ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ»
(Н И У « Б е л Г У »)

ИНСТИТУТ ИНЖЕНЕРНЫХ ТЕХНОЛОГИЙ И ЕСТЕСТВЕННЫХ НАУК

КАФЕДРА МАТЕМАТИЧЕСКОГО И
ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ

ДЕТЕКТИРОВАНИЕ И РАСПОЗНАВАНИЕ ТЕКСТА СРЕДСТВАМИ СВЕРТОЧНЫХ НЕЙРОСЕТЕЙ

Выпускная квалификационная работа
обучающегося по направлению подготовки 02.04.01 Математика и Компьютерные науки
очной формы обучения, группы 07001631
Кучеренко Павла Александровича

Научный руководитель
к.т.н., доцент
Бурданова Е.В.

Рецензент
к.т.н., ст. преподаватель
Лифиренко М.В.

БЕЛГОРОД 2018

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

На сегодняшний день существует ряд направлений науки и техники, которые в значительной степени ориентированы на развитие систем, анализирующих информацию, представленную в виде изображений. Задача обработки и распознавания изображений относится к разряду трудно формализуемых задач, и является одной из наиболее важных на сегодняшний день.

Для решения данной задачи в последнее время стали активно разрабатываются различные архитектуры нейронных сетей, которые дают значительно более точные результаты распознавания по сравнению с другими алгоритмами.

В рамках данной работы была выбрана архитектура сверточных нейронных сетей, как одна из наиболее подходящих для решения поставленной задачи. Данная архитектура нацелена на эффективное распознавание изображений, она хорошо зарекомендовала себя в решении подобных задач. Так же эта архитектура нейронных сетей входит в состав технологий глубокого обучения.

Целью данной работы является разработка алгоритма и программного обеспечения для сегментации и распознавания машиночитаемого рукописного и печатного текста на основе сверточной нейронной сети.

Для решения поставленной задачи необходимо решить следующие задачи:

- Исследовать существующие архитектуры нейросетей для классификации изображений;
- Исследовать известные методы сегментации изображений;
- Выбрать оптимальную архитектуру сверточной нейросети для решения поставленных задач;
- Найти экспериментальным путем оптимальные параметры для сети;

- Проектирование и разработка программного продукта для решения задачи выделения и классификации печатных и рукописных символов.

Объектом исследования в данной работе выступают алгоритмы сегментации и распознавания объектов на изображениях, применяемые для решения задачи выделения и распознавания рукописных и печатных символов.

Предметом исследования является задача разработки программного продукта, реализующего алгоритм сегментации и распознавания на основе сверточной нейронной сети для решения задачи классификации рукописных и печатных символов.

ГЛАВА 1. ОБЗОР АЛГОРИТМОВ РАСПОЗНАВАНИЯ И СЕГМЕНТАЦИИ

Распознавание рукописных и печатных текстов на сегодняшний день является актуальным направлением как научной, так и технической деятельности. и широко освещается в специализированной литературе. В литературных источниках [1, 3, 5] затронуты основные моменты проектирования нейронных сетей для различных задач науки и техники.

В литературных источниках [10, 13, 25] приводится обоснование актуальности использования сверточных нейронных сетей для распознавания текста на изображениях. Ключевым моментом в разработке нейронных сетей является выбор алгоритма обучения. В [4, 5, 6] рассматривается метод обучения сетей с помощью обратного распространения ошибки и способы его организации по отношению к разным архитектурам нейронных сетей.

С целью повышения эффективности последующей сегментации, в источниках [4, 9, 11] предоставлена информация об основных методах предварительной обработки изображений.

Обзор некоторых алгоритмов сегментации для различных типов изображений представлены в источниках [14, 17]. Также в источниках [15, 18, 19] описываются способы их реализации.

1.1 Искусственные нейронные сети

Достаточно давно учеными предпринимаются попытки воспроизвести способность биологической нервной системы обучаться и исправлять ошибки, что привело к созданию искусственных нейронных сетей. Искусственные нейронные сети – математические модели, а также их аппаратные и программные воплощения строятся по принципу организации и функционирования биологических нейронных сетей.

На сегодняшний день нейросети успешно применяются для решения различного рода задач, таких как: кластеризация, адаптивное управление, распознавание речи, машинное зрение и ряда других не менее важных направлений.

Несмотря на наличие различий в архитектурах нейронных сетей, все они обладают некоторыми общими чертами. Во-первых, основу каждой нейронной сети составляют относительно простые и однотипные элементы (формальные нейроны), имитирующие работу нейронов головного мозга.

Во-вторых, общей чертой для всех нейронных сетей является возможность параллельной обработки сигналов, достигающейся за счет объединения нейронов в слои. На рисунке 1 представлена структура связей однослойной полносвязной нейронной сети, где каждый нейрон из предыдущего слоя связан с каждым нейроном из следующего слоя.

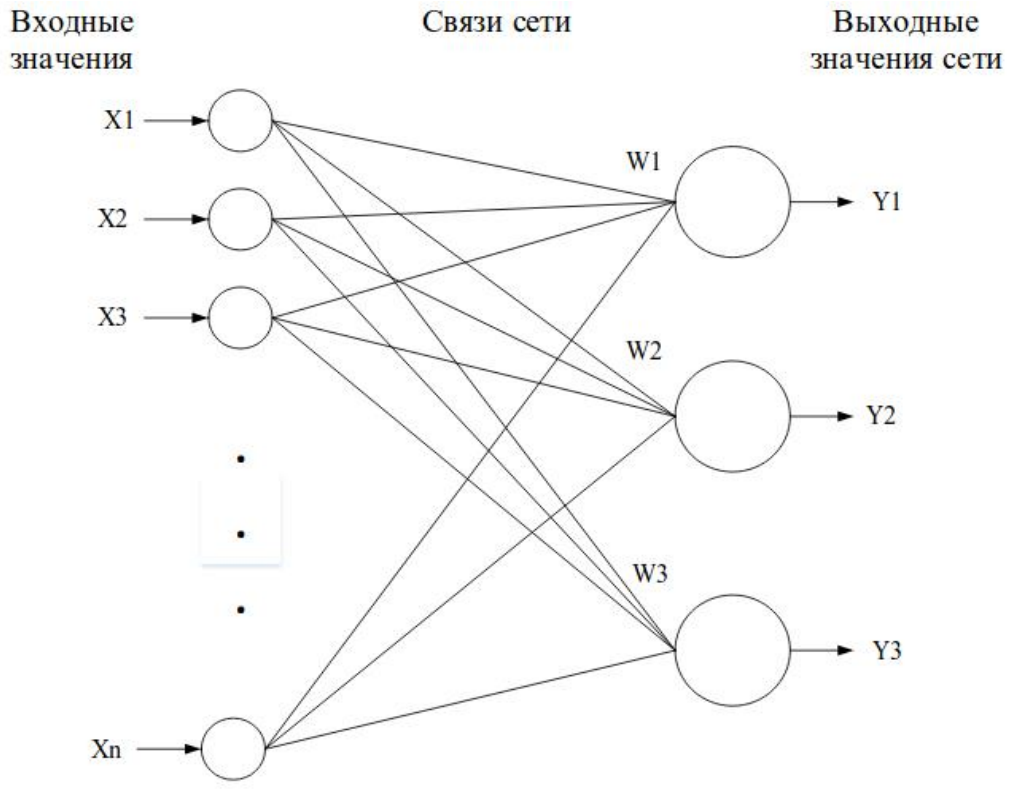

Рис. 1.1. Структура связей полносвязной нейронной сети

В теории, количество слоев и нейронов в слое может быть выбрано произвольным образом, но на практике оно ограничивается ресурсами компьютера, на котором будут производиться вычисления и временем, за которое сеть должна производить необходимые вычисления. Однако, чем сложнее нейронная сеть, тем более масштабные задачи она может решать.

Процесс функционирования нейронной сети зависит от величин синаптических связей (весов сети), поэтому, после определения структуры нейронной сети для решения какой-либо задачи необходимо найти оптимальные значения всех настраиваемых параметров. Этот процесс называется обучением нейронной сети. От того, насколько точно будут подобранны значения параметров (весов), зависит способность сети решать поставленные перед ней задачи.

1.1.1 Модель формального нейрона

В основе любой нейронной сети лежит модель формального нейрона (рис. 1.2), который является основной функциональной единицей нейронной сети.

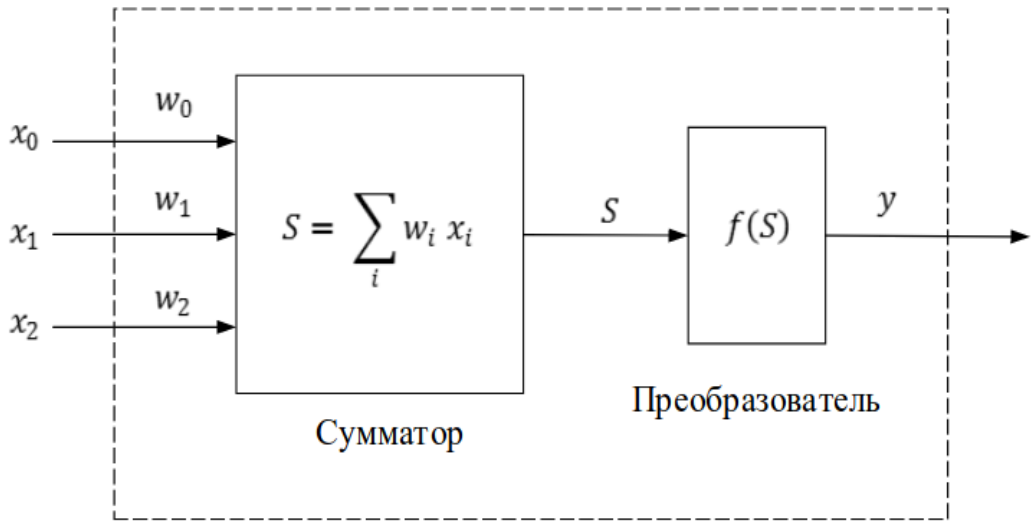

Рис. 1.2. Модель формального нейрона

В этой модели нейрон получает входные сигналы (*x0 … xn*), которые проходят через связи-синапсы, вес которых (*w0 … wn* соответственно) эмулируя различную пропускную способность синапсов в естественных нейронах. После прохождения входных сигналов по синапсам они обрабатываются сумматором, задачей которого является получение линейной комбинации всех входных сигналов, в свою очередь полученная линейная комбинация служит аргументом для функции-преобразователя. Именно эта функция определяет значение выходного сигнала нейрона, который затем посылается на единственный выход нейрона – аксон. Таким образом искусственные нейроны объединяют в сети путем соединения выходов одних нейронов с входами других.

Следовательно, формальный нейрон состоит из элементов 3 типов [4]:

- синапсов (весов, также иногда называемых умножителями), которые характеризуют силу связи между двумя нейронами;
- сумматор, который выполняет сложение входных сигналов нейрона, перед этим умноженных на соответствующие веса связей;
- преобразователь, который реализует функцию одного аргумента. Эта функция называется функцией активации или передаточной функцией нейрона.

Исходя из полученного описания, можно представить математическую модель нейрона следующим образом:

$$y = f(S), \tag{1.1}$$

$$S = \sum_{i=1}^{n} w_i x_i + b, \tag{1.2}$$

где *x* – элементы вектора входных сигналов, *w* – значения весов связей нейрона, *b* – смещение нейрона, а *y* – выходной сигнал нейрона.

1.1.2 Функция активации

Функция активации, представленная в формуле (1) как *f*(*S*), определяет выходной сигнал нейрона в зависимости от взвешенной суммы входов *S*.

Ниже приведены основные виды функций активации [1, 4]:

1. ***Пороговая функция активации.*** Эта функция описывается следующим образом:

$$f(S) = \begin{cases} 0, \text{при } S < 0 \\ 1, \text{при } S \geq 0 \end{cases} \quad (1.3)$$

2. ***Линейная функция.*** Линейная функция описывается следующим образом:

$$f(S) = S \quad (1.4)$$

3. ***Кусочно-линейная функция.*** Кусочно-линейная функция описывается следующим выражением:

$$f(S) = \begin{cases} 1, S \geq \frac{1}{2} \\ |S|, +\frac{1}{2} > S > -\frac{1}{2} \\ 0, S \leq -\frac{1}{2} \end{cases}. \quad (1.5)$$

4. ***Сигмоидальная логистическая функция.***

Сигмоидальная функция является одной из самых распространенных функций, применяемых в искусственных нейронных сетях. Примером

сигмоидальной функции может служить логсигмойдная функция, описываемая следующим выражением:

$$f(S) = \frac{1}{1+e^{-aS}}\ , \tag{1.6}$$

где, а – параметр наклона, изменение которого позволит построить функцию с различной крутизной наклона. Область значений сигмоидальной функции лежит в диапазоне от 0 до 1.

Однако иногда требуется функции активации, имеющая область значений от -1 до +1, в этом случаи функция активации должна быть симметрично относительно начала координат. Тогда пороговую функцию можно определить следующим образом:

$$f(S) = \begin{cases} 1,\ S > 0 \\ 0,\ S = 0 \\ -1,\ S < 0 \end{cases}\ . \tag{1.7}$$

Данная функция называется *сигнум*, в непрерывном дифференцируемом виде она будет иметь форму *гиперболического тангенса*:

$$f(S) = \frac{e^{(aS)} - e^{(-aS)}}{e^{(aS)} + e^{(-aS)}} = Atanh(aS), \tag{1.8}$$

где f (S) – искомое значение элемента, S – взвешенная сумма входов, a – параметры функции.

Ниже описаны используемые в данной работе функции активации ReLU и Softmax, на данный момент они имеют лучшие результаты при обучении нейронных сетей.

5. Выпрямленная линейная функция активации (rectified linear unit, ReLU). Известно, что нейронные сети способны приблизить сколь угодно сложную функцию, если в них достаточно слоев, и функция активации является

нелинейной. Функции активации типа сигмоидной или тангенциальной являются нелинейными, но приводят к проблемам с затуханием или увеличением градиентов в процессе обучения. Однако можно использовать и гораздо более простой вариант – выпрямленную линейную функцию активации. Функция ReLU является выпрямленной линейной функцией и на данным момент считается гораздо более простым и эффективным с точки зрения вычислительной сложности вариантом передаточной функции, она является одним из последних успехов в области методов настройки глубоких нейронных сетей:

$$f(S) = max(0, S) = \begin{cases} 0, \text{при} S < 0 \\ S, \text{при} S \geq 0 \end{cases}. \quad (1.9)$$

На рисунке 1.3 представлен график функции ReLU и ее первой производной.

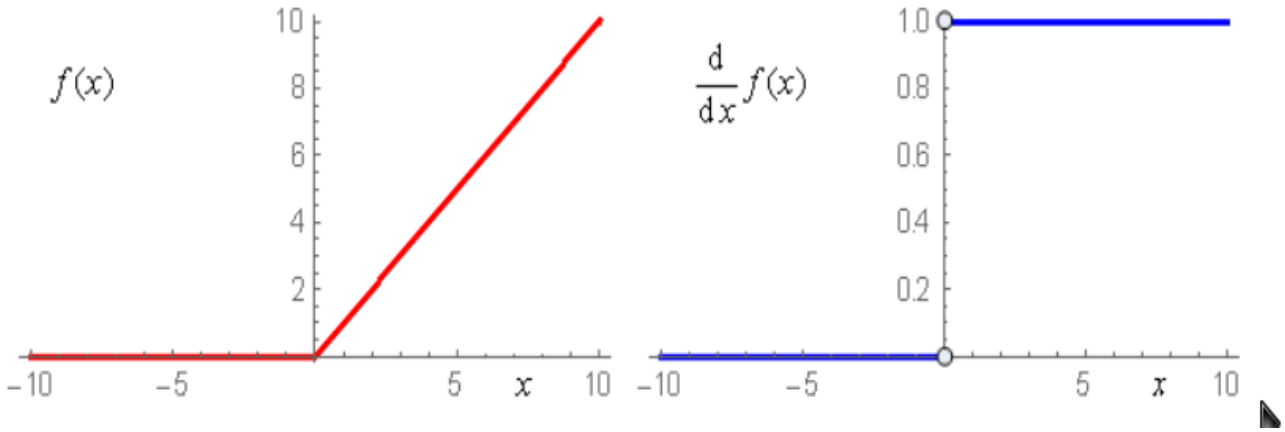

Рис. 1.3. График функции ReLU.

Ее производная равна либо единице, либо нулю, и поэтому не может произойти разрастания или затухания градиентов. Более того, использование данной функции приводит к прореживанию весов.

На сегодняшний день существует семейство различных модификаций ReLU, решающих проблемы надёжности этой передаточной функции при

прохождении через нейрон больших градиентов: Leaky ReLU, Parametric ReLU, Randomized ReLU.

6. Функция активации Softmax. Эта функция активации разработана, чтобы превратить любой вектор с реальными значениями в вектор вероятностей и определяется для *i*-ого нейрона следующим образом:

$$Z_i = \frac{exp(y_i)}{\sum_{j=1}^{n} exp(y_j)}, \tag{1.10}$$

где z_i – искомое значение выхода *i*-ого нейрона, y_i – исходное значение выхода *i*-ого нейрона.

Из формулы (1.3) видно, что значение каждого выходного нейрона зависит от суммы всех остальных нейронов. Преимущество данной функции заключается в том, что частная производная *i*-ого нейрона равна:

$$\frac{\partial z_i}{\partial y_i} = Z_i(1 - Z_i), \tag{1.11}$$

На рисунке 1.4 представлен график функции *Softmax*.

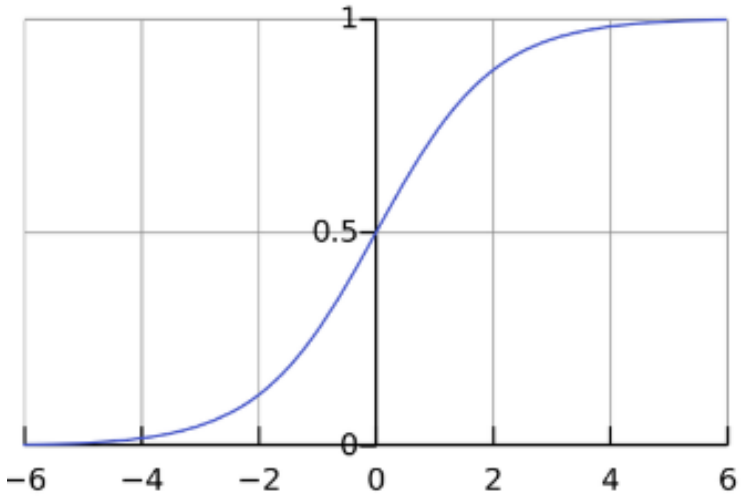

Рис. 1.4. График функции *Softmax*

1.1.3 Функция потерь

В данном разделе будет рассмотрена одна из наиболее часто встречающихся функций потерь, используемая в данной работе – перекрестная энтропия.

Так как в качестве функции активации в выходном слое будет использоваться функция активации softmax, которая преобразует любой входной вектор в вектор вероятностей, то для сравнения двух вероятностных распределений необходимо выбрать корректную меру. В качестве такой меры будет использоваться перекрестная энтропия [10]:

$$C = -\sum_{j=1}^{n} t_j log(y_j) \tag{1.12}$$

где t_j – требуемый выход для текущего обучающего примера, y_j - реальный выход нейронной сети.

1.1.4 Алгоритм обучения нейронной сети

Наиболее распространенным алгоритмом обучения нейронных сетей является метод градиентного спуска или как его еще называют метод обратного распространения ошибки, и его модификации. Данный алгоритм относится к методам обучения с учителем [4, 5, 6].

В процессе обучения нейронной сети с применением алгоритма обратного распространения ошибки, на входы сети подаются множества примеров из обучающей выборки. Один цикл предъявления всего набора примеров из обучающей выборки называют эпохой. Процесс обучения происходит от эпохи к эпохе, пока веса и уровни порога (смешения) сети не

стабилизируются, а ошибка сети на всем обучающем множестве не сойдется к некоторому заданному минимальному значению.

Алгоритм обратного распространения ошибки может быть реализован тремя способами:

Последовательный режим. Последовательный режим обучения также иногда называют стохастическим градиентным спуском. В этом режиме изменение весов связей происходит после подачи каждого примера из обучающей выборки.

Пакетный режим. В пакетном режиме обучения корректировка весов связей происходит после подачи на вход сети всех обучающих примеров одной эпохи обучения.

Mini-batch. Между этими двумя видами метода обратного распространения ошибки (градиентного спуска) существует компромисс, называемый иногда «mini-batch». В этом случае корректировка синоптических весов сети происходит после небольшого количества обучающих образцов.

С точки зрения производительности, последовательный режим обучения является более предпочтительным, чем пакетный, так как для хранения каждой синоптической связи требуется меньший объем внутренней памяти. Помимо этого, предъявление обучающих примеров в случайном порядке в процессе обучения для последовательно режима, делает поиск в пространстве весов стохастическим. Таким образом уменьшая возможность остановки алгоритма в точке какого-либо локального минимума.

Алгоритм работы метода обратного распространения ошибки для пакетного режима обучения

Данный метод обучения нейронной сети называют обобщенным дельта-правилом или правилом « Error backpropagation». Он был предложен в 1986 г. Румельхартом, Макклеландом и Вильямсом.

Этот алгоритм используется для минимизации отклонения реальных значений выходных сигналов нейронной сети от требуемых. В качестве функции

оценки работы ИНС будем использовать метод наименьших квадратов (квадратичная ошибка) [4]:

$$E(w) = \frac{1}{2}\sum_{i,k}\left(f_{i,k} - y_{i,k}^{T}\right)^2, \quad (1.13)$$

де $f_{i,k}$ – значение выходного сигнала k-го выходного нейрона сети при подаче на её входы i-го набора обучающих данных, $y_{i,k}^{(T)}$ – требуемое значение выходного сигнала k-го выходного нейрона для i-го набора данных для обучения. Суммирование ведется по всем нейронам выходного слоя.

В качестве метода минимизации функции $E(w)$, будем использовать методом градиентного спуска, который обеспечивает коррекцию весовых коэффициентов следующим образом:

$$\Delta w_{ij}^{(q)} = \eta \frac{\partial E}{\partial w_{ij}}, \quad (1.14)$$

где $\Delta W_{i,j}^{(q)}$ – величина изменения веса связи, соединяющей i-й нейрон $(q–1)$ слоя с j-м нейроном слоя q; η – коэффициент скорости обучения, $0<\eta<1$. Таким образом, вес связи изменяется пропорционально её вкладу в значение ошибки нейрона, для которого эта связь является входной, т.к. частная производная по весу $\frac{\partial E}{\partial w_j}$ показывает зависимость скорости изменения функции ошибки E от изменения этого веса.

Изменение веса связи определяется следующим образом:

$$\Delta w_{ij}^{(q)} = \eta \delta_j x_i, \quad (1.15)$$

где δj – значение ошибки j-го нейрона в слое q, xi – значение i-го входного сигнала для j-го нейрона слоя q. Данная формула применима и для настройки смещений нейронов, только вместо x_i необходимо подставить «1».

Значение ошибки нейрона определяется в зависимости от его положения в сети. Для нейронов выходного слоя:

$$\delta_j = \left(f_{i,k}(S)\right)' \left(f_{i,k} - y_{i,k}\right), \quad (1.16)$$

где $y_{i,k}$ – требуемое, а $f_{i,k}$ – фактическое значение выходного сигнала k-го нейрона для i-го набора данных из обучающей выборки, $(fi,k(S))'$ – значение производной активационной функции k-го нейрона для i-го набора обучающих данных. Если нейрон принадлежит одному из скрытых слоев, то:

$$\delta_i^{(q)} = \left(f_i^{(q)}(S)\right)' \sum_j w_{ij} \delta_i^{(q+1)}, \quad (1.17)$$

где $\delta_i^{(q)}$ – ошибка i-го нейрона в слое q, $\delta_i^{(q+1)}$ – ошибка j-го нейрона в $(q+1)$ слое, w_{ij} – вес связи, соединяющей эти нейроны, $(f_{i,k}(S))'$ – значение производной активационной функции i-го нейрона слоя q.

Из этого следует, что влияние каждого нейрона на величину ошибок нейронов следующего слоя, пропорционально значению ошибки этого нейрона.

1.2 Сверточные нейронные сети

С появлением больших объемов данных и больших вычислительных возможностей стали активно использоваться нейронные сети. Особую популярность получили сверточные нейронные сети.

Свёрточная нейронная сеть – архитектура искусственных нейронных сетей, предложенная Яном Лекуном, предназначенная для эффективного распознавания изображений.

Данная разновидность нейронных сетей использует некоторые особенности зрительной части коры головного мозга, где были открыты простые клетки, которые реагируют на прямые линии на изображении, расположенных под разными углами, и сложные клетки, реагирующих на активацию определённого набора простых клеток.

Свое название сверточные сети получили из-за присутствия операции свёртки, суть которой заключается в вычисление нового значения текущего пикселя, учитывая значения соседних пикселей. Для вычисления значения используется ядром свертки. Во время вычисления нового значения выбранного пикселя на него накладывается ядро свертки (матрица свертки), соседние пиксели так же накрываются ядром. Далее подсчитывается сумма, где слагаемыми являются произведения значений пикселей на значения ячейки ядра, накрывшей данный пиксель. Получившийся результат суммируется и записывается в аналогичную позицию выходного изображения.

В архитектуру сети заложены априорные знания из предметной области компьютерного зрения:

- пиксель изображения сильнее связан с соседним (локальная корреляция);
- объект на изображении может встретиться в любой части изображения.

Особое внимание сверточные нейронные сети получили после конкурса ImageNet в октябре 2012 года, который был посвящен распознаванию объектов на фотографиях. Победитель данного конкурса - Алекс Крижевский, используя сверточную нейронную сеть, значительно превзошёл остальных участников.

1.2.1 Архитектура сверточной нейронной сети

Архитектура сверточной нейронной сети представляет из себя чередование сверточных слоев, субдискретизирующих слоев (подвыборочный слой) и полносвязных слоев (fully-connected layer) на выходе. Все перечисленные виды слоев могут располагаться в нейронной сети произвольным образом.

В свою очередь, свёрточный слой представляет собой набора плоскостей, образующийся за счет того, что изображение предыдущего слоя сканируется

небольшим окном и пропускается сквозь набор весов, а результат отображается на соответствующий нейрон текущего слоя. Таким образом получается, что каждый нейрон выполняет свертку некоторой области предыдущего слоя. Следовательно, набор плоскостей представляет собой карты особенностей входного изображения, и каждая плоскость находит соответствующие признаки в любом месте предыдущего слоя, который содержит некоторое изображение.

На рисунке 1.5 изображена стандартная структура сверточной нейронной сети.

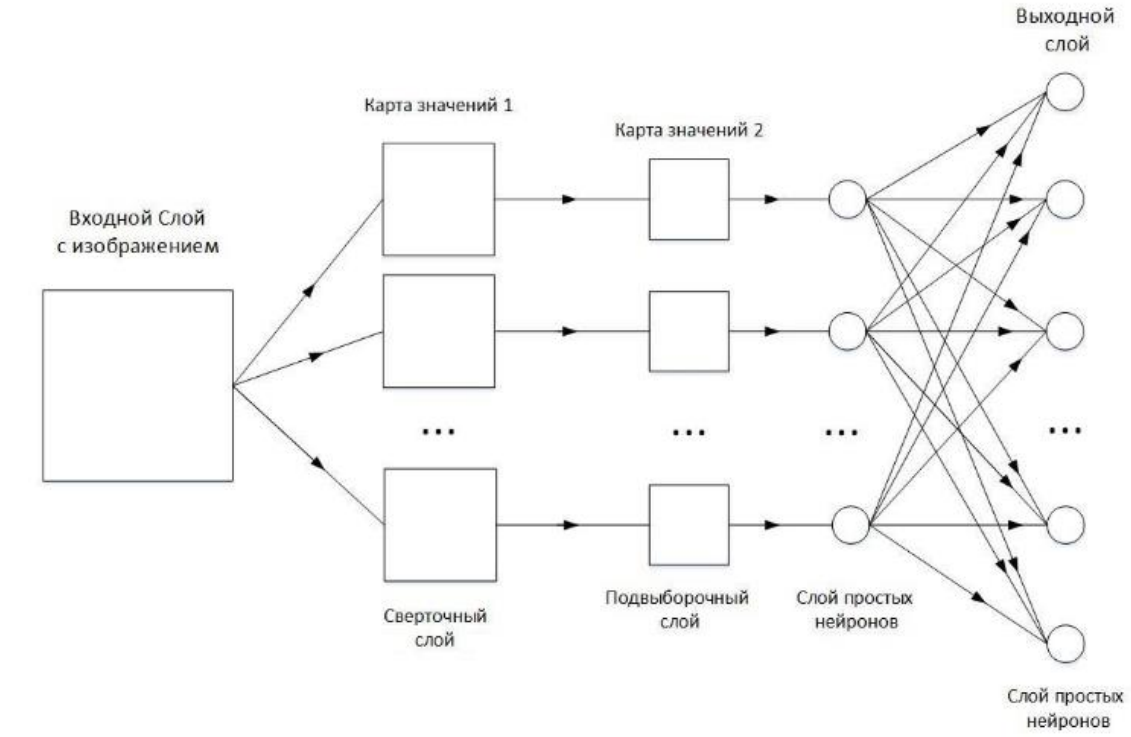

Рис. 1.5. Структура простой сверточной нейронной сети

1.2.2 Сверточный слой

Каждый нейрон находящийся в плоскости сверточного слоя получает свои входные значения от некоторой области предыдущего слоя, то есть изображение, которое подается на вход просматривается небольшим окном и

пропускается сквозь набор весов, полученный результат записывается в определенный нейрон сверточного слоя.

Принцип работы сверточного слоя представлен на рисунке 1.6.

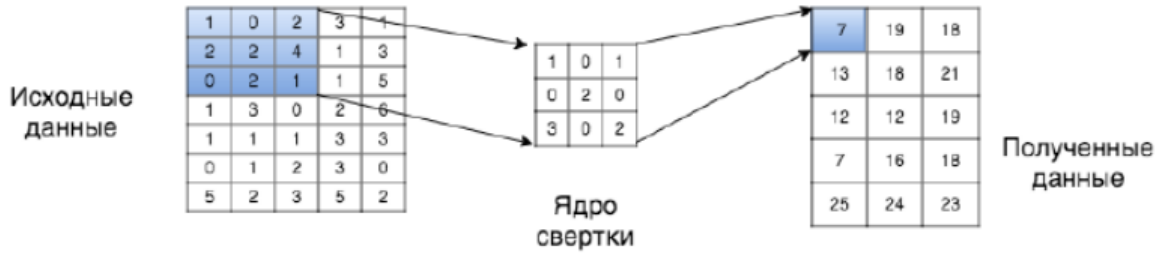

Рис. 1.6. Принцип работы сверточного слоя с ядром свертки 3×3

Таким образом, процесс функционирования нейрона свёрточного слоя $y_k^{(i,j)}$ задается выражением [14]:

$$y_k^{(i,j)} = b_k + \sum_{s=1}^{K} \sum_{t=1}^{K} w_{k,s,t} x^{((i-1)+s,(j+t))}, \quad (1.18)$$

где, $yk^{(i,j)}$ – нейрон k-ой плоскости свёрточного слоя, b_k – нейронной смещение k-ой плоскости, K - размер рецептивной области нейрона, $w_{k,s,t}$ – матрица синоптических коэффициентов, x – выходы нейронов предыдущего слоя.

Затем полученный результат свертки каждой локальной области, подается на активационную функцию:

$$out_k^{(i,j)} = f\left(y_k^{(i,j)}\right). \quad (1.19)$$

Отличительной особенностью сверточного слоя, является то что он немного уменьшает исходное изображение за счет краевых эффектов.

1.2.3 Подвыборочный слой

В современных ИНС используются субдискретизирующие слои (subsampling), выполняющие уменьшение размерности входной карты

признаков. Это можно делать разными способами, но чаще всего, для этого используется метод выбора максимального элемента (maxpooling) в окрестности текущего элемента. Вся карта признаков разделяется на ячейки, из которых выбираются максимальные по значению. Использование maxpooling позволяет сделать сеть инвариантной к масштабным преобразованиям.

На рисунке 1.7 показан пример субдискретизирующего слоя с методом выбора максимального элемента и размером окна 2×2.

Рис. 1.7. Принцип работы нейронов подвыборочного слоя

Этот слой обычно идет сразу за свёрточным, он также состоит из плоскостей, как правило, имеет такое же количество плоскостей, что и в предыдущем свёрточном слое.

1.2.4 Полносвязный слой

Слой, в котором каждый нейрон соединен со всеми нейронами на предыдущем уровне, причем каждая связь имеет свой весовой коэффициент. На рисунке 1.8 показан пример полносвязного слоя.

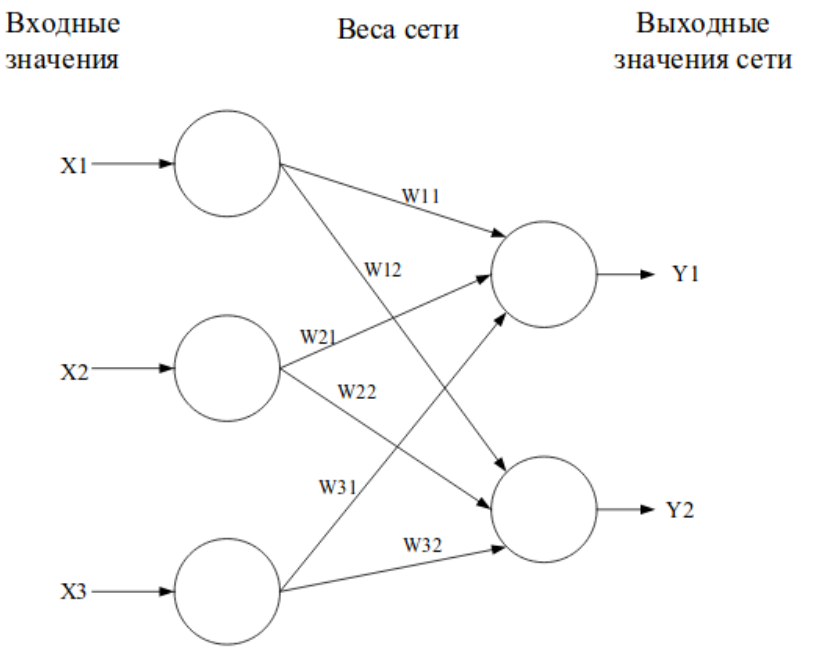

Рис. 1.8. Полносвязный слой

где w_{ij} – весовые коэффициенты связей, x_i – входные значения сигналов, y_i – выходные значения сети.

1.2.5 DropOut слой

Dropout является простым и эффективным методом регуляризации и заключающийся в том, что в процессе обучения сети из её совокупной топологии многократно случайным образом выделяется подсеть, и очередное обновление весов происходит только в рамках выделенной подсети. Каждый нейрон исключается из совокупной сети с некоторой вероятностью, которая называется коэффициентом dropout.

1.2.6 Преимущества и недостатки сверточных сетей

Преимущества [6, 7]:

- хорошо справляется с задачей классификации и распознаванию изображений;
- меньшее количество настраиваемых параметров, по сравнению с полносвязной нейронной сетью;
- более высокая устойчивость к повороту и сдвигу распознаваемого изображения, по сравнению полносвязными нейронными сетями (типа персептрона).

К недостаткам сверточной нейронной сети можно отнести большое количество настраиваемых гиперпараметров, к которым относятся:

- количество сверточных и подвыборочных слоев;
- количество карт признаков в сверточных и подвыборочных слоях;
- размерность матрицы свёртки;
- функция активации нейронов;
- скорость обучения сети.

Все выше перечисленные параметры существенно влияют на результат работы сверточной нейронной сети, и как правило выбираются эмпирически. Для распространенных задач существуют некоторые выверенные и прекрасно работающие конфигурации сетей, но для новых задач подбор гиперпараметров осуществляется опытным путем, так как не существует общего правила для их определения.

1.3 Алгоритмы сегментации

Сегментация строки на символы является одним из важнейших этапов в процессе оптического распознавания символов, в частности, при оптическом распознавании изображений документов. Сегментацией строки называется декомпозиция изображения, содержащего последовательность символов, на фрагменты, содержащие отдельные символы.

Важность сегментации обусловлена тем обстоятельством, что в основе большинства современных систем оптического распознавания текста лежат классификаторы (в том числе на основе нейронных сетей) отдельных символов, а не слов или фрагментов текста. В таких системах ошибки неправильного проставления разрезов между символами, как правило, являются причиной львиной доли ошибок конечного распознавания.

В данной работе сегментирование текстового изображения разделен на несколько этапов, с целью улучшения производительности разрабатываемой системы и повышения точности определения областей расположения отдельных символов.

Сегментацию изображения текста будем проводить в два этапа:

- выделение строк – исходное изображение текста необходимо «разрезать» на полосы-строки нужной ширины;
- сегментация символов – в изображении слова проводим границы символов.

1.3.1 Выделение строк

В данном случае полагается, что исходное изображение текста правильно ориентированно, т.е. строки ровные и картинка не повёрнута относительно наблюдателя. Такое предположение сделано на том основании, что оцифровка

исходного документа производилась с помощью сканирующего устройства, следовательно, это позволяет избежать непреднамеренного наклона изображения и разности освещенности, в отличии от использования ручной съемки с помощью фотокамеры.

В связи с этим, для выделения на исходном изображении строки был выбран пороговый алгоритм сегментации, основанный на использовании гистограммы [12, 15].

Данные алгоритмы эффективны, если сравнивать с другими методами сегментации, из-за того, что они делают всего один проход по пикселям. В этом методе гистограмма вычисляется по всем пикселям изображения и её минимумы и максимумы используются, чтобы найти кластеры на изображении. Цвет или яркость пикселя могут быть использованы при сравнении.

В данном случаи метод гистограмм используется для сегментации изображения на строки.

Данный метод предполагает построение гистограммы для черных пикселей бинарного изображения. В таком случае, по оси Y будет располагаться шкала распределения пикселей по количеству, а на оси X размешены номера строк.

Алгоритм основан на том что количество черных писклей в межстрочных интервалах существенно меньше чем в текстовых строках. Основываясь на этом предположении, определим каким должно быть наименьшее количество черных пикселей в строке, чтобы отнести ее к текстовой строке. Рассчитаем значение по формуле.

Сначала для всех пикселей строки исходного изображения находим количество черных пикселей:

$$s_i = s_i(B) = \sum_{i=1}^{n} b_{ij}. \quad (1.20)$$

Затем определяем среднее количество черных пикселей для всего изображения:

$$s(B) = \frac{1}{m}\sum_{j=1}^{m} s_j\,(B) \qquad (1.21)$$

Теперь исходя из полученного значения определим минимальный порог количества черных пикселей в текстовой строке:

$$\mathrm{N} = k\ \mathrm{s(B)} \qquad (1.22)$$

где k – коэффициент, принятый за 0.1.

Следовательно, используя найденный порог разделим все изображение на строки.

Работа алгоритма сегментации строк заключается в последовательном просмотре массива содержащего количество черных пикселей для каждой строки сравнение их с минимальным количеством N и выявлении множества пар индексов (s_i^{1}, s_i^{2}) строк, соответствующих границам печатных строк.

На рисунке 1.9 представлены результаты работы программы после построения гистограммы для фрагмента текста.

Рис. 1.9. Построение гистограмма для фрагмента изображения

На данном изображении явна выделяются участки гистограммы, где количество черных пикселей значительно больше, чем в других областях. Такие участки будут расценены программой как текстовые строки.

1.3.2 Выделение символов

Для выделения одиночных символов на изображении, был выбран метод связанных компонент [12], как наиболее подходящий для ращения поставленной задачи.

Под выделением связных компонент понимают присвоение уникальной метки каждому объекту изображения. При следующем анализе данные метки служат в качестве идентификаторов при обращении к объектам. Это делает операцию выделения связных компонент неотъемлемой частью почти всех приложений распознавания образов и компьютерного зрения.

Поэтому, перед тем как классификатор сможет определить какой объект находится на изображении, смежные пикселей одного объекта должны быть идентифицированы и промаркированы. Каждая выделенная группа пикселей соответствует одному объекту на изображении.

Метод связанных компонент хорошо подходит для сегментирования слов на отдельные символы, так как он предполагает выделение на исходных изображениях непрерывных участков с близкими по некоторым свойствам наборами точек.

Основные понятия:

Связанная область (участок) – набор точек на изображении, в котором любые две точки соединены друг с другом через последовательность соседей.

Виды связности:

- 4-связность - соседями точки считаются точки, имеющие с данной точкой одну общую сторону;
- 8-связность – соседями точки считаются точки, имеющие с данной точкой общую сторону или общий угол

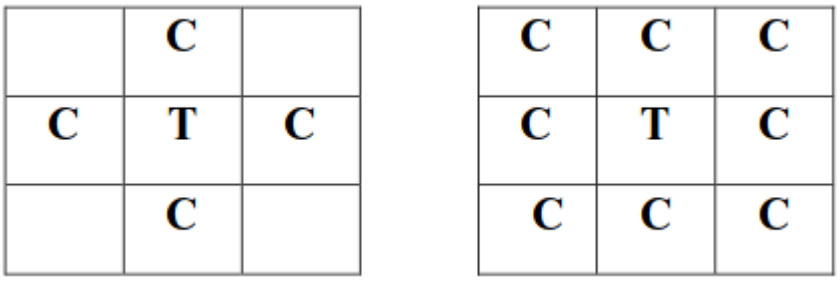

Рис. 1.10. «4-связанность» и «8-связанность» где С – сосед, Т – текущая (данная) точка.

Алгоритмы выделения связанных компонент можно разделить на следующие виды:

- многопроходные;
- двухпроходные;
- однопроходные;
- алгоритмы с использованием иерархических структур представления изображения;
- параллельные алгоритмы.

В данной работе будет рассмотрен двухпроходной алгоритм как наиболее подходящий для обработки монохромных изображений и наименее затратным по времени из всех выше предложенных алгоритмов.

Для описания алгоритма введем некоторые понятия. Обозначим через I матрицу изображения. Если $I(i, j)=0$, то пиксель является фоновым. Если $I(i, j)=1$, то пиксель принадлежит объекту. Через L обозначим двумерную матрицу, скан-маска, которая используется для хранения информации о метках и имеющую размеры, равные размерам изображения.

Двухпроходной алгоритм последовательно сканирует изображение. Первый проход по изображению осуществляется из верхнего левого угла, слева на право и сверху вниз. На внешнем цикле – перебор строк, на внутреннем – перебор столбцов строки, и анализируются только соседи сверху и слева. Тесть при первом проходе мы будем анализировать уже известные значения пикселей, относительно текущего пикселя.

Каждый раз при обнаружении черного пикселя его соседи, принадлежащие скан-маске, исследуются для определения метки, которая будет присвоена рассматриваемому пикселю. Если в скан-маске содержатся только белые пиксели, то текущий пиксель получает новую промежуточную метку. Если скан-маска содержит только один черный пиксель, то текущий пиксель получает метку соседа. Если скан-маска содержит несколько черных пикселей, то их промежуточные метки являются эквивалентными, среди них выбирается метка с наименьшим значением и пикселю присваивается значение выбранной метки.

После окончания первого прохода каждый объектный пиксель получает временную метку, на втором проходе значение метки уточняется. Второй сканпроход осуществляется в противоположенном направлении, снизу-вверх и справа налево. Во время второго прохода, алгоритм анализирует не только соседей сверху и слева, а все пиксели, имеющие с данным общую сторону или общий угол.

Не смотря на то что метод связанных компонент не является универсальным алгоритмом выделения символов в строке, в большинстве случаев он будет справляться с поставленной задачей.

После выделения, связанных копанет могут образоваться мелкие области, не представляющие интереса, к примеру остаточные шумы на изображении. Для устранения таких объектов, определим минимальный размер элемента – эталон, в случаи если объект меньше эталона, он не засчитывается за символ. Определение минимального размера происходит по формуле:

$$Smin = Smax * 0.2 , \quad (1.23)$$

где $Smax = a*b$, a и b соответственно ширина и высота самого большого объекта.

Данный алгоритм хорошо подходит для сегментации рукопечетного и печатного текста, который в отличии от рукописного не имеет связей между буквами и так как в данная работа нацелена на обработку именно таких текстов, он хорошо подходит для решения поставленной задачи и в большинстве

случаев дает корректные результаты сегментации, которые играют важную роль в последующем распознавании символов.

ГЛАВА 2. ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ

В качестве программной среды выбран продукт компании JetBrains IntelliJ IDEA 2018.1 и язык программирования Java.

Дополнительным средством для проведения экспериментов был выбраны язык программирования Python, так как это высокоуровневый язык программирования общего назначения, который имеет хорошую совместимость с различными фреймворками машинного обучения.

2.1 Обучающая выборка MNIST

В качестве одного из наборов, данных для обучения и тестирования сети будет использоваться база изображений рукописных цифр MNIST [11]. Изображения в данной базе имеют разрешение 28x28 пикселей и хранятся в формате оттенков серого, следовательно, каждое значение пикселя лежит в диапазоне от 0 (представляет белый цвет) до 255 (представляет черный цвет). Цифры отцентрированы на изображении. Данный набор разбит на две части: тренировочную и тестовую выборки состоящих из 60000 и 10000 изображений соответственно. На рисунке 2.1 показаны первые восемь изображений в обучающем наборе.

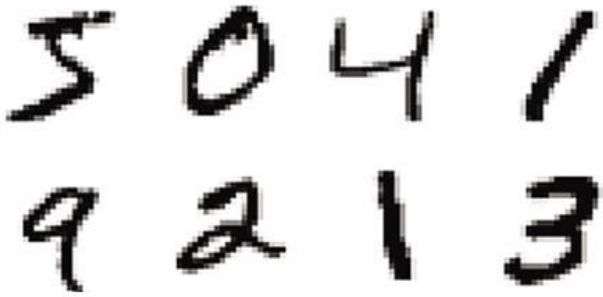

Рис. 2.1. Первые восемь обучающих изображений MNST

Формат файла обучающих пиксельных данных MNIST имеет начальное целочисленное значение (32 бита), равное 2051, за которым следует количество изображений (целочисленное значение), затем количество строк и количество столбцов (целочисленные значения), потом 60 000 изображений по 28 × 28 пикселей = 47 040 000 байтовых значений.

Обучающие, и тестовые данные хранятся в двух файлах, один файл содержит значения пикселей для изображений, а другой – метки изображений (0–9). Каждый из четырех файлов также содержит заголовочную информацию, и все они хранятся в двоичном формате, сжатом в формате gzip.

2.2 Обучающая выборка рукописных символов русского алфавита

Приведенный выше набор данных MNIST, не содержит в себе рукописных символов, которые необходимы для обучения сети распознавать текст. Для этих целый будет использоваться обучающая выборка, содержащая 146651 изображений рукописных символов, включающая в себя цифры, буквы русского алфавита, а также некоторые знаки препинания.

Рис. 2.2. Примеры элементов из обучающей выборки

Ниже приведена таблица 2.1 с частой встречаемости каждого символа в наборе данных рукописных букв русского алфавита.

Таблица 2.1

Частота распределения символов для букв русского алфавита

Символ	Кол-во	Символ	Кол-во	Символ	Кол-во	Символ	Кол-во
0	252	Б	2343	Л	3062	Ц	132

1	12803	В	8483	М	2276	Ч	1197
2	4939	Г	2096	Н	4299	Ш	887
3	6028	Д	5594	О	11834	Щ	55
4	6533	Е	11978	П	5484	Ъ	72
5	1852	Ё	12	Р	8776	Ы	3278
6	1152	Ж	806	С	4825	Ь	3744
7	1168	З	1779	Т	8284	Э	149
8	940	И	8291	У	829	Ю	467
9	456	Й	1688	Ф	15	Я	1908
А	6114	К	2934	Х	136		

Как можно видеть из графика, представленного на рисунке 2.3, не смотря на большое количество обучающих примеров в исходном наборе, количество букв в выборке распределено не равномерно, следовательно, было принято решение создать обучающую выборку, содержащую около 100 – 200 примеров для каждого символа.

Символы, которые не имели такого количества изображений были размножены за счет имеющихся изображений этих символов.

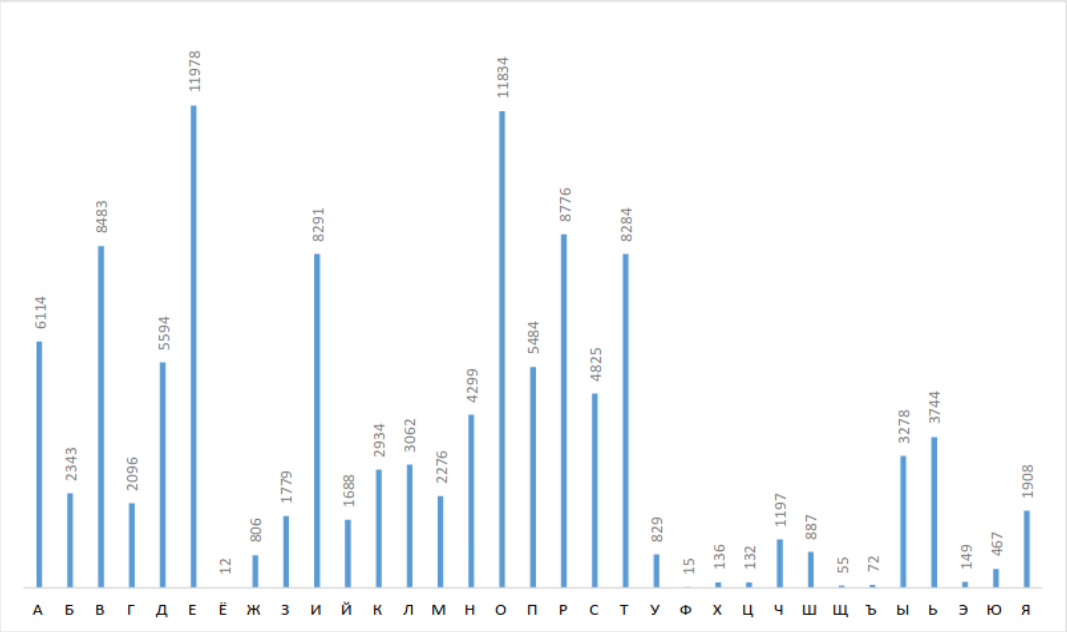

Рис. 2.3. Гистограмма распределения рукописных символов в выборке

Таблица 2.2 содержит количество изображений для каждого символа в получившейся выборке.

Таблица 2.2

Количество символов в сбалансированной выборке

Символ	Кол-во	Метка	Символ	Кол-во	Метка	Символ	Кол-во	Метка
А	209	0	Л	213	11	Ц	119	22
Б	210	1	М	207	12	Ч	197	23
В	209	2	Н	210	13	Ш	210	24
Г	212	3	О	200	14	Щ	100	25
Д	210	4	П	228	15	Ъ	100	26
Е	210	5	Р	211	16	Ы	210	27
Ж	225	6	С	210	17	Ь	210	28
З	211	7	Т	210	18	Э	149	29
И	209	8	У	210	19	Ю	210	30
Й	209	9	Ф	100	20	Я	210	31
К	204	10	Х	122	21			

Размер сбалансированной выборки составил – 6155 изображений, далее выборка была разделена на два набора обучающую и тестовую, в соотношении 9:1 соответственно. В результате размер обучающей выборки составил 5532 примеров, тестовой – 662 примеров.

Изображения в данной базе имеют размер 32х32 пикселей и хранятся в формате оттенков серого, следовательно, каждое значение пикселя лежит в диапазоне от 0 (черный цвет) до 255 (белый цвет), символы центрированы на изображении. Для ускорения работы сети необходимо инвертировать значения пикселей следующим образом 0 – белый цвет, 255 – черный цвет, по следующей формуле:

$$y_i = 255 - x_i \quad (2.1)$$

Рис. 2.4. Инвертированное изображение

Для ускорения сходимости обучения сети значения входных пикселей изображений нормализуются по формуле:

$$y_i = \frac{x_i}{255} \tag{2.2}$$

где x_i – значение i-го пикселя изображения из базы, y – значение, подаваемое на вход сети.

2.3 Подбор архитектуры сверточной нейронной сети

Большое количество обучающих и тестовых примеров требует достаточно большого времени обучения, поэтому, для подбора наилучшей конфигурации сети было решено воспользоваться готовыми библиотекам, а затем перейти к реализации собственной сверточной сети выбранной архитектуры.

2.3.1 Обзор библиотек машинного обучения

Для подбора наиболее удачных параметров разрабатываемой сети, был произведен обзор существующих библиотек машинного обучения. На основе выбранной библиотеки спроектирована нейронная сеть с различными

параметрами, для выявления наиболее подходящей конфигурации сети для решения поставленной задачи.

Существует множество программных средств для решения задач машинного обучения, в таблице 2 приведены наиболее популярные из них, они предоставляют возможности для создания полносвязных нейронных сетей (FC NN), сверточных нейронных сетей (CNN), автокодировщиков (AE) и ограниченных машин Больцмана (RBM).

Таблица 2.3

Обзор библиотек машинного обучения

Название	**Языки**	**ОС**	**FC NN**	**CNN**	**AE**	**RBM**
DeepLearnToolbox	Matlab	Windows, Linux	+	+	+	+
Theano	Python	Windows, Linux, Mac	+	+	+	+
Pylearn2	Python	Linux, Vagrant	+	+	+	+
Deepnet	Python	Linux	+	+	+	+
Torch	Lua, C	Linux, Mac OS X, iOS, Android	+	+	+	+
Darch	R	Windows, Linux	+	–	+	+
Caffe	C++, Python, Matlab	Linux, OS X	+	+	–	–
nnForge	C++	Linux	+	+	–	–
CXXNET	C++	Linux	+	+	–	–

Cuda-convnet	C++	Linux, Windows	+	+	–	–
Cuda CNN	Matlab	Linux, Windows	+	+	–	–

Из ходя из таблицы 2.3, была выбрана библиотека машинного обучения Theano, как наиболее оптимальная. Библиотека реализована на языке Python, поддерживается на операционных системах Windows, Linux и Mac OS X. В состав Theano входит компилятор, который переводит математические выражения, написанные на языке Python в эффективный код на C или CUDA.

Theano предоставляет базовый набор инструментов для конфигурации нейросетей и их обучения. Возможна реализация многослойных полностью связанных сетей (Multi-Layer Perceptron), сверточных нейронных сетей (CNN), рекуррентных нейронных сетей (Recurrent Neural Networks, RNN), авто-кодировщиков и ограниченных машин Больцмана. Также предусмотрены различные функции активации, в частности, сигмоидальная, Softmax-функция, RELU, кросс-энтропия.

Для работы с библиотекой машинного обучения Theano библиотека Keras.

Keras – это библиотека, позволяющая на более высоком уровне работать с нейросетями. Она упрощает множество задач, используется в быстрых экспериментах и сильно уменьшает количество однообразного кода. В качестве бекендной библиотеки для вычислений Keras может использовать Theano и TenzorFlow, в данной работе в качестве бекенда использовалась библиотека машинного обучения Theano.

2.3.2 Реализация простой нейронной сети

Действительно ли нам нужна такая сложная модель, как сверточные

нейронные сети, для получения высокой точности распознавания рукописных символов. В начале реализуем очень простую модель нейронной сети с одним скрытым слоем.

Спроектируем простую полносвязную многослойную модель персептрона, которая достигает точности распознавания около 90%. Полученные результаты будут использованы в качестве основы для сравнения с более сложной архитектурой сверточных нейронных сетей.

Для задачи распознавания рукописных символов из обучающей выборки MNIST с помощью многослойного персептрона элементы матрицы входного изображения записываются по строкам, в результате получается вектор x длины $28^2 = 784$ — признаковое описание объекта.

Ниже приведена реализация классической нейронной сети с 3 слоями для распознавания рукописных символов из обучающей выборки MNIST.

Таблица 2.4

Конфигурация классической нейронной сети

Слой	1	2	3
Тип слоя	Входной	Скрытый	Выходной
Функция активации	ReLU	ReLU	Softmax
Число нейронов	784	784	10
Число связей	–	615 440	7 850
Общее кол-во настраиваемых параметров	623 290		

На рисунке 2.5 визуализированная архитектура простой нейронной сети для распознавания рукописных цифр из обучающей выборки MNIST.

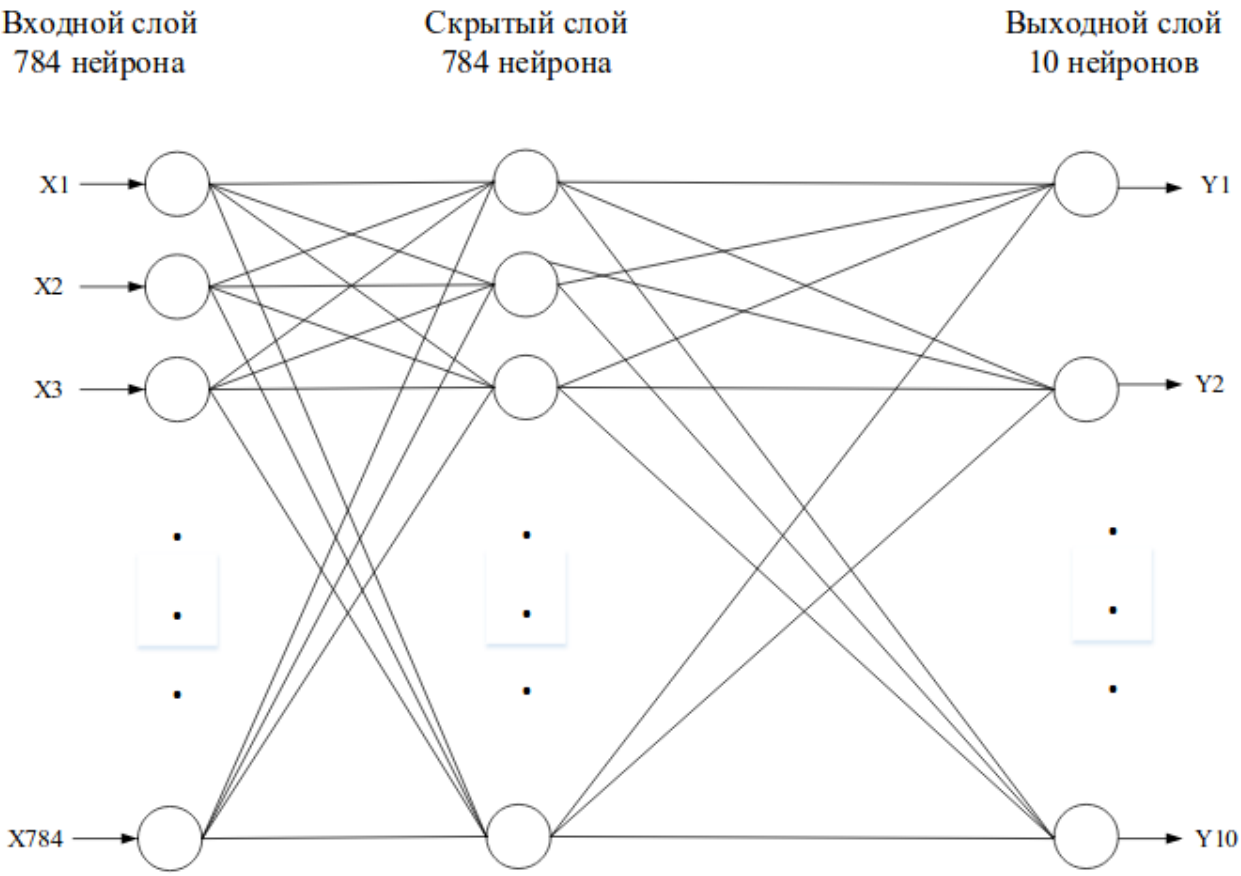

Рис. 2.5. Архитектура нейронной сети для распознаванья рукописных цифр

В таблице 2.5 приведены результаты тестирования классической однослойной нейронной сети для распознавания рукописных символов из обучающей выборки MNIST.

Для обучения сети использовались различные модификации метода градиентного спуска, коррекция весов производилась после 200 элементов из обучающей выборки, такой способ обучения сети, является оптимальным вариант между последовательным (стохастическим) и пакетным обучением.

Таблица 2.5

Влияние количества эпох на точность распознавания

Размер мини выборки = 200				
Количество эпох	Алгоритмы оптимизации			
	SGD	Adam	AdaGrad	AdaDelta

5	90.80%	98.03%	97.00%	97.00%
10	92.06%	98.23%	98.08%	97.74%
50	93.27%	98.38%	98.21%	98.35%

Также был проведен ряд экспериментов с изменением размера мини выборки. Их результаты приведены в таблице 2.6.

Таблица 2.6

Влияние размера мини выборки на точность распознавания

Количество эпох обучения = 5				
Размер мини выборки	Алгоритмы оптимизации			
	SGD	Adam	AdaGrad	AdaDelta
25	94.98%	98.01%	98.95%	98.23%
50	93.58%	97.88%	97.88%	97.72%
100	92.12%	97.85%	97.83%	97.52%
200	90.80%	98.03%	97.00%	97.00%

Дальнейшее увеличение количества эпох обучения не имеет смысла, так как точность на проверочной выборке после 25 эпохи стала колебаться ±0.02, что свидетельствует о переобучении сети.

В таблице 2.7 приведены результаты тестирования описанной выше архитектуры нейронной сети для набора данных рукописных букв. Архитектура сети была изменена следующим образом, входной слой и скрытый слой

содержит 32^2 = 1024 нейронов, выходной слой содержит 32 нейрона – по 1 нейрону на каждый класс (32 буквы – 32 класса).

Таблица 2.7

Конфигурация классической нейронной сети

Слой	1	2	3
Тип слоя	Входной	Скрытый	Выходной
Функция активации	ReLU	ReLU	Softmax
Число нейронов	1 024	1 024	32
Число связей	–	1 049 600	32 800
Общее кол-во настраиваемых параметров	1 082 400		

На рисунке 2.6. представлен архитектура нейронной сети для распознавания букв.

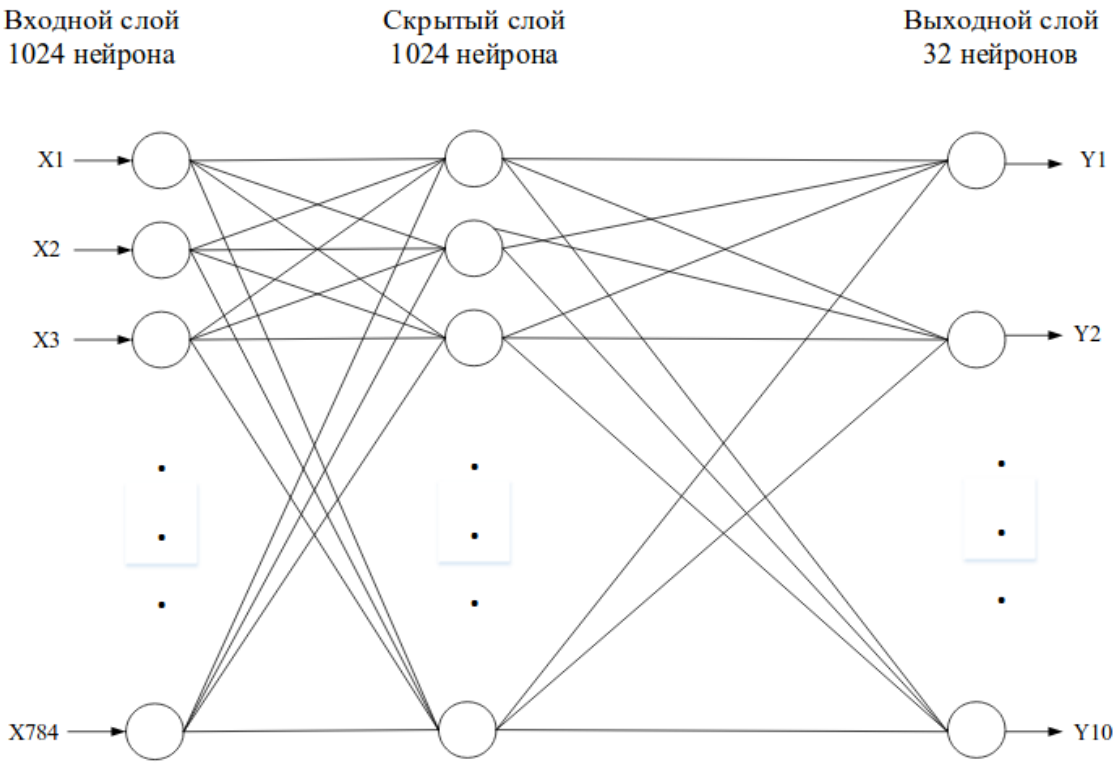

Рис. 2.6. Архитектура нейронной сети для распознавания букв

Таблица 2.8

Влияние количества эпох на точность распознавания

Размер мини выборки = 200				
Количество эпох	Алгоритмы оптимизации			
	SGD	Adam	AdaGrad	AdaDelta
5	67.20%	92.12%	91.96%	89.87%
10	78.14%	92.93%	92.44%	90.51%
50	87.46%	93.57%	93.09%	93.41%

Визуализируем изменение точности распознавания от эпохи к эпохе для проверочной выборки Результаты представлены на рисунках 2.7а и 2.7б. И графика видно, что после 15 эпохи значение точности начинает колебаться в пределах ±0.2, это свидетельствует о переобучение сити.

Следовательно, количество эпох равное 20, является избыточным и следует ограничить их количество в пределах 10-15 эпох.

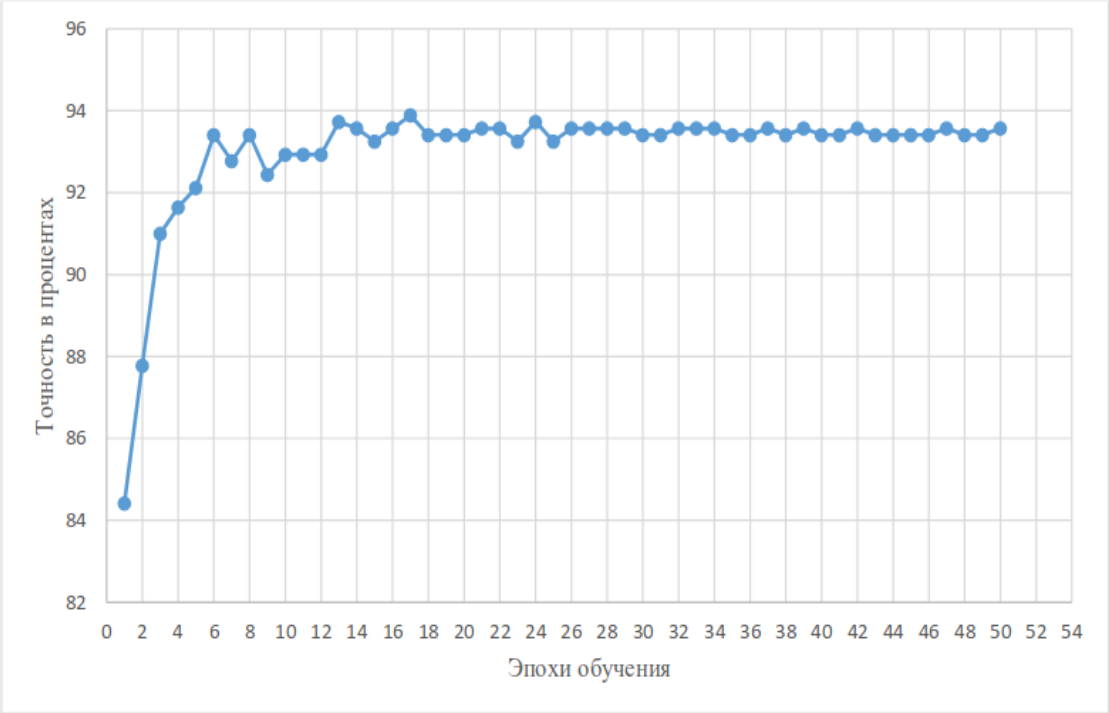

Рис. 2.7а. Изменение точности распознавания для тестовой выборки

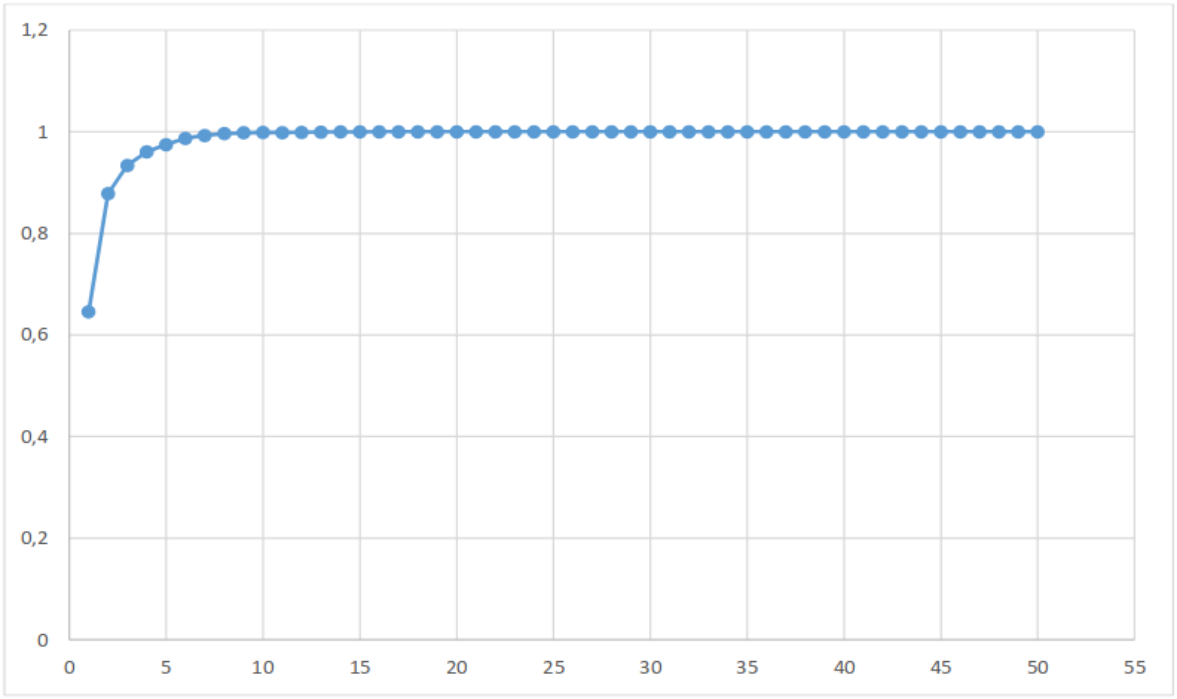

Рис. 2.7б. Изменение точности распознавания для обучающей выборки

В таблице 2.9 приведены результаты обучения сети, для разного количества эпох. Исходя из полученных данных, можно сделать вывод, что в данном случаи изменение количества эпох не значительно влияет на точность распознавания.

Таблица 2.9

Влияние размера мини выборки на точность распознавания

Количество эпох обучения = 15				
Размер мини выборки	Алгоритмы оптимизации			
	SGD	Adam	AdaGrad	AdaDelta
50	88.10%	94.05%	93.89%	93.41%
100	84.73%	93.73%	93.41%	92.44%
200	81.03%	93.25%	92.60%	90.84%

Лучшая точность распознавания, которой удалось добиться, используя простую многослойную нейронную сеть с одним скрытым слоем для классификации рукописных букв русского алфавита составила 94.05%. Данный результат несколько ниже точности полученной для данных из обучающей выборки MNIST, с похожей архитектурой сети.

Можно сделать вывод что такая разница в точности связанна с тем, что в наборе данных MNIST значительно больше обучающих примеров и меньше количество классов по сравнению с сформированной выборкой рукописных букв русского алфавита, где количество классов равно 32 и на каждый класс в среднем приходится 100 – 200 обучающих примеров.

2.3.3 Реализация сверточной нейронной сети

В данном разделе спроектируем сверточную нейронную сеть для распознавания рукописных букв. В таблице 8 представлена конфигурация сети с одним сверточным и подвыборочным слоем, размер матрицы свертки 5×5, размер окна подвыборки 2×2 (maxpooling).

Таблица 2.10

Конфигурация сверточной нейронной сети

Слой	1	2	3	4
Тип слоя	Слой свертки, Кол-во карт призна-ков: 32	Слой подвыборки Кол-во карт признаков: 32	Полносвя-занный слой, Кол-во нейро-нов:128	Полносвя-занный слой, Кол-во нейро-нов:32
Функция активации	ReLU	–	ReLU	Softmax

Количество настраиваемых параметров (весов)	832	–	802944	4128
Общее кол-во настраиваемых параметров	807 904			

На рисунке 2.8 визуализирована схема описанной в таблице 2.8 архитектуры сверточной нейронной сети для распознавания рукописных букв русского алфавита.

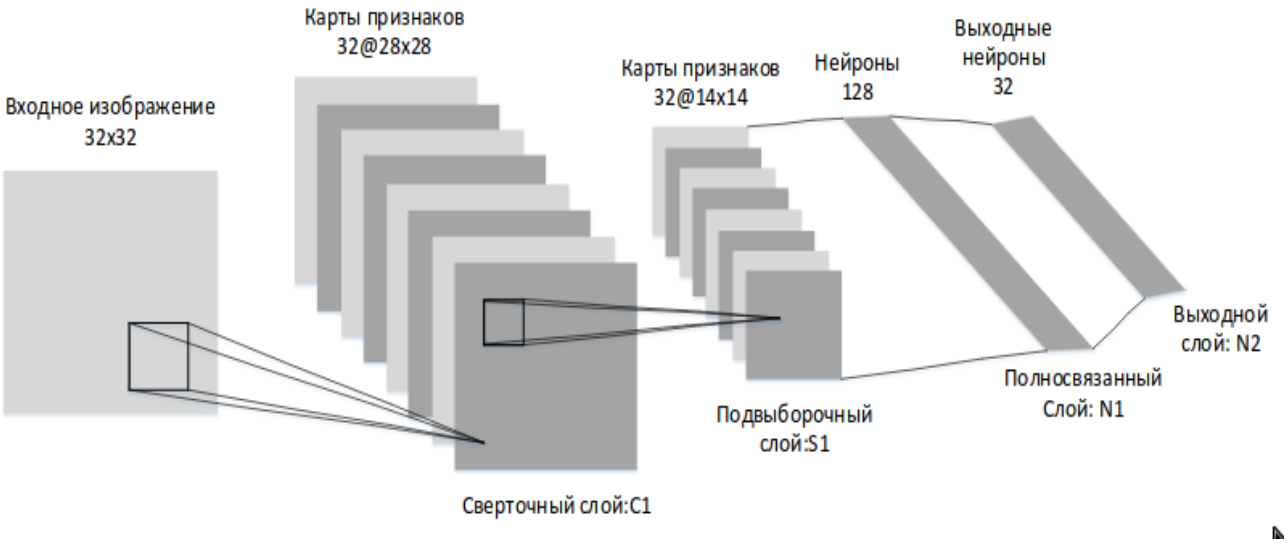

Рис. 2.8. Архитектура тестируемой сверточной нейронной сети

Ниже приведены результаты тестирования сверточной сети для набора данных рукописных букв. Точность распознавания на тестовой выборке представлена в таблицах 2.11 и 2.12.

Таблица 2.11

Влияние количества эпох на точность распознавания

Размер мини выборки = 100				
Количество эпох	Алгоритмы оптимизации			
	SGD	Adam	AdaGrad	AdaDelta

5	81.03%	94.53%	96.14%	93.09%
10	84.41%	95.18%	96.30%	94.21%
20	88.42%	95.98%	96.46%	95.82%

На рисунке 2.9, показан график изменение точности распознаванья сети на проверочные выборки. В качестве анализируемых данных были выбраны показатели для лучшей точность из таблицы 9, которая оставила 96.46%.

Проанализировав данный график можно сделать вывод что количество эпох равное 20 является избыточным, так как значительное изменение точности происходит с 1 по 10 эпоху, а после 10 эпохи значение точности начинают колебаться, что свидетельствует о переобучении сети.

Следовательно, для дальнейших экспериментов следует ограничить количество эпох обучения, не более 10.

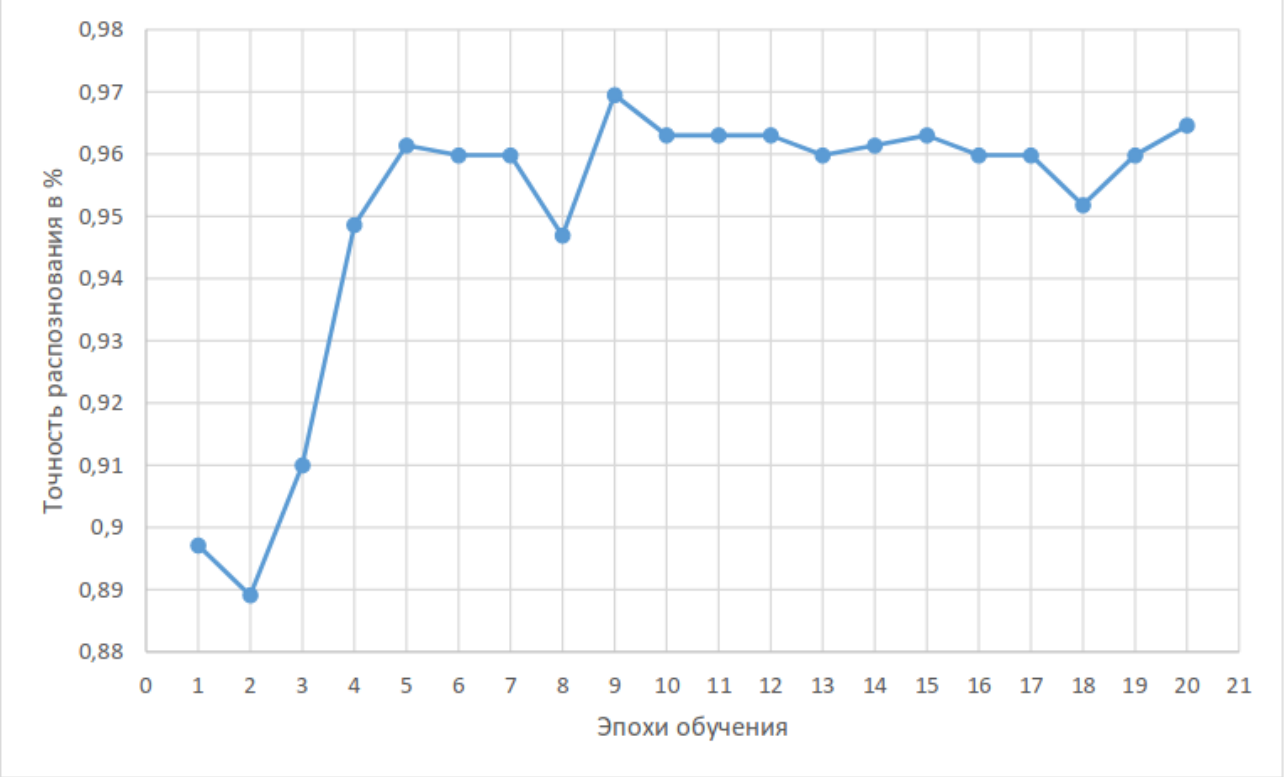

Рис. 2.9. Изменения точности распознаванья для тестовой выборки

Таблица 2.12

Влияние размера минивыборки на точность распознавания

Количество эпох = 10				
Размер мини выборки	Алгоритмы оптимизации			
	SGD	Adam	AdaGrad	AdaDelta
50	89.23%	95.98%	96.46%	95.18%
100	84.41%	95.18%	96.30%	94.21%
200	83.92%	94.53%	96.46%	94.21%

В таблице 2.11 приведены значения точности для разного количества эпох, лучшие результаты были получены для размера минивыборки 50 и 200, учитывая, что данные значения равны, в дальнейшем, при обучении предпочтительней использовать размер минивыборки = 200, так как он требует меньших затрат на обучение.

Исходя из данных таблиц 2.11 и 2.12, можно сделать вывод что уменьшение размера мини выборки и увеличения количества эпох незначительно влияет на точность распознавания. Следовательно, дальнейшая точность распознавания может быть увеличена либо за счёт увеличения карт признаков в сверточном слое, либо за счет увеличения глубины сети.

Таблица 13

Влияние количества карт признаков на точность распознавания

Количество эпох = 10, размер минивыборки = 100				
Количество карт призна-ков	Алгоритмы оптимизации			
	SGD	Adam	AdaGrad	AdaDelta

32	84.41%	95.18%	96.30%	94.21%
64	85.21%	95.34%	96.78%	95.98%

Увеличение карт признаков в сверточном слое не привело к значительному изменению точности распознавания, следовательно, можно попробовать увеличить точность за счет добавления сверточных и подвыборочных слоев. Архитектура сверточной сети с двумя слоями свертки описана в таблице 2.14.

Таблица 2.14

Конфигурация сверточной нейронной сети

Слой	1	2	3	4	5	6
Тип слоя	Слой свертки, Кол-во карт признаков: 16, 5×5	Слой подвыборки Кол-во карт признаков: 16	Слой свертки, Кол-во карт признаков: 32	Слой подвыборки Кол-во карт признаков: 32, 5×5	Полносвязанный слой, Кол-во нейронов:128	Полносвязанный слой, Кол-во нейронов:32
Функция активации	ReLU	–	ReLU	–	ReLU	Softmax
Количество настраиваемых параметров (весов)	416	–	12832	–	102528	4128
Общее кол-во настр. параметров	119 904					

В таблице 2.15 приведены результаты тестирования описанной выше архитектуры сверточной сети.

Таблица 2.15

Влияние размера минивыборки на точность распознавания

Количество эпох = 10				
Размер мини выборки	Алгоритмы оптимизации			
	SGD	Adam	AdaGrad	AdaDelta
100	84.89%	96.46	96.80%	95.96%
200	56.27	95.03	95.78%	94.80%

Максимальная точность работы на тестовом наборе данных составила 96.80%, что практически идентично результатам, полученным с помощью сети с одним сверточным слоем, исходя из этого можно сделать вывод, что в условиях небольшого количества обучающих данных достигнутая точность в 96.8% является максимально доступной для данного объема обучающего набора.

Так же было принято решения для дальнейшей работы использовать архитектуру с одним сверточным слоем, так как разница в точности по сравнению с сетью с двумя сверточными слоями незначительна, но реализации более простая архитектура дает преимущество в скорости обучении сети и дальнейшей ее эксплуатации.

Следовательно, оптимальной была выбрана архитектура описанной выше сверточной сети с одним скрытом слое и параметрами обучения:

- Алгоритм оптимизации: AdaGrad;
- Количество эпох обучения: 10;
- Размер минивыборки: 200.

2.4 Обзор основных блоков разрабатываемой системы

Блок сегментации изображения. Данный блок отвечает за определение области распознавания машиночитаемого бланка, выделение строк и символов

на изображение. Так же он включает в себя этап бинаризации исходного изображения, так как используемые в работе алгоритмы сегментации работают с бинарными изображениями.

Масштабирование и центрирование символа. Данный блок отвечает за масштабирование изображения символа, полученного после сегментации к размеру, на который ориентирована используемая нейронная сеть.

Инвертирование изображения. Данный блок выполняет инвертирование цветов входного изображения символа, в диапазоне от 0 до 255, где 0 – черный пиксель, 255 – белый.

Нормализация значений. Исходные значения пикселей входного изображения символа в диапазоне от 0 до 255, нормализуются в диапазоне от 0 до 1 с целью повышения эффективности работы сети.

Распознавание. На данном этапе происходит распознавание изображения с помощью, обученной сверточной нейронной сети.

На рисунке 21 представлена схемам основных блоков разрабатываемой системы, в виде их логической последовательности применения к входному изображению.

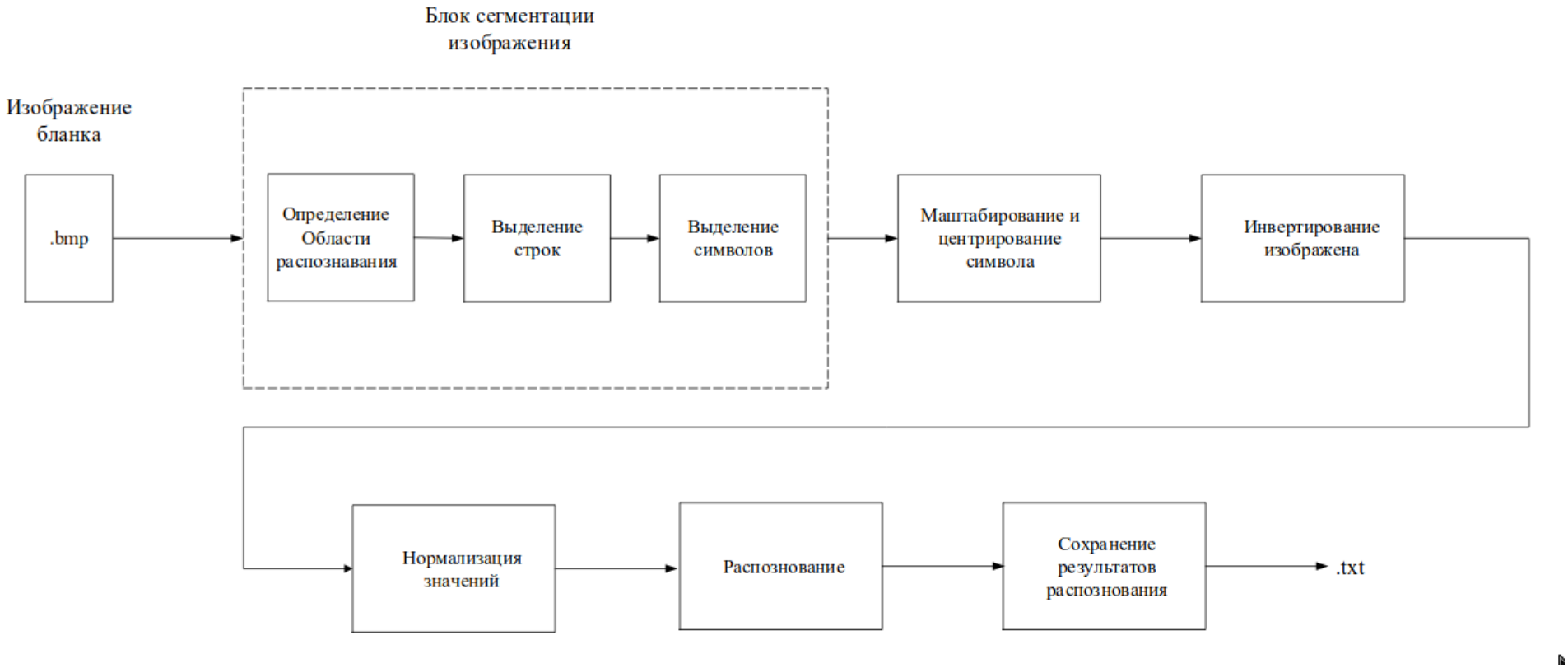

Рис. 2.10. Схема основных блоков системы

2.5 Проектирование логической модели системы

Построение логической модели начинается с определения классов и методов разрабатываемой системы. На рисунке 2.11 представлены основные классы системы в виде UML диаграммы, для двух основных блоков: сегментации и распознавания.

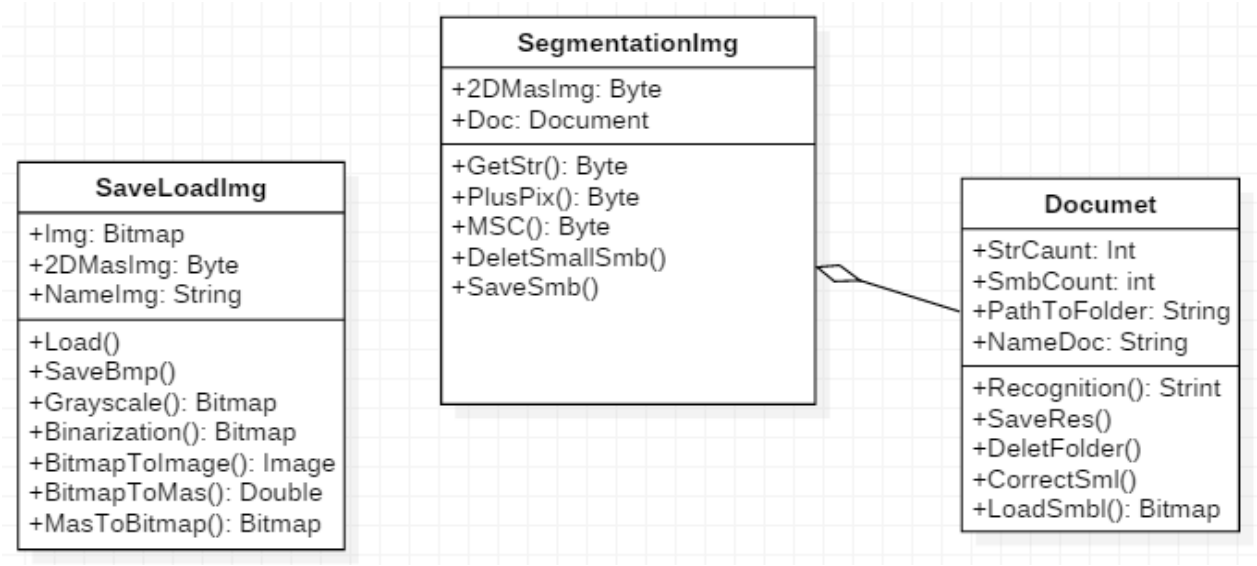

Рис. 2.11. UML диаграммы классов для блока сегментации

Класс SaveLoadImg – реализует основной функционал для загрузки исходного изображения и предобработки сходного изображения, для его последующий сегментации.

Класс SegmentationImg – отвечает за выделение строк и символов на исходном изображении, и сохраняет выделенные символы на диске.

Класс Document – содержит информацию о количестве строк и символов в распознаваемом документе. Основная задача этого класса – распознавание сегментированных символов с помощью обученной нейронной сети, которая подключается к проекту как сторонняя библиотека.

Так как разработанная нейронная сеть подключается к основному проекту в качестве библиотеки, ниже приведено описание ее основных классов. Исходя из архитектуры сети, были определены следующие классы:

Абстрактный класс BaseObject – описывает механизмы работы всех типов объектов проектируемой нейронной сети (нейроны, сверточные слои, подвыборочные слои), объявляет методы, ответственные за режимы обучение и коррекции весов;

Класс ConvolutionalPlane – реализует функционал сверточного слоя сети;

Класс SubsamplingPlane – реализует функционал подвыборочного слоя сети;

Класс Neuron – реализует функционал для работы нейронов полносвязаного слоя;

Класс Layer – описывает механизмы работы слоев нейронной сети. Абстрактный класс InputData – описывает механизмы обработки входных данных нейронной сети (в нашем случаи изображений рукописных символов).

Класс MNIST – описывает механизмы работ с базой данных рукописных символов MNIST или с любой другой базой изображений, имеющих аналогичную структуру хранения.

Класс NeuralNet – содержит в себе основные настраиваемые параметры сети, сведенья о ее общей структуре и значения результатов распознавания.

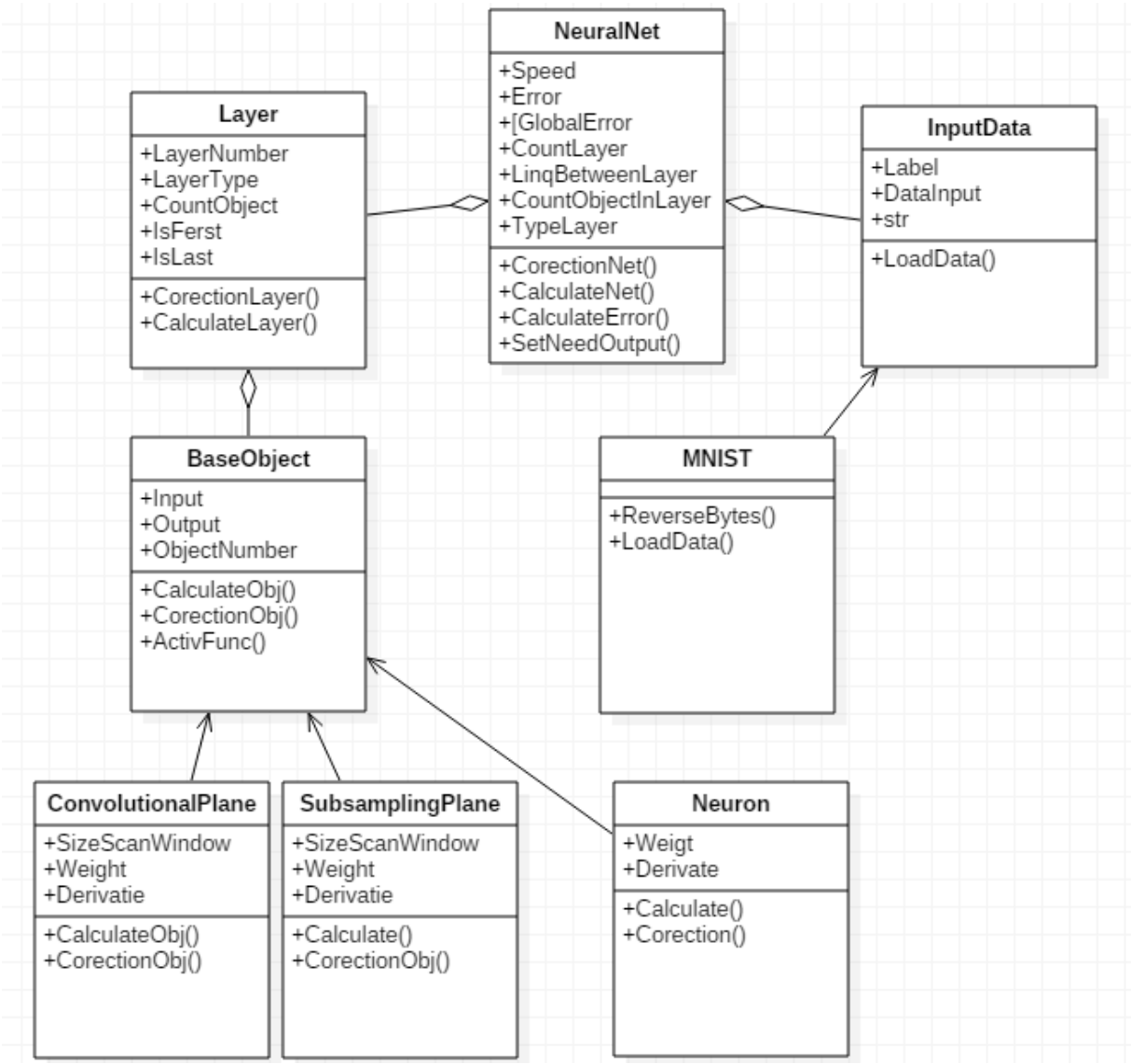

Рис. 2.12. Модель сверточной нейронной сети в виде UML диаграммы классов.

На данном этапе был определен основной функционал программы, были реализованы классы и методы, необходимые для работы выбранных алгоритмов сегментации и распознавания.

ГЛАВА 3. РЕЗУЛЬТАТЫ ПРОВЕДЕННОГО ИССЛЕДОВАНИЯ

3.1 Оценка результатов

Для оценивания результатов классификации была посчитана доля верно классифицированных объектов к общему количеству объектов. На данный момент лучшая точность распознавания, которой удалось добиться при реализации выбранной модели составляет 94.1%. Значение точности вычислялось по формуле:

$$R = \frac{n}{N} = \frac{623}{662} = 0.941\ , \tag{3.1}$$

где R – точность распознавания по всему набору тестовой выборки, n – количество правильно распознанных символов из тестовой выборки, N – количество элементов в тестовой выборке.

Достигнутая точность несколько отличается от той, которой удалось достигнуть при тестировании модели с помощью библиотеки машинного обучения. Данное различие связанно с различной настройкой некоторых гиперпараметров в ходе реализации модели.

На рисунке 3.1 представлены некоторые примеры некорректного распознавания различных символов.

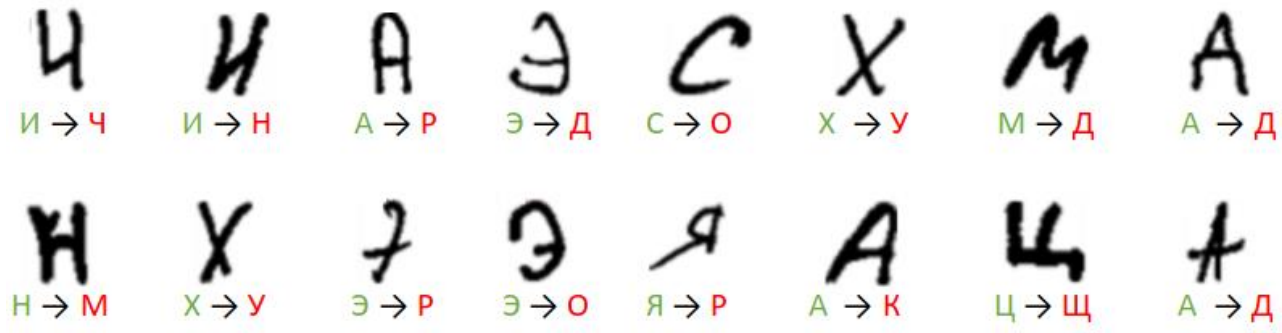

Рис. 3.1. Примеры неверно распознанных символов

На рисунках 3.2 и 3.3 изображена визуализация выхода сверточного и подвыборочного слоев, для символа «Я».

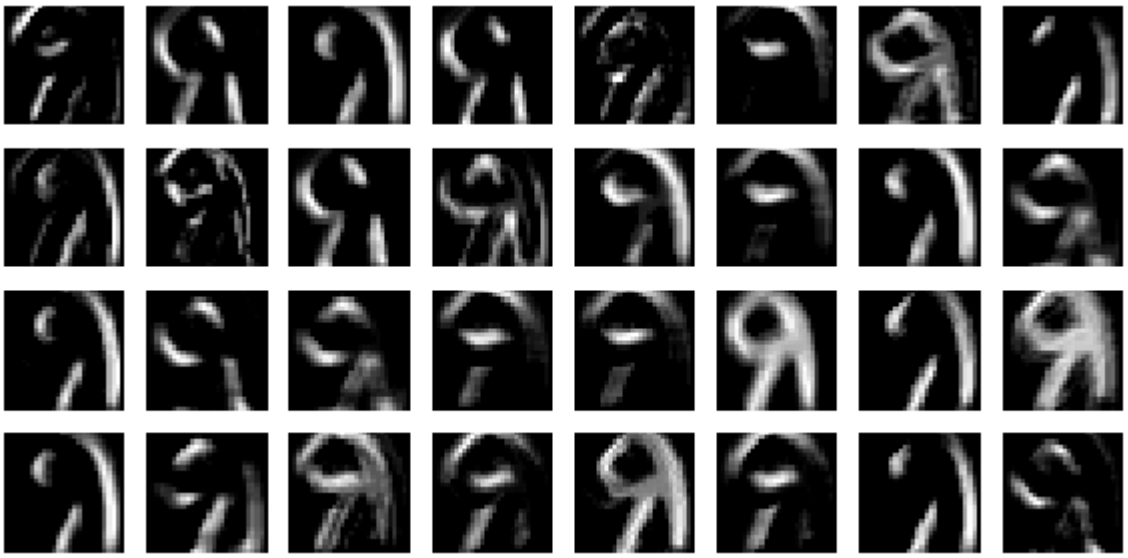

Рис. 3.2. Визуализация выходов сверточного слоя сети

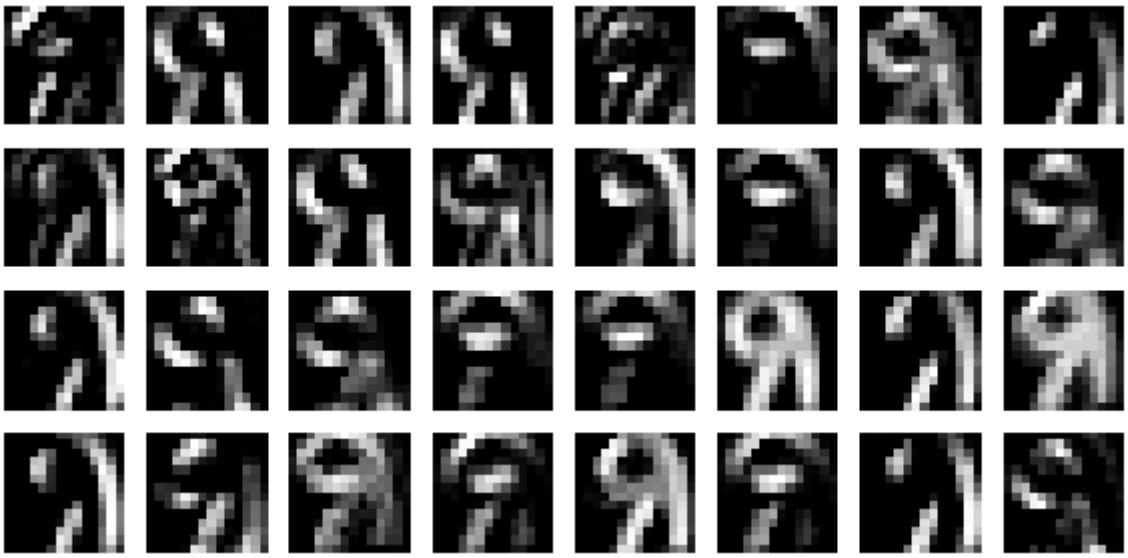

Рис. 3.3. Визуализация выходов подвыборочного слоя сети

Полученные изображения, наглядно демонстрируют принцип работы сверточной нейросети. Первый сверточный слой занимается выделением некоторых признаков на изображении с помощью матрицы свертки (чем больше карт в сверточном слое, тем больше признаков может выделить сеть). В последующем слое за сверточным – подвыборочный слой, который

уменьшает исходное изображение, делая сеть устойчивой к инвариантности и масштабированию.

3.2 Демонстрация работы прикладного приложения

Для демонстрации работы алгоритмов было разработано мобильное приложение под ОС Android. Его задачей является распознавание текста и его англо-русский и русско-английский перевод средствами сервиса Google Translate.

Приложение умеет распознавать как печатный, так и рукописный текст. Для ускорения обработки и отделения требуемого фрагмента текста от заведомого шума, добавлена возможность выделения области с помощью ограничивающей рамки.

Для начала, продемонстрируем работу на печатном английском тексте. На рисунках 3.5 а, б изображен процесс распознания.

Рис. 3.5 а. Исходное изображение с камеры

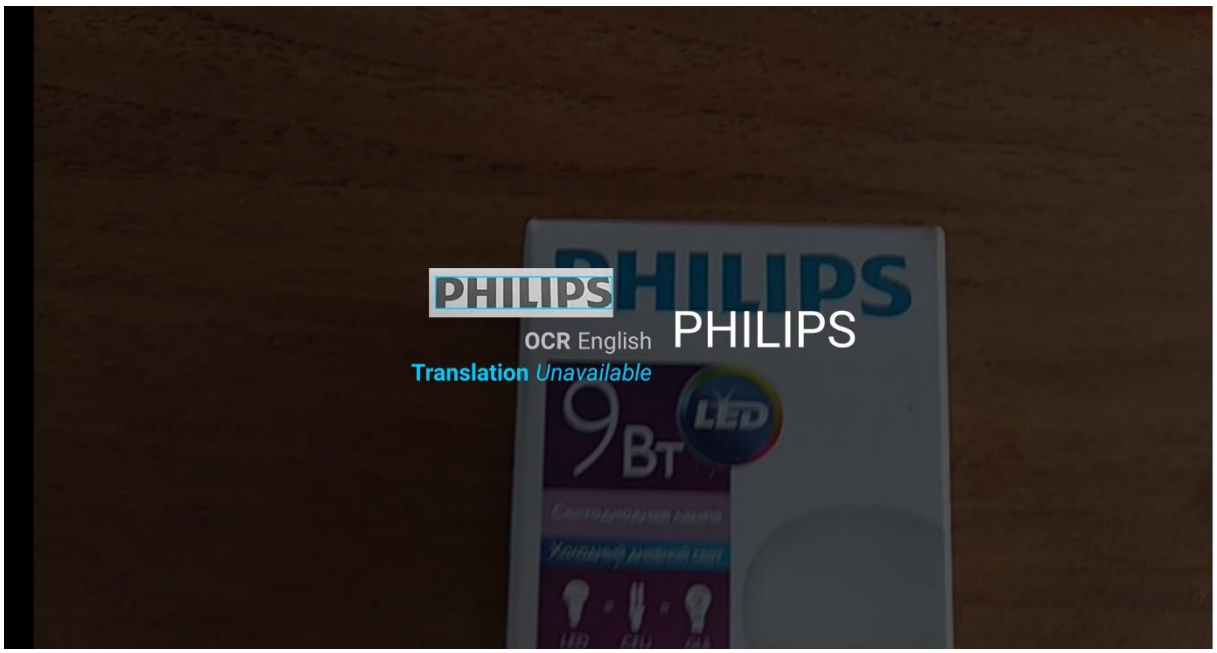

Рис. 3.5 б. Выделение текста и его распознание

Продемонстрирован пример работы со смешанным типом англоязычных символов и цифр на рисунке 3.6 а, б.

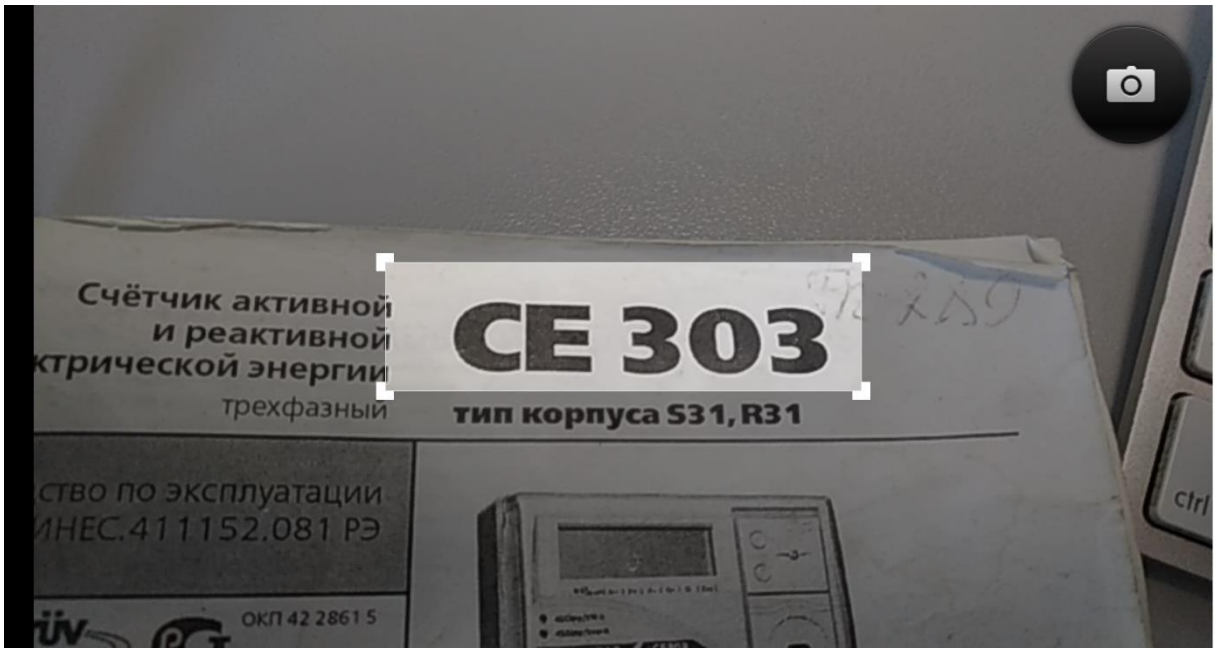

Рис. 3.6 а. Исходное изображение с камеры

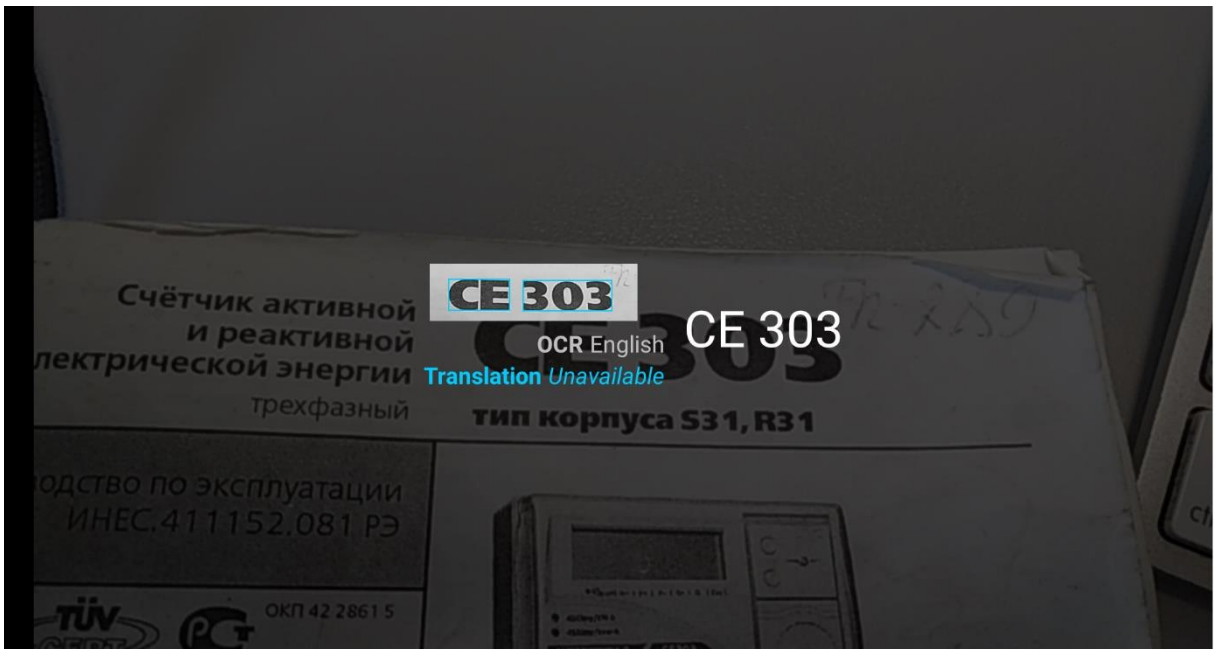

Рис. 3.6 б. Выделение смешанного текста и его распознание

Теперь продемонстрируем работу с английским рукопечатным текстом. Процесс изображен на рисунках 3.7 а, б. Как можно заметить, теперь доступен перевод.

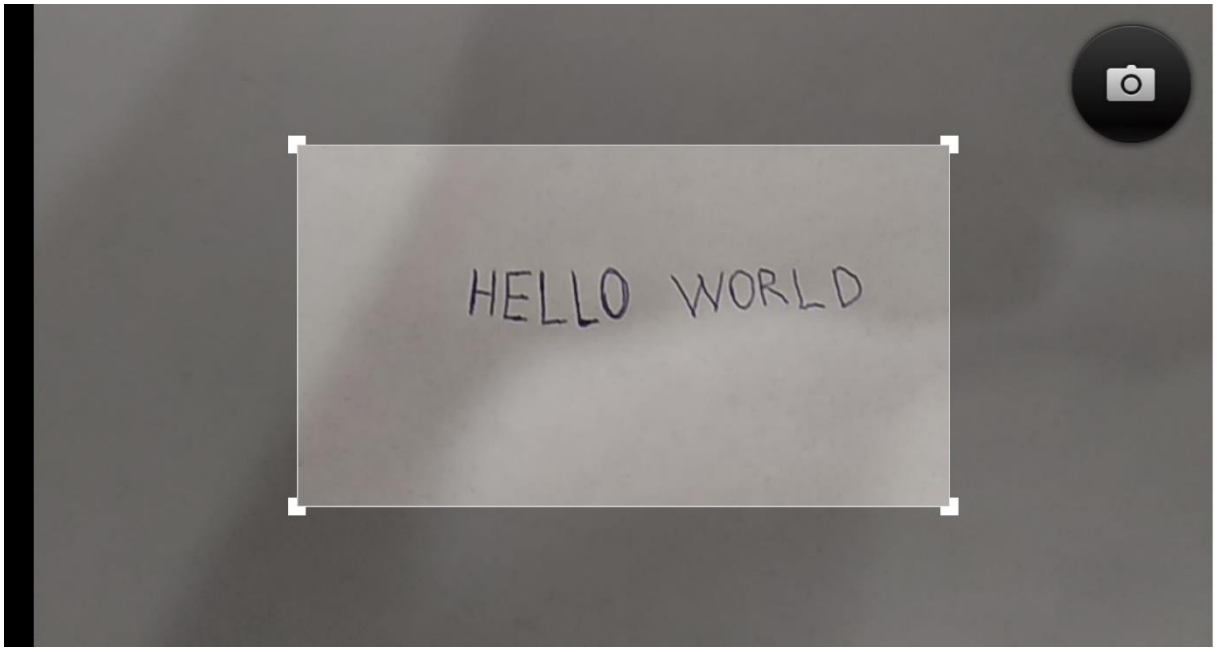

Рис. 3.7 а. Исходное изображение с текстом

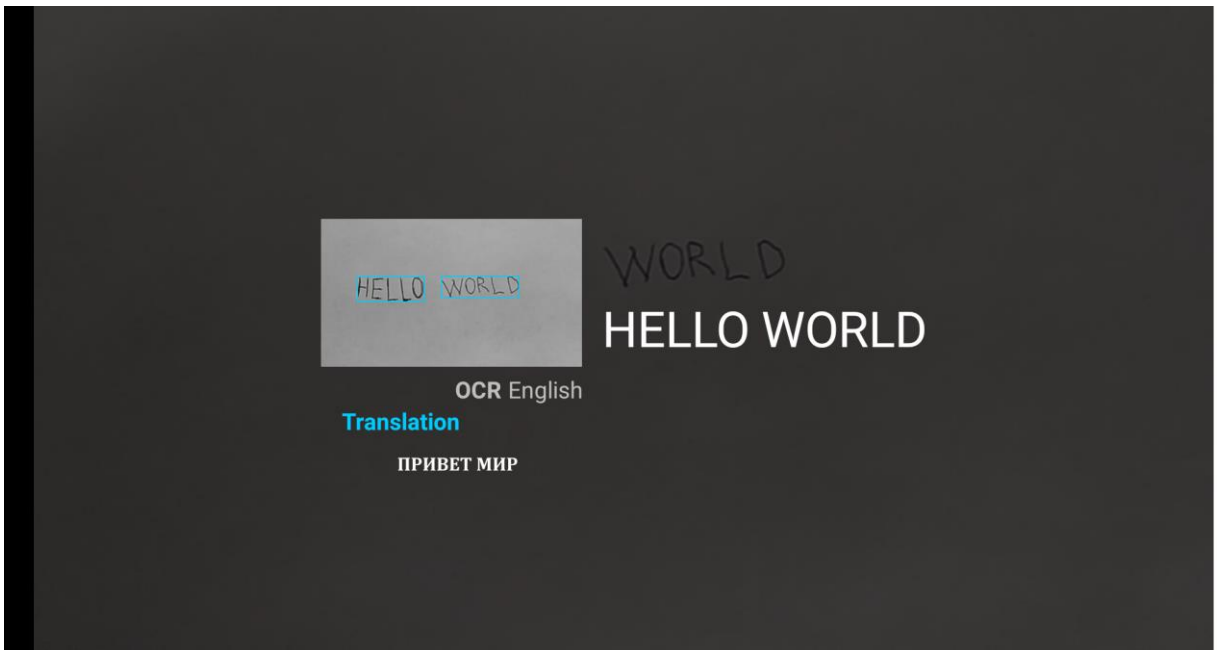

Рис. 3.7 б. Распознанный текст с переводом

В заключение продемонстрируем текст с русскими рукописными символами. Как можно заметить, также доступен русско-английский перевод. (Рис. 3.8 а и б)

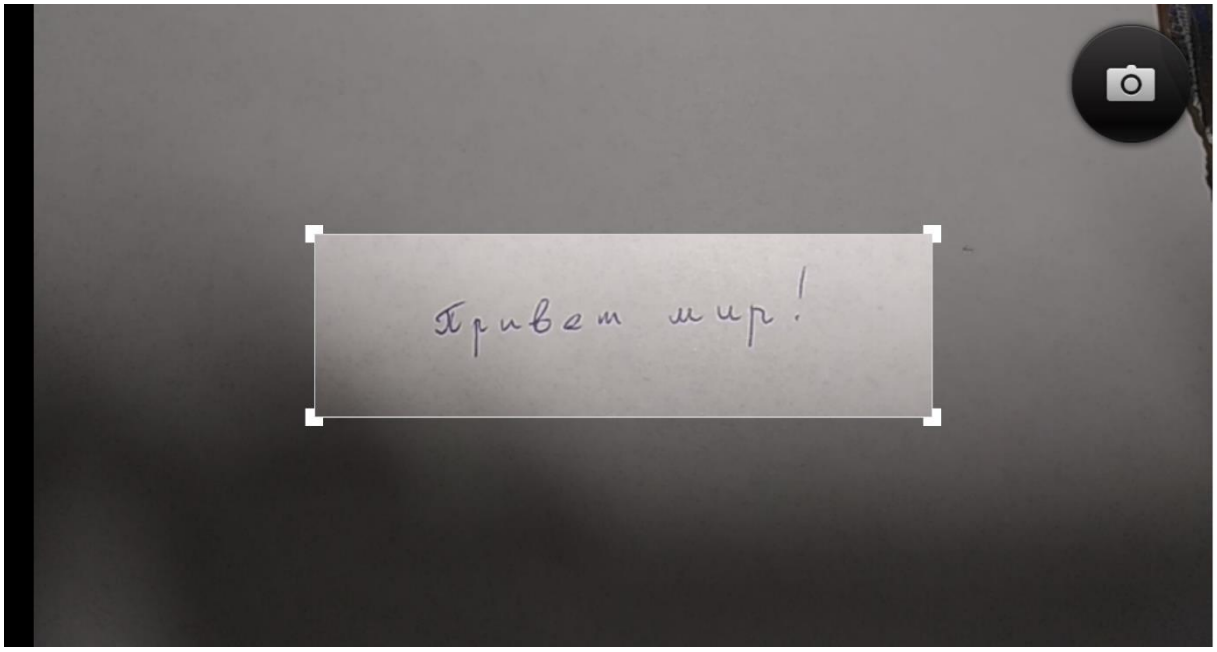

Рис. 3.8 а. Исходное изображение с текстом

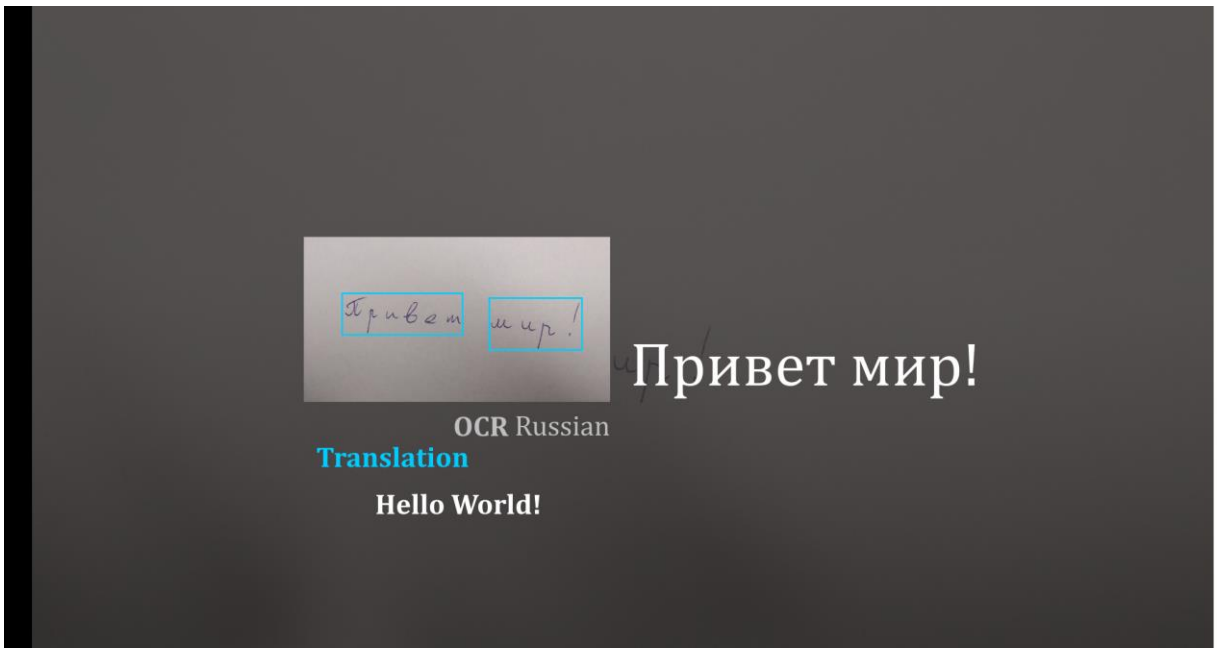

Рис. 3.8 б. Распознанный текст с переводом

Таким образом, мы наглядно продемонстрировали работоспособность системы и одну из возможностей для ее применения. Исходный код приложения находится на прилагаемом диске.

ЗАКЛЮЧЕНИЕ

Распознавание печатного и рукописного текста является важной задачей при обработке документов. Одним из самых распространенных методов машинного обучения при решении данной задачи являются нейронные сети.

В ходе работы был проведен сравнительный анализ использования сверточной и простой нейронной сети на основе персептрона для распознавания печатных и рукописных символов. Проведенные исследования показали, что наиболее подходящим решением для поставленных задач является использование сверточной нейросети, так как она имеет гораздо меньше настраиваемых параметров и более высокую точность распознавания.

Опытным путем была выявлена наиболее подходящая архитектура сверточной нейронной сети для решения поставленной задачи, исследованы алгоритмы оптимизации на основе метода градиентного спуска и виды функции активации нейронов. Все тесты проводились с использованием баз символов MNIST и базы данных символов русского алфавита.

Были исследованы алгоритмы сегментации текстовых изображений, на основе пороговых алгоритмов и методов связанных компонент.

Было разработано программное обеспечение, реализующее набор классов и методов для работы с выбранными алгоритмами распознавания и сегментации текста. Была осуществлена программная реализация предложенных алгоритмов, а также разработано прикладное мобильное приложение «переводчик» для распознавания текста со снимков и его перевода. Наилучший результат по точности распознавания составил 94,1% для изображений символов из тестовой выборки.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Хайкин С. Нейронные сети: полный курс. М.: Вильямс, 2006. - 1104с.
2. Маркелов А.А. Алгоритмы и программная система классификации полутоновых изображений на основе нейронных сетей: Диссертация на соискание ученой степени кандидата технических наук / А. А. Маркелов. – Томск, 2007.
3. Гонсалес. Р. Цифровая обработка изображений / Р. Гонсалес, Р. Вудс. – М.: Техносфера, 2005. – 1072с.
4. В. Г. Спицын, Интеллектуальные системы: учебное пособие / В. Г. Спицын, Ю.Р. Цой; Томский политехнический университет. – Томск: Изд-во Томского политехнического университета, 2012. – 176с.
5. LeCun, Y. Efficient BackProp in Neural Networks: Tricks of the trade / Y. LeCun, L. Bottou, G. Orr, K. Muller – Springer, 1998.
6. LeCun,Y. Scaling learning algorithms towards AI / Y. LeCun, Y. Bengio – MIT Press, 2007.
7. Шапиро Л. Компьютерное зрение / Л. Шапиро, Дж. Стакан, 206. – 53с.
8. Систематические методы распознавания образов/ Ю. Лившиц, 205. – 12с.
9. Грузман И.С., Киричук В.С., Косых В.П., Перетягин Г.И., Спектор А.А. Цифровая обработка изображений в информационных системах: Учеб. пособие. – Новосибирск.: Изд-во НГТУ, 2003. – 352с.
10. Солдатова О. П., Гаршин А. А. Применение сверточной нейронной сети для распознавания рукописных цифр. Компьютерная оптика. – 2010. – Том34, №2. – с. 252-260 – ISSN0134-2452.
11. LeCun Y. The MNIST database of handwritten digits / Y. LeCun. [Электронный ресурс] Точка доступа: http://yann.lecun.com/exdb/mnist
12. Ba L.J., Mnih V., Kavukcuoglu K. Multiple Object Recognition with Visual Attention // arXiv, 2014. [Электронный ресурс] Точка доступа: http://arxiv.org/abs/1412.7755.

13. Ba L.J., Kiros R., Hinton G. E. Layer Normalization // arXiv, 2016. [Электронный ресурс] Точка доступа: http://arxiv.org/abs/1607.06450
14. Boureau Y.-L., Ponce J., LeCun Y. A Theoretical Analysis of Feature Pooling in Visual Recognition. // Proc. 27th ICML, Omnipress, 2010. — P. 111-118.
15. Back-Propagation Applied to Handwritten Zip Code Recognition / Y. LeCun et al. // Neural Computation, 1989, vol. 1, no. 4. — P. 541—551.
16. A Backward Progression of Attentional Effects in the Ventral Stream / E. A. Buffalo et al. // Proc. National Academy of Sciences, 2010, vol. 107, no. 1. — P. 361—365.
17. Badnnarayanan V., KendallA., Cipolla R. SegNet: A Deep Convolutional Encoder-Decoder architecture for Image Segmentation // arXiv, 2015. [Электронный ресурс] Точка доступа: http://arxiv.org/abs/15t1.00561.
18. Bag of Tricks for Efficient Text Classification / A. Joulin et al. // arXiv, 2016. http://arxiv.org/abs/1607.759.
19. Concrete Sentence Spaces for Compositional Distributional Models of Meaning / E. (Grefenstette et al. // Meaning / Springer, 2014.
20. Conditional linage Generation with PixelCNN Decoders / A. van den Oord et al. // arXiv, 2016. [Электронный ресурс] Точка доступа: http://arxiv.org/abs/1606.05328.
21. A Context-Aware Topic Model for Statistical Machine Translation / J. Su et al. // Proc. 53rd ACL and the 7th I.JCNLP, vol. 1: Long Papers, Beijing, China: ACL, '2015. P. 229238.
22. Context-Dependent Pre-trained Deep Neural Networks / G. Dahl et al. / vol. 20,
23. Covariate Shift by Kernel Mean Matching / A. Gretton et al. // Dataset Shift in Machine Learning, 2009, vol. 3, no. 4. — P. 5.
24. Cox R. The Algebra of Probable Inference, Johns Hopkins Press Baltimore, 1961. — IM P. Creswell A., Bharath A. A. Denoising Adversarial Autoencoders

// arXiv, 2017. [Электронный ресурс] Точка доступа: http:// arxiv.org/abs/1703.01220.

25. Convolutional Neural Networks for Visual Recognition. [Электронный ресурс] Точка доступа: http://cs231n.github.io/neural-networks-3/#second.
26. cuDNN: Efficient Primitives for Deep Learning / S. Chetlur et al. // arXiv, [Электронный ресурс] Точка доступа: https://arxiv.org/abs/1410.0759.
27. Dataset Shift in Machine Learning /J. Quionero-Candela et al., The MIT Press, 2009.
28. DaugmanJ. G. Uncertainty Relation for Resolution in Space, Spatial Frequency, and Orientation Optimized by Two-Dirnensional Visual Cortical Filters // Journal of the Optical Society of America A, 1985, vol. 2, no. 7. P. 1160—1169.
29. Dayan P., Abott L. Theoretical Neuroscience, Cmnbridge, MA, USA: MIT Press, 2001.
30. Deep Generative Adversarial Networks for Cornpre.ssed Sensing Automates MRI / M. Mardani et al. // arXiv, 2017. [Электронный ресурс] Точка доступа: http://arxiv.org/abs/1706.00051.
31. Deep Learning for Visual Understanding: A Review / Y. Guo et al. // Neurocomputing, 2016, vol. 187. — P. 27 — 48, Recent Develomnents on Deep Big Vision.
32. Bogstra J., Breuleux O. et al. Deep Learning Tutorial: Theano Documentation, 2015.
33. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups / G. Hinton et al. // IEEE Signal Processing Magazine, Nov 2012, vol. 29, no. 6. P. 82-97.
34. Deep Reinforcement Learning for Robotic Manipulation / S. Gu et al. // arXiv, 2016. [Электронный ресурс] Точка доступа: http://arxiv.org/abs/1610.00633.
35. Deep Residual Learning for Image Recognition / K. He et al. // Proc. 2016 CVPR, 2016. — P. 770-778.

36. Deng L A Tutorial Survey of Architectures, Algorithms, and Applications for Deep Learning // APSIPA Transactions on Signal and Information Processing, 2014.
37. Онлайн-курсы от ведущих университетов [Электронный ресурс] Точка доступа: https://www.coursera.org
38. Mike O'Neill. Neural Network for Recognition of Handwritten Digits. [Электронный ресурс] Точка доступа: https://www.codeproject.com/Articles/16650/Neural-Network-for-Recognition-of-Handwritten-Digi
39. Поршнев С. В., Левашкина А. О. Универсальная классификация алгоритмов сегментации изображений // Журнал научных публикаций аспирантов и докторантов [Электронный ресурс] — Электронный научный журнал – 2006. – Режим доступа: http://jurnal.org/articles/2008/inf23.html
40. Введение в цифровую обработку изображений: лекция 3. [Электронный ресурс]– 2011. – Режим доступа: http://cvbeginner.blogspot.ru/2011/09/3.html

ПРИЛОЖЕНИЕ 1

```
import numpy
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dense, Flatten, Activation
from keras.layers import Dropout
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.utils import np_utils
from keras.optimizers import SGD

numpy.random.seed(42)

(X_train, y_train), (X_test, y_test) = cifar10.load_data()
batch_size = 32
nb_classes = 10
nb_epoch = 25
img_rows, img_cols = 32, 32
img_channels = 3

X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255

Y_train = np_utils.to_categorical(y_train, nb_classes)
Y_test = np_utils.to_categorical(y_test, nb_classes)

model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',
                 input_shape=(32, 32, 3), activation='relu'))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
```

```
model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(nb_classes, activation='softmax'))

sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy',
              optimizer=sgd,
              metrics=['accuracy'])

model.fit(X_train, Y_train,
              batch_size=batch_size,
              epochs=nb_epoch,
              validation_split=0.1,
              shuffle=True,
              verbose=2)

scores = model.evaluate(X_test, Y_test, verbose=0)
print("Точность работы на тестовых данных: %.2f%%" % (scores[1]*100))



import numpy
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import np_utils

numpy.random.seed(42)

(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

```
X_train = X_train.reshape(60000, 784)
X_test = X_test.reshape(10000, 784)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255

Y_train = np_utils.to_categorical(y_train, 10)
Y_test = np_utils.to_categorical(y_test, 10)

model = Sequential()
model.add(Dense(800, input_dim=784, activation="relu", kernel_initializer="normal"))
model.add(Dense(10, activation="softmax", kernel_initializer="normal"))
model.compile(loss="categorical_crossentropy", optimizer="SGD", metrics=["accuracy"])
print(model.summary())

model.fit(X_train, Y_train, batch_size=200, epochs=25, validation_split=0.2, verbose=2)

scores = model.evaluate(X_test, Y_test, verbose=0)
print("Точность работы на тестовых данных: %.2f%%" % (scores[1]*100))
```

```
import numpy
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.utils import np_utils

numpy.random.seed(42)

img_rows, img_cols = 28, 28
```

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()

X_train = X_train.reshape(X_train.shape[0], img_rows, img_cols, 1)
X_test = X_test.reshape(X_test.shape[0], img_rows, img_cols, 1)
input_shape = (img_rows, img_cols, 1)



X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255

Y_train = np_utils.to_categorical(y_train, 10)
Y_test = np_utils.to_categorical(y_test, 10)

model = Sequential()

model.add(Conv2D(75, kernel_size=(5, 5),
                 activation='relu',
                 input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))
model.add(Conv2D(100, (5, 5), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(500, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
print(model.summary())

model.fit(X_train, Y_train, batch_size=200, epochs=10, validation_split=0.2, verbose=2)
```

```
scores = model.evaluate(X_test, Y_test, verbose=0)
print("Точность работы на тестовых данных: %.2f%%" % (scores[1]*100))
```

ПРИЛОЖЕНИЕ 2

```
import numpy
import math
import time
import scipy.io
import scipy.signal
import scipy.optimize
import matplotlib.pyplot


class ConvolutionalNeuralNetwork(object):

    def __init__(self, W1, b1, zca_white, mean_patch, patch_dim, pool_dim):

        """ Store the weights, taking into account preprocessing done """

        self.W = numpy.dot(W1, zca_white)
        self.b = b1 - numpy.dot(self.W, mean_patch)

        self.patch_dim = patch_dim
        self.pool_dim  = pool_dim


    def sigmoid(self, x):
        return (1 / (1 + numpy.exp(-x)))


    def convolve(self, input_images, num_features):

        image_dim      = input_images.shape[0]
        image_channels = input_images.shape[2]
        num_images     = input_images.shape[3]

        conv_dim          = image_dim - self.patch_dim + 1
```

```
        convolved_features = numpy.zeros((num_features, num_images, conv_dim, conv_dim));

        for image_num in range(num_images):

            for feature_num in range(num_features):

                convolved_image = numpy.zeros((conv_dim, conv_dim))

                for channel in range(image_channels):
                    limit0  = self.patch_dim * self.patch_dim * channel
                    limit1  = limit0 + self.patch_dim * self.patch_dim
                    feature = self.W[feature_num, limit0 : limit1].reshape(self.patch_dim, self.patch_dim)

                    image = input_images[:, :, channel, image_num]

                    convolved_image = convolved_image + scipy.signal.convolve2d(image, feature,
'valid');

                convolved_image = self.sigmoid(convolved_image + self.b[feature_num, 0])
                convolved_features[feature_num, image_num, :, :] = convolved_image

        return convolved_features


    def pool(self, convolved_features):
        num_features = convolved_features.shape[0]
        num_images   = convolved_features.shape[1]
        conv_dim     = convolved_features.shape[2]
        res_dim      = conv_dim / self.pool_dim

        pooled_features = numpy.zeros((num_features, num_images, res_dim, res_dim))

        for image_num in range(num_images):

            for feature_num in range(num_features):
```

```
            for pool_row in range(res_dim):

                row_start = pool_row * self.pool_dim
                row_end   = row_start + self.pool_dim

                for pool_col in range(res_dim):

                    col_start = pool_col * self.pool_dim
                    col_end   = col_start + self.pool_dim

                    patch = convolved_features[feature_num, image_num, row_start : row_end,
                                               col_start : col_end]
                    pooled_features[feature_num, image_num, pool_row, pool_col] =
numpy.mean(patch)

        return pooled_features


class SoftmaxRegression(object):

    def __init__(self, input_size, num_classes, lamda):

        self.input_size  = input_size  # input vector size
        self.num_classes = num_classes # number of classes
        self.lamda       = lamda       # weight decay parameter

        rand = numpy.random.RandomState(int(time.time()))

        self.theta = 0.005 * numpy.asarray(rand.normal(size = (num_classes*input_size, 1)))


    def getGroundTruth(self, labels):

        labels = numpy.array(labels).flatten()
```

```
    data   = numpy.ones(len(labels))
    indptr = numpy.arange(len(labels)+1)

    ground_truth = scipy.sparse.csr_matrix((data, labels, indptr))
    ground_truth = numpy.transpose(ground_truth.todense())

    return ground_truth


def softmaxCost(self, theta, input, labels):
    ground_truth = self.getGroundTruth(labels)
    theta = theta.reshape(self.num_classes, self.input_size)

    theta_x       = numpy.dot(theta, input)
    hypothesis    = numpy.exp(theta_x)
    probabilities = hypothesis / numpy.sum(hypothesis, axis = 0)

    cost_examples    = numpy.multiply(ground_truth, numpy.log(probabilities))
    traditional_cost = -(numpy.sum(cost_examples) / input.shape[1])

    theta_squared = numpy.multiply(theta, theta)
    weight_decay  = 0.5 * self.lamda * numpy.sum(theta_squared)

    cost = traditional_cost + weight_decay

    theta_grad = -numpy.dot(ground_truth - probabilities, numpy.transpose(input))
    theta_grad = theta_grad / input.shape[1] + self.lamda * theta
    theta_grad = numpy.array(theta_grad)
    theta_grad = theta_grad.flatten()

    return [cost, theta_grad]


def softmaxPredict(self, theta, input):
```

```
        theta = theta.reshape(self.num_classes, self.input_size)

        theta_x       = numpy.dot(theta, input)
        hypothesis    = numpy.exp(theta_x)
        probabilities = hypothesis / numpy.sum(hypothesis, axis = 0)

        predictions = numpy.zeros((input.shape[1], 1))
        predictions[:, 0] = numpy.argmax(probabilities, axis = 0)

        return predictions


def loadTrainingDataset():

    train_data   = scipy.io.loadmat('stlTrainSubset.mat')
    train_images = numpy.array(train_data['trainImages'])
    train_labels = numpy.array(train_data['trainLabels'])

    return [train_images, train_labels]


def loadTestDataset():

    test_data   = scipy.io.loadmat('stlTestSubset.mat')
    test_images = numpy.array(test_data['testImages'])
    test_labels = numpy.array(test_data['testLabels'])

    return [test_images, test_labels]


def visualizeW1(opt_W1, vis_patch_side, hid_patch_side):

    figure, axes = matplotlib.pyplot.subplots(nrows = hid_patch_side,
                                              ncols = hid_patch_side)
```

```
    opt_W1 = (opt_W1 + 1) / 2

    """ Define useful values """

    index  = 0
    limit0 = 0
    limit1 = limit0 + vis_patch_side * vis_patch_side
    limit2 = limit1 + vis_patch_side * vis_patch_side
    limit3 = limit2 + vis_patch_side * vis_patch_side

    for axis in axes.flat:

        img = numpy.zeros((vis_patch_side, vis_patch_side, 3))

        img[:, :, 0] = opt_W1[index, limit0 : limit1].reshape(vis_patch_side, vis_patch_side)
        img[:, :, 1] = opt_W1[index, limit1 : limit2].reshape(vis_patch_side, vis_patch_side)
        img[:, :, 2] = opt_W1[index, limit2 : limit3].reshape(vis_patch_side, vis_patch_side)

        image = axis.imshow(img, interpolation = 'nearest')
        axis.set_frame_on(False)
        axis.set_axis_off()
        index += 1

    matplotlib.pyplot.show()


def getPooledFeatures(network, images, num_features, res_dim, step_size):

    num_images = images.shape[3]

    pooled_features_data = numpy.zeros((num_features, num_images, res_dim, res_dim))

    for step in range(num_images / step_size):

        """ Limits to access batch of images """
```

```
        limit0 = step_size * step
        limit1 = step_size * (step+1)

        image_batch = images[:, :, :, limit0 : limit1]

        convolved_features = network.convolve(image_batch, num_features)
        pooled_features    = network.pool(convolved_features)

        pooled_features_data[:, limit0 : limit1, :, :] = pooled_features

        del(image_batch)
        del(convolved_features)
        del(pooled_features)

    input_size = pooled_features_data.size / num_images
    pooled_features_data = numpy.transpose(pooled_features_data, (0, 2, 3, 1))
    pooled_features_data = pooled_features_data.reshape(input_size, num_images)

    return pooled_features_data


def executeConvolutionalNeuralNetwork():

    image_dim       = 64
    image_channels  = 3
    vis_patch_side  = 8
    hid_patch_side  = 20
    pool_dim        = 19

    visible_size = vis_patch_side * vis_patch_side * image_channels
    hidden_size  = hid_patch_side * hid_patch_side
    res_dim      = (image_dim - vis_patch_side + 1) / pool_dim

    opt_param  = numpy.load('opt_param.npy')
```

```
    zca_white  = numpy.load('zca_white.npy')
    mean_patch = numpy.load('mean_patch.npy')

    limit0 = 0
    limit1 = hidden_size * visible_size
    limit2 = 2 * hidden_size * visible_size
    limit3 = 2 * hidden_size * visible_size + hidden_size

    opt_W1 = opt_param[limit0 : limit1].reshape(hidden_size, visible_size)
    opt_b1 = opt_param[limit2 : limit3].reshape(hidden_size, 1)

    """ Visualize the learned optimal W1 weights """

    visualizeW1(numpy.dot(opt_W1, zca_white), vis_patch_side, hid_patch_side)

    """ Initialize Convolutional Neural Network model """

    network = ConvolutionalNeuralNetwork(opt_W1, opt_b1, zca_white, mean_patch,
vis_patch_side, pool_dim)

    step_size = 50

    train_images, train_labels = loadTrainingDataset()
    test_images, test_labels   = loadTestDataset()
    train_labels = train_labels - 1
    test_labels  = test_labels - 1

    softmax_train_data = getPooledFeatures(network, train_images, hidden_size, res_dim, step_size)
    softmax_test_data  = getPooledFeatures(network, test_images, hidden_size, res_dim, step_size)

    input_size     = hidden_size * res_dim * res_dim  # input vector size
    num_classes    = 4                          # number of classes
    lamda          = 0.0001                     # weight decay parameter
    max_iterations = 200                          # number of optimization iterations
```

```
    regressor = SoftmaxRegression(input_size, num_classes, lamda)

    opt_solution  = scipy.optimize.minimize(regressor.softmaxCost, regressor.theta,
                                    args = (softmax_train_data, train_labels,), method = 'L-BFGS-B',
                                    jac = True, options = {'maxiter': max_iterations})
    opt_theta     = opt_solution.x

    predictions = regressor.softmaxPredict(opt_theta, softmax_test_data)

    correct = test_labels[:, 0] == predictions[:, 0]
    print """Accuracy :""", numpy.mean(correct)

executeConvolutionalNeuralNetwork()
```

Выпускная квалификационная работа выполнена мной совершенно самостоятельно. Все использованные в работе материалы и концепции из опубликованной научной литературы и других источников имеют ссылки на них.

«19» июня 2018 г.

(подпись)

Кургенко П.Х.

(Ф.И.О.)