\partial net_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \left(\sum_{k=1}^{n} w_{kj} o_k \right) = \frac{\partial}{\partial w_{ij}} w_{ij} o_i = o_i$$

Учитывая, что

$$o_j = \varphi(net_j) = \varphi(\sum_{k=1}^{n} w_{ij} o_k)$$

частная прозводная выхода по значение сумматора — это

$$\frac{\partial o_j}{\partial net_j} = \frac{\partial \varphi(net_j)}{\partial net_j}$$

Если нейрон приналдежит выходному слою, то производная функции потерь относительно выхода для него выглядит довольно просто. $o_j = y$, следовательно:

$$\frac{\partial E}{\partial o_j} = \frac{\partial E}{\partial y}$$

Однако если рассмотреть произвольный j-ый нейрон скрытого слоя, то для него все менее очевидно. Пуст L множетсво нейронов, входные значения которых являются взвешенным выходом нейрона j, тогда производная функция потерь для него выглядит следующим образом:

$$\frac{\partial E(o_j)}{\partial o_j} = \frac{\partial E(net_{l_1}, net_{l_2}, ..., net_{l_m})}{\partial o_j}$$

где $l_k \in L$. Вычисляя эту производную, получаем рекуррентное выражение:

$$\frac{\partial E}{\partial o_j} = \sum_{\ell \in L} \left(\frac{\partial E}{\partial net_\ell} \frac{\partial net_\ell}{\partial o_j} \right) = \sum_{\ell \in L} \left(\frac{\partial E}{\partial o_\ell} \frac{\partial o_\ell}{\partial net_\ell} \frac{\partial net_\ell}{\partial o_j} \right) = \sum_{\ell \in L} \left(\frac{\partial E}{\partial o_\ell} \frac{\partial o_\ell}{\partial net_\ell} w_{j\ell} \right)$$

Таким образом, производная относительно o_j может быть рассчитана, если все производные относительно o_ℓ — известны.

Учитывая вышескзанное получаем:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial net_j} o_i$$

$$\frac{\partial E}{\partial w_{ij}} = o_i \delta_j$$

где

$$\delta_j = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial \text{net}_j} = \begin{cases} \frac{\partial L(o_j,t)}{\partial o_j} \frac{d\varphi(\text{n et}_j)}{d\text{net}_j} & j \text{ — нейрон выходного слоя} \\ (\sum_{\ell \in L} w_{j\ell} \delta_\ell) \frac{d\varphi(\text{net}_j)}{d\text{net}_j} & j \text{ — скрытый нейрон} \end{cases}$$

Такой подход имеет некотоыре минусы и ограничения:

- Градиентный спуск не позволяет найти глобальный минимум функции, а только локальный. Также если проиводная на каком-то множетве точек имеет производную равную 0, то алгоритм градиентного поиска прекращает работу и выдает некорректный результат.
- Необходимо знать производную функции активации заранее.

Глава 2. Свёрточные нейронные сети

В глубоком обучении сверточная нейронная сеть (CHH) - это класс нейронных сетей, наиболее часто применяемых для анализа визуальных образов. Они также известны как пространственно-инвариантные искусственные нейронные сети, связи с их архитектурой общих весов. Они эффективно решают задачу распознавания изображений и видео, а также применяются рекомендательных системах, классификации изображений, обработке естественного языка и анализе временных рядах.

2.1 Общая архитектура

Архитектура сверточных нейронных сетей формируется из слоев нескольких типов:

- Свёрточный слой
- Слой субдискретизации
- Полносвязный слой

Ниже рассмотирм каждый из них.

2.1.1 Свёрточный слой

Прежде всего стоит рассмотреть понятие *свёртки*. Свёертка представляет собой операцию над двумя матрицами $A_{n_a \times m_a}$ и $B_{n_b \times m_b}$, результатом которой является матрица $C_{(n_a - n_b) \times (m_a - m_b)}$ Каждый элеменет матрицы равен скалярному произведению матрицы B и некоторой подматрицы A, соответсвующей положению элемента в C. Более формально:

$$C_{ij} = \sum_{u=0}^{n_b - 1} \sum_{b=0}^{m_b - 1} A_{i+u, j+v} B_{u,v}$$

Матрицу B называют *фильтром*, а матрицу A — изображением.

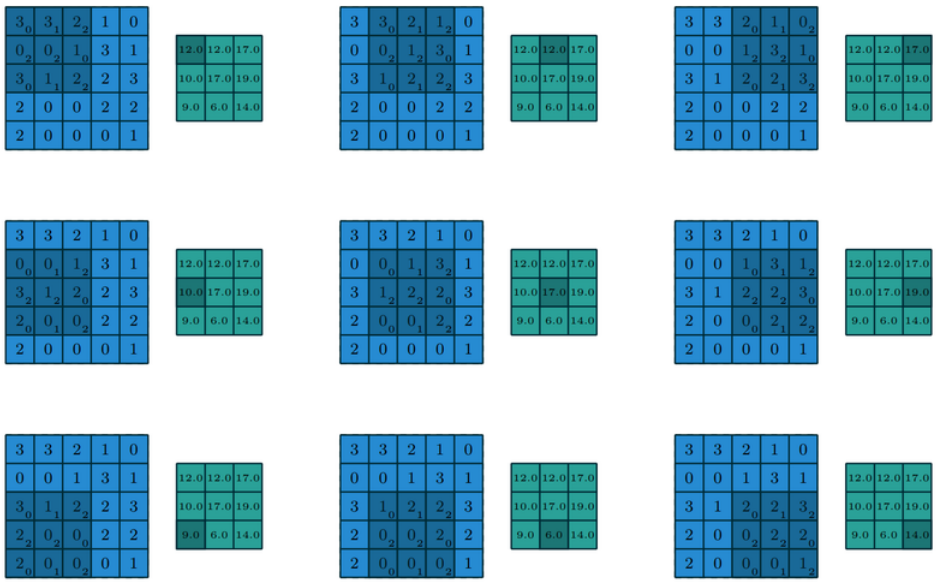


Рис. 2: Пример операции свёртки

На свёрточном слое к входным значениям применяется операция свёртки, а зачения ядра — это обучаемые параметры, то есть веса. Также ко всем элементам свёртки прибавляется константное смещение(англ. *bias*).

Три гиперпараметра контролируют размер выхода свёерточного слоя:

- Глубина(англ. *depth*)
- Сдвиг(англ. *stride*)
- Дополнение(англ. *padding*)

Глубина контролирует размер выхода свёрточного слоя. Пусть на вход поступило изображение размером $w \times h$, и в слое находится k свёрток размерности $n \times m$, тогда размерность выхода будет:

$$n \times (w - n + 1) \times (h - m + 1)$$

Сдвиг отвечает за то, на сколько пикселей мы будем перемещать фильтр. Для любого целого $S > 0$, сдвиг равный S означает, что для каждого выхода фильтр сдвигается на S пикселей. На практике $S > 0$ используется редко.

Иногда бывает удобно дополнить входную матрицу нулями на границе. Размер такого *дополнения* — третий гиперпараметр. Он также позволяет управлять размером выхода слоя.

В итоге, размер выхода может быть вычислен следующим образом:

$$\left\lfloor \frac{W - K + 2P}{S} + 1 \right\rfloor$$

где W — размер входа, K — размер ядра, P — размер доплонения на границе.

2.1.2 Слой субдискретизации

Цель слоя субдискретизации — снизить размерность изображения. Входная матрица делится на блоки, и для каждого из них вычисляется функция. Часто в качестве такой функции использую максимум(англ. *max pooling*). Это позволяет ускорить вычисления.

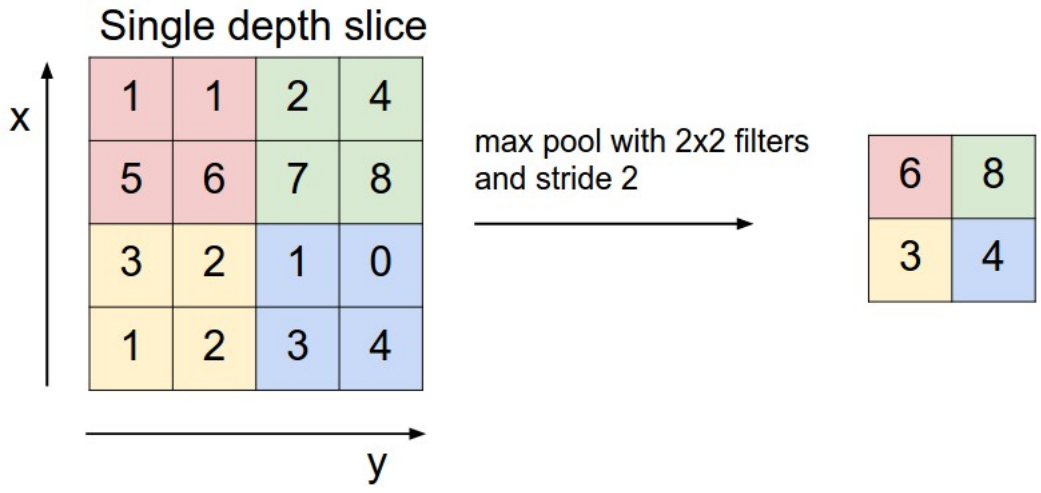


Рис. 3: Пример пулинга

2.1.3 Полносвязный слой

Полносвязные слои соединяют каждый нейрон в одном слое с каждым нейроном в другом слое. Это в принципе то же самое, что и традиционная многослойная нейронная сеть. Сплющенная матрица проходит через полностью связанный слой для классификации изображений.

2.2 Известные архитектуры CHH

2.2.1 LeNet

LeNet - это сверточная нейронная сетевая структура, предложенная Янном Лекуном в 1998 году. LeNet была одной из самых ранних сверточных нейронных сетей и способствовала развитию глубокого обучения. С 1988 года, после многих лет исследований, новаторская работа была названа LeNet5.

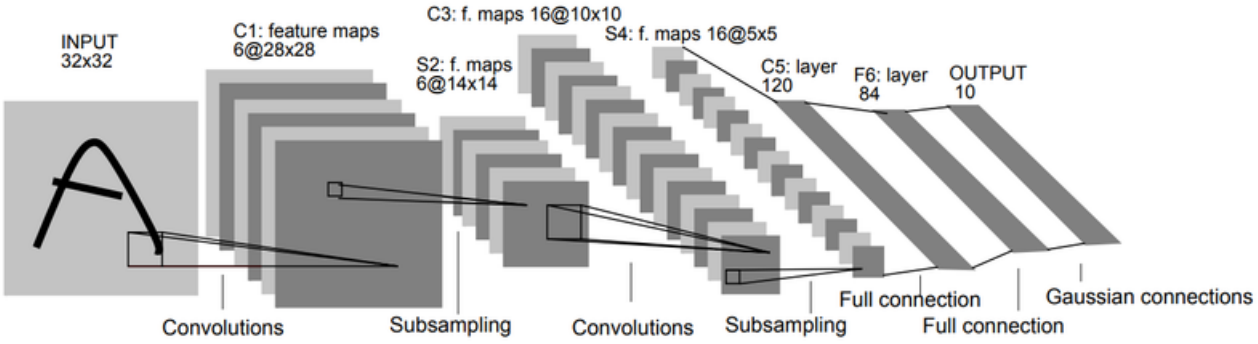


Рис. 4: Архитектура LeNet5

2.2.2 AlexNet

AlexNet — это название сверточной нейронной сети, разработанной Алексом Крижевским и опубликованной совместно с Ильей Суцкевером. 30 сентября 2012 года AlexNet победила в крупномасштабном конкурсе визуального распознавания ImageNet.

AlexNet содержала восемь слоев; первые пять были сверточными слоями, за некоторыми из них следовали MaxPooling-слои , а последние три были полносвязными.

2.2.3 ResNet

ResNet создана для решения проблемы *исчезающего градиента*. Она возникает при ифференцировании по рекуррентному правилу. На глубоких слоях значение градиента очень мало. В связи с этим был разработан *residual block*.

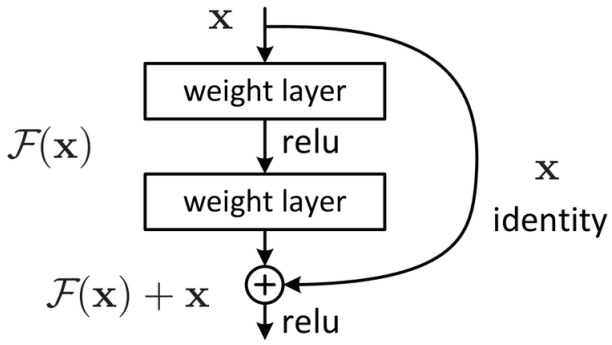


Рис. 5: Residual block

Основная идея состоит в том, чтобы для пары слоёв добавить дополнительную связь, которая проходит мимо этих слоёв.

Глава 3. Препроцессинг

В машинном обучении важным этапом ялвяется предобработка исходных данных. Препроцессинг(предобработка) позвоялет избавиться от нежелательныих признаков и, как следствие, улучшить качество работы вычислительных моделей. Ниже рассмотрены методы, помогающие подготовить текствые изображения для задачи распознавания.

3.1 Бинаризация

На этом этапе изображение приводится к бинарному виду, то есть каждый пиксель изобраения примет значение либо 0, либо 1. Бинаризация частично устраняет шумы, увеличивает контраст и упрощает последующую обработку изображения. Рассмотрим алгоритм приведения изображения к бинарному виду Брэдли-Рота [2], основная идея которого заключается в следующем:

1. Пусть $f(i, j)$ - цвет пискеля с координатами i и j. Посчитаем сумму:

$$S(i, j) = \sum_{i,j} f(i, j)$$

2. Заметим, что её можно посчитать рекуррентно:

$$S(i, j) = \begin{cases} 0 & i < 0 \text{ или } j < 0 \\ f(i, j) + S(i-1, j) + S(i, j-1) - S(i, j) & \text{иначе} \end{cases}$$

3. Разделим изображение на d частей со сторонам длиной h/d и w/d:

4. Рассмотрим часть с координамти левого верхнего угла (x_1, y_1) и координатами нижнего правого угла (x_2, y_2). Каждому пикселю данной части присвоим значение в соответствии с средним значеним среди

рассматриваемой части:

$$f(i,j) \coloneqq \begin{cases} 0 & f(i,j) < m \\ 1 & \text{иначе} \end{cases}$$

, где $m \coloneqq \sum_{i=x_1}^{x_2} \sum_{j=y_1}^{y_2} f(i,j)/(x_2 - x_1 + 1) \cdot (y_2 - y_1 + 1)$. Стоит отметить, что m мы можем посчитать за $O(1)$ операций с помошью уже посчитанных префиксных сумм:

$$\sum_{i=x_1}^{x_2} \sum_{j=y_1}^{y_2} f(i,j) = S(x_2,y_2) - S(x_2,y_1-1) - S(x_1-1,y_2) + S(x_1-1,y_1-1)$$

Таким образом получено изображение, состоящее из черных пикселей на белом фоне.

3.2 Устранение угла наклона

В общем случае текст на исходном изображении может иметь некоторый угол наклона, что затрудняет извлечение признаков и самих символов. Для его устранения рассмотрим метод, основанный на преобразовании Хафа[4]. Нашей целью будет обнаружить параллельные прямые на изображении, проходящие через наибольшее количество точек, это позволит определить уго наколна текста.

Прямая может быть задана уравнением $y = kx + b$ и, соответственно, представлена в виде точки с координамти (k, b) в пространстве параметров. Но у такого представления есть недостаток, а именно прямые, параллельные оси ординат, соотвествуют точке с бесконечным значением k. Данное отображение прямой плоскости в точку просртанства параметров создает проблемы для реализации алгоритма. По этим причинам удобнее предстваить прямую с мощью параметров ρ и θ, где ρ – длина радиус вектора точки прямой, ближайшей к началу координат, то есть нормали, проведенной из начала координат, а θ – угол между этим ветором и осью абцисс(проиллюстрировано на рисунке

6). Переписывая в терминах ρ и θ, получаем:

$$y = -\frac{\cos\theta}{\sin\theta} \cdot x + \frac{\rho}{\sin\theta}$$

или

$$\rho = x \cdot \cos\theta + y \cdot \sin\theta$$

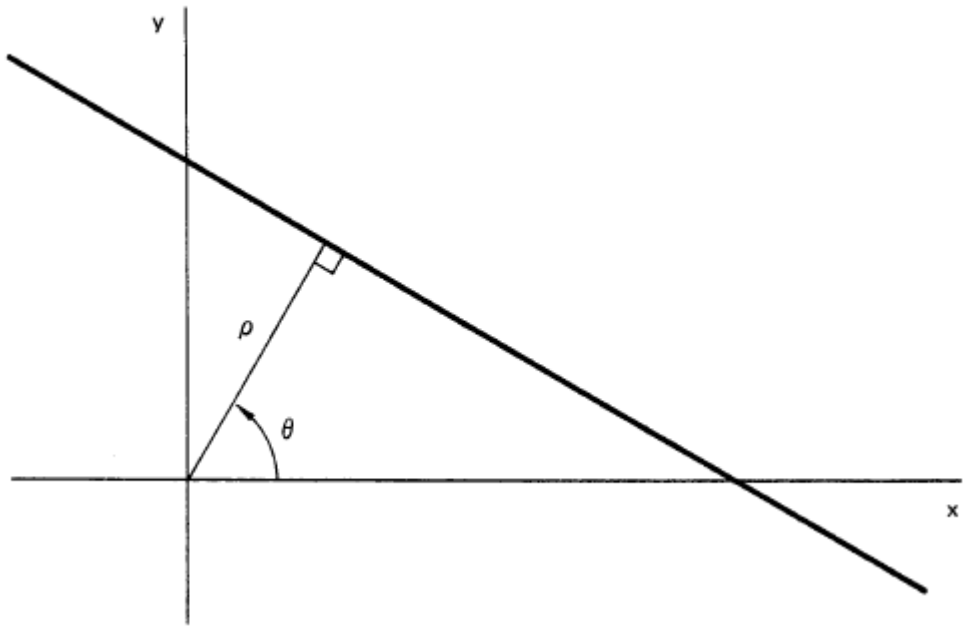


Рис. 6: Нормально уравнение прямой

Если огранить θ интервалом $[0, \pi)$, то каждая прямая плоскости однозначно соответсвует точке $(\theta',\ \rho')$ пространства параметров. Таким образом мы избалвяемся от параметров с бесконечным значением.

Теперь предположим что у нас есть множество точек плоскости

$$\{(x_1, y_1), ..., (x_n, y_n)\}$$

Отобразим каждую точку в множетсво прямых в пространстве парметров, прохоядщих через этут точку.

$$\rho = x_i \cos\theta + y_i \sin\theta$$

Легко покзать, что точки (x_i, y_i), лежащие на одной прямой, имеют в множе-

ствах прямых, проходящих через них, одну общую. Так задача нахождения коллинеарных точек сводится к нахождению параллельных прямых. Также несложно установить, что точки, лежащие на одной кривой в простанстве параметров, сооотвествую прямым проходящим через одну и ту же тчоку на изображении.

$$\rho = x' \cos\theta + y' \sin\theta$$

Задача нахождения колинераных точек изображения может быть решена путем нахождения прямых проходящих черех все пары точек. Но этот подоход займет $O(n^2)$ операций, квадратичный рост сложности алгоритма в зависимости от количества точек на практике приводит к большему времени работы, поэтому воспользуемся вышеизложенными замечениями для более опитмального подхода.

Представим пространство параметров в виде таблицы. Индексы строк ρ будут принадлежать отрезку $[-R,\ R]$, где R - максимум из ширины и высоты изображения. Индексами столбцов таблицы будет параметр θ, заданный с некоторым шагом d. Уменьшение шага приводит к более тоыным результатам алгоритма, но увеличивает время работы и размер используемой памяти. В ходе алгоритма пербираются все точки изобраения, для каждой точки, в свою очередь, рассматриваются все прямые проходящие через них. Соответсвующее значение таблицы инкрементируется(изначально ячейки таблицы имеют значение 0). Подсчитанные значения таблицы позволят нам параллльные прямые ихображения, которые на самом деле являются строками текста. Искомый угол наклона будет соответсовать стобцу котором наибольшее количество элементов находится в заданной окретсности максимума. Тем самым мы получили желаемый результат за $O(\frac{180}{d}Rn) \sim O(n)$ операций

Осталось обратно преобразоватm угол относительно оси абцисс и повернуть изображение на найденный угол:

$$\theta := \begin{cases} \theta + 90^\circ & 0^\circ \leq \theta < 90^\circ \\ \theta - 90^\circ & 90^\circ \leq \theta < 180^\circ \end{cases}$$

3.3 Сегментация

После того, как угол наклона устранен, и изображение оринетировано к наблюдателю правильным образом, перед нами встает задача сегментации изображения. Этот процесс можно условно разделить на три этапа:

1. Разбиение исходного изображения на строки.
2. Выделение слов из каждой строки.
3. Извлечение символов из слова.

Далее рассматривается изображение, полученное после бинаризации исходного. Введем некторые обозначения:

$$I = \{p_{ij}\}$$

$$p_{ij} \in \{0, 1\}$$
$$i = \overline{1, n}, \quad j = \overline{1, m}$$

где n — высота изображения, а m — ширина.

Задача выделения строк сводиться к нахождению верхних и нижних границ строк текста исходного изображения. Алгоритм сегментации строк основывается на том, что средняя яркость в изображениях межстрочных промежутках существенно ниже средней яркости в изображениях текстовых строк. Сначала для всех строк исходного изображения находим их средние значения яркости:

$$a_i = \frac{1}{m} \cdot \sum_{j=1}^{m} p_{ij}$$

Затем использую уже посчитанные значения, определим среднее значение яркости для всего изображения:

$$a = \frac{1}{n} \cdot \sum_{i=1}^{n} a_i$$

Яркость верхних и нижинх границ может быть выражена через яркость

всего изображения:

$$s^t = s^b = k \cdot a$$

Основная идея алгоритма заключается в просмотре массива средних значений $\{a_1, ..., a_n\}$ по строкам и нахождении пар индексов (t_i, b_i), являющихся индексами верхней и нижней границ строки i.

Для детектирования верхней границы строки определим следующие условия:

- Значение текущей пиксельной строки не менее s_t
- Значения двух предшествующих пиксельных строк меньше s_t
- Значение трех строк после рассаматриваемой больше s_b

С другой стороные условиями для нижней границы будут:

- На предыдущих итерациях была обнаружена верхняя граница.
- Значение трех последующих строк ниже s_b

Как резлуьтат мы получаем искомое множетсво пар индексов, но это может обрезать символы, которы в силу своего написания, выходят за верхнюю или нижнюю границу, следовательно нужно увеличить полученные высоты.

Для сегментации слов педпосчитаем средние значения также как и для строк, но только по пиксельным колонкам и для изображеия строки.

$$c_j = \frac{1}{h} \cdot \sum_{i=1}^{h} p_{ij}$$

$$c = \frac{1}{m} \cdot \sum_{i=1}^{m} c_i$$

$$s^l = s^r = k \cdot c$$

Определим условия для левой границы слова:

- Значение текущего и последующего пиксельного столбца болше левой границы яркости s_l.

- Значение предыдущего пиксельного столбца меньше этой границы.

Аналогично для правой границы:

- было зафиксировано начало слова
- Яркость текущего и четырех последующих пиксельных столбцов ниже границы яркости межсловного интервала s_r
- Яркость двух предыдущих пиксельных столбцов выше этой границы

Глава 4. Реализация

4.1 Обучение нейронной сети

Для обучения и тестирования нейронной сети была выбрана база данных EMNIST(Modified National Institute of Standards and Technology extended) [6], предложенная Национальным институтом стандартов и технологий США для калибрации и сопоставления методов распознавания изображений с помощью машинного обучения в первую очередь на основе нейронных сетей. Датасет содержит в себе 145600 изображений рукописных букв английского алфавита размером 28×28 пикселей.

Для релизации нейронной сети было отдано предпочтение библеотеке PyTorch[7]. PyTorch — библиотека машинного обучения для языка Python с открытым исходным кодом, которая позволяет гибко строить граф вычисления. Одим из важных преимуществ PyTorch является то, что она позволяет нативно интегрировать вычисления на GPU, в связи с чем процесс обучения нейросети занимает значительно меньше времени.

Для нашей задачи выбор был остановлен на следующей архитектуре свёрточной нейронной сети.

Таблица 1: Арихитектура нейронной сети

Layer	Size	Kernel size	Activation
Image	28×28	-	-
Convolution	24×24	5×5	tanh
Average Pooling	14×14	2×2	tanh
Convolution	10×10	5×5	tanh
Average Pooling	5×5	2×2	tanh
Convolution	1×1	5×5	tanh
FC	84	-	tanh
FC	26	-	softmax

Часть исходного кода программы со структрой нейросети:

```
class Net(torch.nn.Module):
    def __init__(self):
        super(Net, self).__init__()

        self.convolution1 = torch.nn.Conv2d(
            in_channels=1, out_channels=6, kernel_size=5, padding=2)
        self.activation1  = torch.nn.Tanh()
        self.pooling1 = torch.nn.AvgPool2d(kernel_size=2, stride=2)

        self.convolution2 = torch.nn.Conv2d(
            in_channels=6, out_channels=16, kernel_size=5, padding=0)
        self.activation2  = torch.nn.Tanh()
        self.pooling2 = torch.nn.AvgPool2d(kernel_size=2, stride=2)

        self.fc1   = torch.nn.Linear(5 * 5 * 16, 120)
        self.activation3  = torch.nn.Tanh()

        self.fc2   = torch.nn.Linear(120, 84)
        self.activation4  = torch.nn.Tanh()

        self.fc3   = torch.nn.Linear(84, 10)

    def forward(self, x):

        x = self.convolution1(x)
        x = self.activation1(x)
        x = self.pooling1(x)

        x = self.convolution2(x)
        x = self.activation2(x)
        x = self.pooling2(x)

        x = x.view(x.size(0), x.size(1) * x.size(2) * x.size(3))

        x = self.fc1(x)
        x = self.activation3(x)
        x = self.fc2(x)
        x = self.activation4(x)
        x = self.fc3(x)

        return x

net = Net()

batch_size = 100
```

```
46
47 test_accuracy_values = []
48 test_loss_values = []
49
50 X_test = X_test.to(device)
51 y_test = y_test.to(device)
52
53 for epoch in range(10000):
54     perm = np.random.permutation(len(X_train))
55     for i in range(0, len(X_train), batch_size):
56         optimizer.zero_grad()
57
58         current_batch = perm[i:i+batch_size]
59
60         X_batch = X_train[current_batch].to(device)
61         y_batch = y_train[current_batch].to(device)
62
63         preds = net.forward(X_batch)
64
65         loss_val = loss(preds, y_batch)
66         loss_val.backward()
67
68         optimizer.step()
69
70     test_preds = net.forward(X_test)
71     test_loss_values.append(loss(test_preds, y_test).data.cpu())
72
73     accuracy = (test_preds.argmax(dim=1) == y_test).float().mean().data.
    cpu()
74     test_accuracy_values.append(accuracy)
75
76     print(accuracy)
```

Нейронная сесть выдет точность 96% по прошествию 100 эпох. При данных гиперпараметрах увелечение количества эпох ведет к переобучению. Графики точности и ошибки в зависимости от количества эпох соответственно:

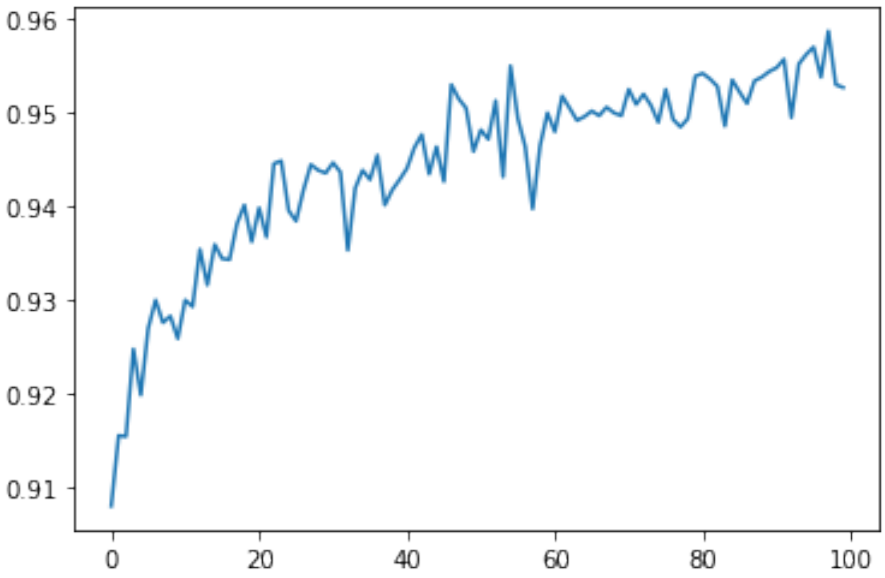


Рис. 7: График точности предсказания

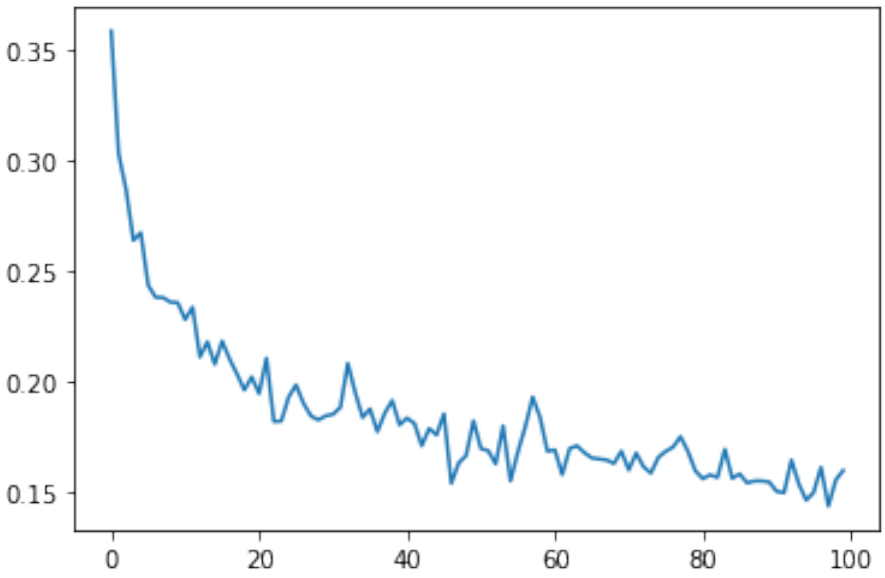


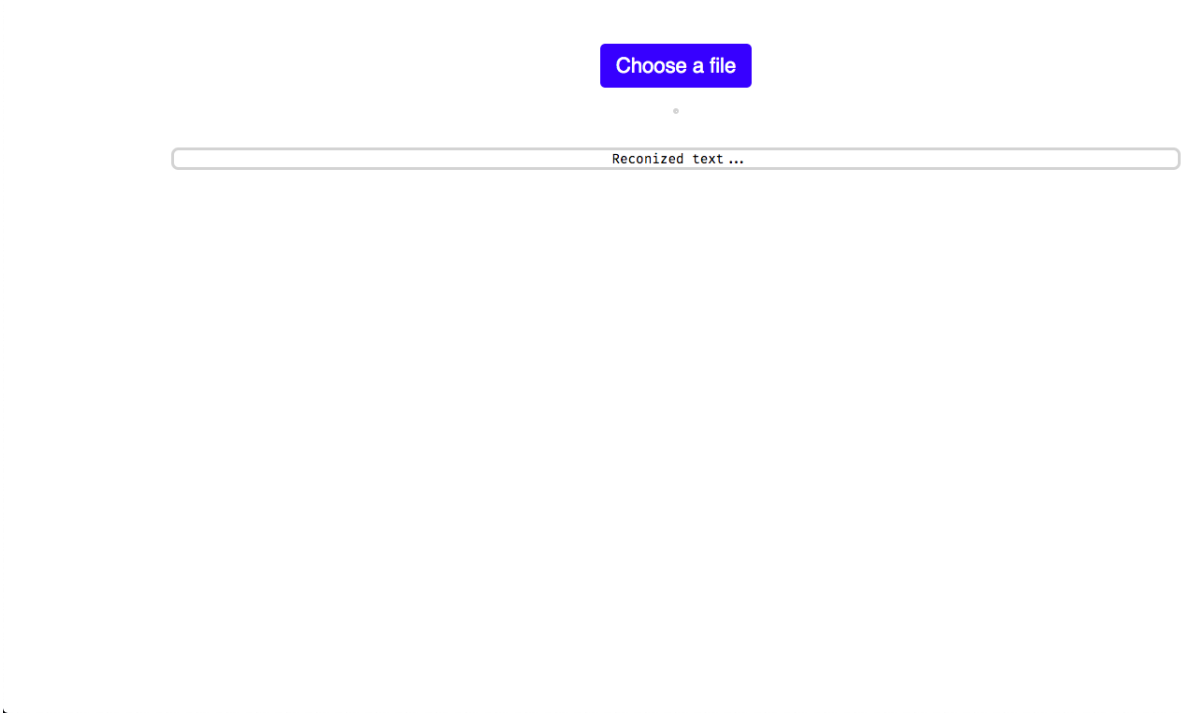
Рис. 8: График ошибки предсказания

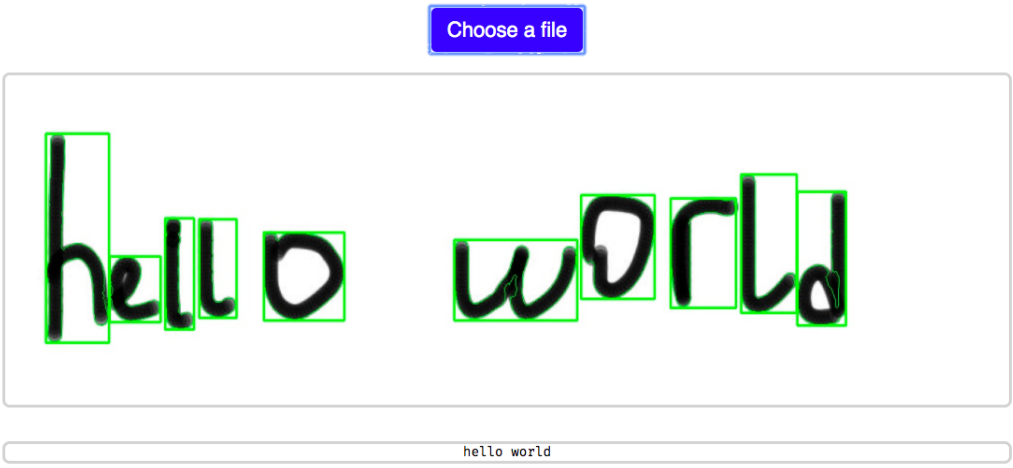
4.2 Используемые технологии

Данную систему было решено реализовать в виде web-приложения, используюя следующие технологии:

- Python —– высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода[8].
- Flask — микрофреймворк для web-приложений написанный на Python [9]
- JavaScript — язык, являющийся реализацией стандарта ECMAScript, более часто используемый для интерактивизации web-страниц
- Hypertext Markup Language (HTML) — стандартный язык разметки, используемый браузерами.

Интефейс представляет из себя поле для загрузки изображений. Загруженное изображение отправляется на серверт, сегментируется, и символы классифицируются с помощью нейронной сети.




Choose a file
hello world
hello world

Заключение

В данной выпускной квалификационной работе были достигнуты следующие результаты:

- Сделан обзор предметной области
- Выбрана и подготовлена база данных для обучения
- Реализована и обучена нейронная сеть для распознавания рукописных английских букв.
- Разработана системы распознавания рукописного английского текста в формате web-приложения. Сслыка на репозиторий с исходным кодом: https://github.com/anatolyefimov/text-recognition

Для свёрточной нейросети была выбрана архитектура, основанная на LeNet5, не смотря на свою простоту, она показывается хорошие результаты в решенни задач более широкго класса, чем распознавание цифр.

Список литературы

[1] Vivek Shrivastava, Navdeep Sharma. Artificial Neural Network Based Optical Character Recognition // Signal & Image Processing : An International Journal, 2012. Vol.3, No 56. P. 73-80

[2] Derek Bradley, Gerhard Roth. Adaptive thresholding using the integral image // ACM J. Graph. Tools, 2007. P. 13-21

[3] O. Boudraa, W. K. Hidouci and D. Michelucci. An improved skew angle detection and correction technique for historical scanned documents using morphological skeleton and progressive probabilistic hough transform // 5th International Conference on Electrical Engineering - Boumerdes (ICEE-B), Boumerdes, 2017. P. 1-6

[4] Richard O. Duda, Peter E. Hart. Use of the Hough Transformation to Detect Lines and Curves in Pictures // Association for Computing Machinery, 1972. Vol.15, No 1. P. 11-15

[5] Kröse B., Smagt P. An Introduction to Neural Networks. (8th ed.) // University of Amsterdam Press, University of Amsterdam, 1962.

[6] Cohen G., Afshar S., Tapson J., van Schaik A. // EMNIST: an extension of MNIST to handwritten letters, 2017

[7] Документация PyTorch. https://pytorch.org/docs/stable/index.html

[8] Документация Python. https://docs.python.org/3/

[9] Документация Flask. https://flask.palletsprojects.com/en/1.1.x/