МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра Прикладной математики
(полное название кафедры)

УТВЕРЖДАЮ

Зав. кафедрой Соловейчик Ю.Г.
(фамилия, имя, отчество)

_____
(подпись)

«_____» _____ 2020 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Мазко Андрея Игоревича
(фамилия, имя, отчество студента – автора работы)

Разработка приложения компьютерного перевода жестовой речи в письменную с использованием технологии машинного обучения
(тема работы)

Факультет Прикладной математики и информатики
(полное название факультета)

Направление подготовки 01.03.02. Прикладная математика и информатика
(код и наименование направления подготовки бакалавра)

Руководитель от НГТУ	**Автор выпускной квалификационной работы**
Персова М.Г. (фамилия, имя, отчество)	Мазко А.И. (фамилия, И.О.)
д.т.н., профессор (ученая степень, ученое звание)	ФПМИ, ПМ-62 (факультет, группа)
(подпись, дата)	(подпись, дата)

Новосибирск 2020 г

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра Прикладной математики
(полное название кафедры)

УТВЕРЖДАЮ

Зав. кафедрой Соловейчик Ю.Г.
(фамилия, имя, отчество)

«17» марта 2020 г.

_____
(подпись)

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

студенту ***Мазко Андрею Игоревичу***
(фамилия, имя, отчество студента)

Направление подготовки *01.03.02. Прикладная математика и информатика*

Факультет Прикладной математики и информатики

Тема ***Разработка приложения компьютерного перевода жестовой речи в письменную с использованием технологии машинного обучения***

Исходные данные (или цель работы):

Цель работы – разработка приложения для перевода жестового дакатильного алфавита в письменный.

Исходные данные – изображения жестов.

Структурные части работы:

1. Обзор основных принципов работы искусственных нейронных сетей

2. Обзор особенностей при работе с нейронными сетями

3.Сверточные нейронные сети, описание принципа работы снс

4. Обзор готовых архитектур сверточных нейронных сетей

5.Исследования влияния различных параметров на работу нейронной сети

6.Подготовка данных к работе с нейронной сетью

7.Создание архитектуры и обучение модели

8.Создание приложения, осуществляющего перевод с жестового языка на письменный

Задание согласовано и принято к исполнению.

Руководитель от НГТУ	**Студент**
Персова М.Г..	Мазко А.И
(фамилия, имя, отчество)	(фамилия, имя, отчество)
д.т.н., профессор.	ФПМИ, ПМ-62
(ученая степень, ученое звание)	(факультет, группа)
17.03.2020 г.	17.03.2020 г.
(подпись, дата)	(подпись, дата)

Тема утверждена приказом по НГТУ № *1535/2* от «*17*» *марта 2020 г.*

ВКР сдана в ГЭК № _____, тема сверена с данными приказа

17 марта 2020 г.

(подпись секретаря экзаменационной комиссии по защите ВКР, дата)

Задорожный А.Г.

(фамилия, имя, отчество секретаря экзаменационной комиссии по защите ВКР)

АННОТАЦИЯ

Отчет 45 с., 22 рис., 2 табл., 19 источников, 1 приложение.

НЕЙРОННАЯ СЕТЬ, СВЕРТОЧНАЯ НЕЙРОННАЯ СЕТЬ, РАСПОЗНАВАНИЕ, ДАТАСЕТ

Объектом исследования является применение метода обучения с учителем сверточных нейронных сетей для решения задачи классификации изображений, с целью создания приложения перевода с жестового языка на письменный.

Цель работы - разработка приложения для распознавания жестов, обозначающих буквы английского алфавита.

В ходе работы были проведены исследования, позволяющие определить влияние гиперпараметров на работу нейросети, так же проведена предобработка данных, позволяющая расширить обучающую выборку. На основе результатов исследований была подобрана архитектура, позволяющая обучить точную модель классификатора. На базе обученной модели классификатора, было реализовано приложение, позволяющее производить перевод с жестового алфавита на письменный в реальном времени.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью работы является разработка приложения для перевода с жестового дактильного языка на текстовый с помощью технологии нейронных сетей.

Во время выполнения поставленной задачи необходимо:

1. Исследовать принципы машинного обучения.
2. Подготовить данные, подходящие для работы с нейросетью
3. Обучить модель, используя подготовленные данные
4. Провести исследования

В России по официальным данным насчитывается около 200 тысяч инвалидов и слабослышащих людей, а так же по оценке Всероссийского общества глухих проблемы со слухом имеются у 13 миллионов Россиян. В настоящее время у людей, владеющих жестовым языком и не способных разговаривать на речевых языках, наблюдаются большие проблемы с коммуникацией, так как жестовый язык почти не распространен среди тех, кто не страдает от проблем со слухом или речью. Переводчик с жестового языка мог бы пригодится многим людям как для общения, так и для изучения этого языка.

Основой для функционирования проекта является сверточная нейронная сеть. Для её оптимальной работы необходимо разработать или выбрать готовую оптимальную архитектуру. Дальше нужно обучить сеть на выбранном наборе данных.

Для работы с обученной нейронной сетью требуется создать приложение, способное

1. Считывать уже скомпилированную модель
2. Производить захват изображения с веб-камеры
3. Проводить анализ изображения

1. ВЫБОР СРЕДСТВ РАЗРАБОТКИ

Для решения поставленной задачи было решено использовать следующее программное обеспечение:

- Язык программирования Python 3.7
- Облачный сервис Google Colab
- Редактор и отладчик кода Visual Studio Code
- Библиотеки машинного обучения для Python

Обоснование выбора программного обеспечения:

Python[https://www.python.org/] – кроссплатформенный язык программирования, имеет лаконичный синтаксис, множество необходимых библиотек для решения поставленной задачи

Google colaboratory [https://colab.research.google.com] – облачный сервис, позволяющий запускать программы, используя большой объем RAM, поддерживает GPU. В данной работе используется для быстрого обучения нейронной сети.

TensorFlow [https://www.tensorflow.org] – основная используемая библиотека. Представляет набор инструментов для решения задач обучения и построения нейронных сетей.

OpenCV [https://opencv.org/] – библиотека, позволяющая работать с изображениями.

NumPy [https://numpy.org/] – предоставляет возможность удобной работы с многомерными матрицами.

Matplotlib [https://matplotlib.org/] – удобная библиотека для построения двумерных графиков, хорошо подходит для того, чтобы следить за процессами обучения.

2. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

2.1 КЛАССИФИКАЦИИ

Задача классификации – задача, в которой имеется определенное множество объектов, разделенное на классы. Также задано конечное множество объектов, для которых известна принадлежность к классам[4].

Целью задачи является построение алгоритма, способного классифицировать объект из исходного множества.

Класс – объединяет объекты в разные категории, исходя из определенных признаков.

Классифицировать объект - присвоить объекту номер класса, к которому он принадлежит.

Задачи классификации разделяют по типологии классов:

1. Бинарная классификация – наиболее простая задача классификации, сводится к определению объекта к одному из двух классов.
2. Многоклассовая классификация – задача, в которой число классов больше 2.
3. Задача с пересекающимися классами – объект может принадлежать нескольким классам одновременно.
4. Задача с непересекающимися классами – объект принадлежит только одному классу.

Например: имеется множество автомобилей, исходя из поставленной задачи, это множество можно разделить по классам, например, по маркам авто.

2.2 НЕЙРОННЫЕ СЕТИ

Нейронная сеть(искусственная) – математическая модель с программной реализацией, построенная по принципу биологических нейронных сетей

Представляет из себя совокупность искусственно созданных нейронов, каждый из которых получает на вход сигнал, который периодически передается другим нейронам. Объединяя такие простые нейроны в сложные сети, можно добиться успеха в решении сложных задач[4].

Чаще всего нейронные сети используются в следующих областях:

- Программирование
- Кибернетика
- Машинное обучение
- Математика

Нейрон – вычислительная единица, нейронной сети.

Нейроны делятся на следующие типы:

- Входные - получают данные
- Скрытые - обрабатывают данные
- Выходные - выдают данные

Синапс – связь между нейронами.

Слой – группа нейронов одного уровня, внутри слоя нейроны не связаны.

Чтобы работать с нейросетью ее для начала нужно обучить. Собственно, такая возможность нейронных сетей, как обучение – ее самое большое преимущество перед традиционными математическими алгоритмами.

Задачи, решаемые с помощью нейронной сети:

1. Классификация – распределение объектов по принадлежности к конкретному классу
2. Прогнозирование – предсказание результата, исходя из известных параметров.

3. Кластеризация – разбиение образов на группы, внутри которых должны оказаться схожие объекты, а объекты в разных групп должны быть более отличны.

2.2.1. Работа нейросети

Все данные, подающиеся на вход нейрону, имеют свой вес, который определяет значимость данных в данный момент.

При запуске работы нейронной сети, данные расставляются случайным образом. Входные данные умножаются на свои веса и суммируются. Дальше сумма проходит через функцию активации, которая формирует адекватный выходной результат.

2.2.2. Функция активации

Существует достаточно разных функций активации. Рассмотрим основные:

Линейная функция (рисунок 1 и формула 1) .

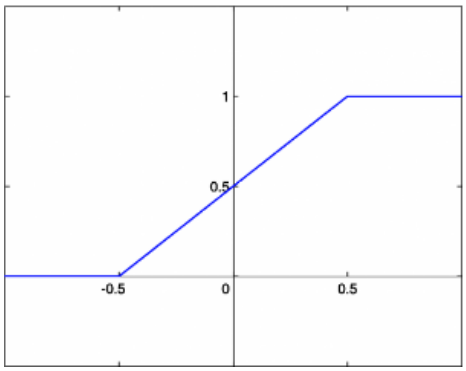

Рисунок 1 – Линейная функция активации

$$f(x) = x \quad (1)$$

Используя линейную функцию активации теряется возможность делать наборы из слоев, потому что вся сеть будет подобна слою с линейной функцией активации.

Данная функция подходит для тестирования работы сети.

Логистическая функция (рисунок2 и формула 2).

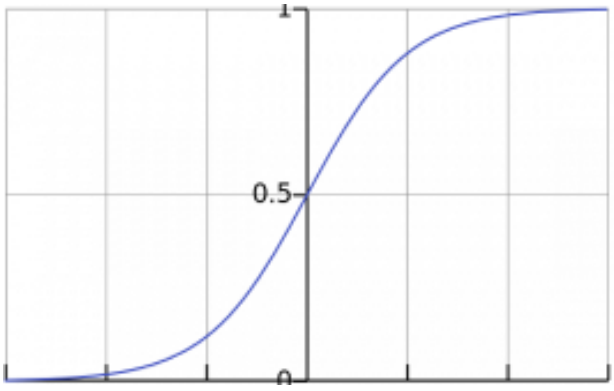

Рисунок 2 – Логистическая функция активации

$$f(x) = \frac{1}{1+e^{-x}} \tag{2}$$

Хорошо подходит для задач бинарной классификации, если использовать её на выходном слое, так как область значений сигмоиды – (0,1)

RuLu (рис3 и формула 3)

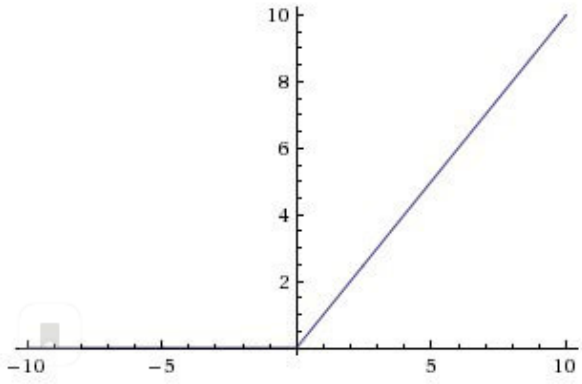

Рисунок 3 – ReLu

$$f(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases} \tag{3}$$

Одна из самых популярных функций активации, связано это с ее достоинствами - не ресурсоемкая, не подвергается перенасыщению.

Softmax(формула 4)

Логистическая функция, обобщенная для многомерного случая представлена ниже (формула 4).

$$f_i(\vec{x}) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}}, \text{для } i = 1, \ldots, J \quad (4)$$

2.2.3. Коррекция весов

Чтобы добиться нужной точности работы нейронной сети, необходима постоянная корректировка её весов. В данной работе использовался следующий метод корректировки:

Метод обратного распространения:

Один из самых популярных методов коррекции весов. Основная идея такова - на основе полученной ошибки, необходимо изменять веса нейронов в лучшую сторону, начиная с конца нейронной сети(выходных нейронов), заканчивая началом(выходными нейронами).

2.3. МЕТОДЫ ОБУЧЕНИЯ НЕЙРОННОЙ СЕТИ

Без учителя - суть данного метода заключается в поиске связей между входными объектами, на вход подается только некоторое множество без всяких пояснений к нему.

Чаще всего применяется для решения задач кластеризации.

С учителем – суть данного метода повторяет метод обучения без учителя, только с той разницей, что на вход так же подаются примеры – учителя, с помощью которых нейросеть должна выявить зависимости между объектами и классами, к которым они принадлежат.

Применяется для решения задач классификации и регрессии.[6]

2.4. ОПТИМИЗАТОРЫ НЕЙРОННОЙ СЕТИ

Во время обучения нашей модели, постоянно изменяются веса, чтобы наладить максимально точную работу, необходимо минимизировать функцию потерь, для этого подходят следующие оптимизаторы:

Стохастический градиентный спуск - минимизирует функцию, делая шаги в сторону убывания функции.

Рассматривается задача минимизации следующей функции (формула 5):

$$Q(\omega) = \frac{1}{n}\sum_{i=1}^{n} Q_i(\omega), \text{где } Q_i(\omega) \quad (5)$$

RMS Prop - Схож с стохастическим градиентным спуском, главное отличие данного метода в том, что на каждой итерации происходит вычисление производной для текущей подвыборки данных, которая возводится в квадрат.

Adam - Данных алгоритм является слиянием СГС и RMS Prop.

На каждом шаге вычисляется производная подвыборки, а затем считаются производные (формулы 6 – 7)[6].

$$V_{d\omega} = \beta_1 V_{d\omega} + (1 - \beta_1)\frac{\partial L}{\partial \omega} \quad (6)$$

$$V_{db} = \beta_1 V_{db} + (1 - \beta_1)\frac{\partial L}{\partial b} \quad (7)$$

Производится корректировка смещения, а затем вычисляется производная (8 – 11):

$$V_{d\omega} = \frac{V_{d\omega}}{1-\beta_1^t} \quad (8)$$

$$V_{db} = \frac{V_{db}}{1-\beta_1^t} \quad (9)$$

$$S_{d\omega} = \beta_2 S_{d\omega} + (1 - \beta_2)\left(\frac{\partial L}{\partial \omega}\right)^2 \quad (10)$$

$$S_{db} = \beta_2 S_{db} + (1 - \beta_2)\left(\frac{\partial L}{\partial b}\right)^2 \quad (11)$$

Так же производится корректировка смещения (формулы 12 – 13):

$$S_{d\omega} = \frac{S_{d\omega}}{1-\beta_2^t} \quad (12)$$

$$S_{db} = \frac{S_{db}}{1-\beta_2^t} \quad (13)$$

В конце выполняется вычисление новых переменных (формулы 14 – 15):

$$\omega := \omega - \alpha \frac{V_{d\omega}}{\sqrt{S_{d\omega}} + \varepsilon} \quad (14)$$

$$b := b - \alpha \frac{V_{db}}{\sqrt{S_{db}} + \varepsilon} \quad (15)$$

2.5. ГИПЕРПАРАМЕТРЫ НЕЙРОННОЙ СЕТИ

В области машинного обучения параметры, которые устанавливаются, непосредственно, перед запуском самого процесса обучения и не изменяются в ходе процесса называют гиперпараметрами.

Программист сам подбирает нужные значения этих параметров. Если результат обучения не отвечает нужным требованиям, то, зачастую, добиться нужной точности помогает подбор гиперпараметров.

К гиперпараметрам можно отнести:

- Количество скрытых слоев
- Количество нейронов в слое
- Число эпох обучения
- Скорость обучения
- Выбор функции активации
- Выбор оптимизатора

Подбор наилучшего сочетания данных параметров приводит к эффективной работе нейронной сети. Данный процесс, как можно было понять, является экспериментальным, поэтому умение подбирать гиперпараметры, напрямую связано с опытом работы в сфере машинного обучения[4].

2.6. ОЦЕНКА РАБОТЫ НЕЙРОННОЙ СЕТИ

Функцию от пары значений ошибки и эпохи обучения называют сходимостью. Если отследить график сходимости, можно дать оценку работе нейронной сети. Если, с увеличением эпохи, ошибка уменьшается и не образует

“зигзагов” на график, то результат работы нейронной сети считается хорошим (рисунок 4).

Рисунок 4 – Обучение нейронной сети при разных параметрах

Чаще всего сеть перестает сходиться из-за неверных параметров или же по причине переобучения.

Переобучение – состояние, в котором сеть не учится, а “запоминает” идущие на вход ответы. Связано это с тем, что в обучающую выборку попадают однотипные данные. Работа с переобученной сетью не считается эффективной, так как она сможет адекватно распознавать только те примеры, которые входили в обучающую выборку. Для решения этой проблемы придумано достаточное количество методов, например:

- Корректировка обучающей выборки
- Регуляризация весов
- Нормализация подвыборки

2.7. СВЕРТОЧНЫЕ НЕЙРОННЫЕ СЕТИ

Яном Лекуном в 1998 году была предложена специальная архитектура для нейронных сетей - сверточная нейронная сеть (снс). За основу были взяты особенности зрительной коры, в которой открыли простые и сложные клетки, помогающие человеку определять образы. Архитектура СНС так же нацелена на эффективное распознавание образов.

В ее основе лежат чередующиеся сверточные слои и слои подвыборки, имитирующие простые и сложные клетки в зрительной коре[11].

2.8. АРХИТЕКТУРА СВЕРТОЧНОЙ НЕЙРОННОЙ СЕТИ

Сверточный слой – набор матриц (карт признаков), у каждой матрицы есть синаптическое ядро. Ядро – это фильтр, который проходит по всей предыдущей карте и выделяет необходимые признаки. Подбор размера ядра также является гиперпараметром.

Если будет выбран слишком большой размер ядра, то велика возможность ошибки из-за большого количества связи, а если выбрать слишком маленький размер, то оно не сможет нормально функционировать и выбирать нужные признаки (рисунок5)[3].

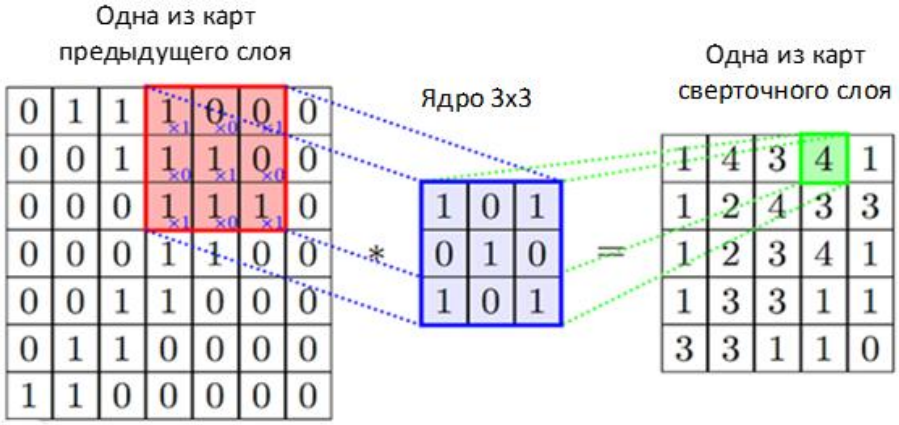

Рисунок 5 – Операция свертки

Слой подвыборки - задачей слоя подвыборки является снижение пространственных размеров входного представления, что приводит к снижению количества используемых параметров и сложности вычислений в самой сети, кроме этого слой подвыборки позволяет контролировать переобучение модели.

Один нейрон подвыборочного слоя связан с несколькими нейронами сверточного, значение этого нейрона вычисляется следующими методами:

1. Усреднение
2. Выбор максимального значения (рисунок 6)[3].

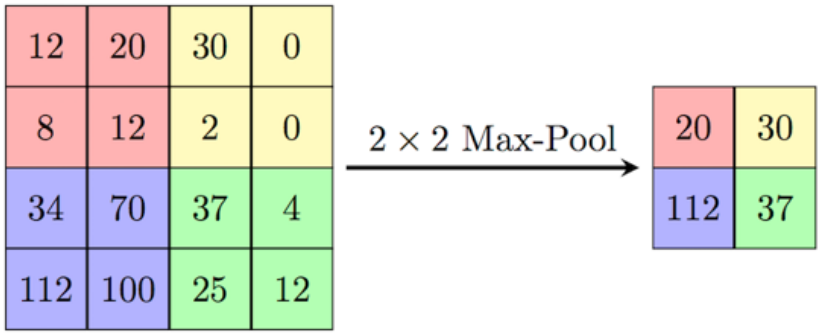

Рисунок 6 – Формирование карты подвыборки выбором максимального значения

От количества сочетаний слоев свертки и слоев подвыборки зависит сложность распознаваемых образов. Увеличить количество карт признаков можно при помощи уменьшения размерности подвыборки.

Полносвязный слой – Преобразует матричные данные в одномерный формат.

Выходной слой - отвечает за конечную классификацию. Число нейронов в данном классе должно быть равно числу классов данных.

Также следует добавить, что СНС поддерживают использование GPU, что значительно ускоряет процесс обучения[13].

2.9. ГОТОВЫЕ АРХИТЕКТУРЫ СВЕРТОЧНЫХ НЕЙРОННЫХ СЕТЕЙ

В настоящее время разработано большое количество архитектур СНС, подходящих для решения различных задач, рассмотрим некоторые из них:

2.9.1. LeNet

Данная архитектура была предложена еще в 1998 году. Включает в себя: 2 сверточных слоя, 2 слоя подвыборки, 2 полносвязных слоя и выходной слой с гауссовским соединением (рисунок 7)[10].

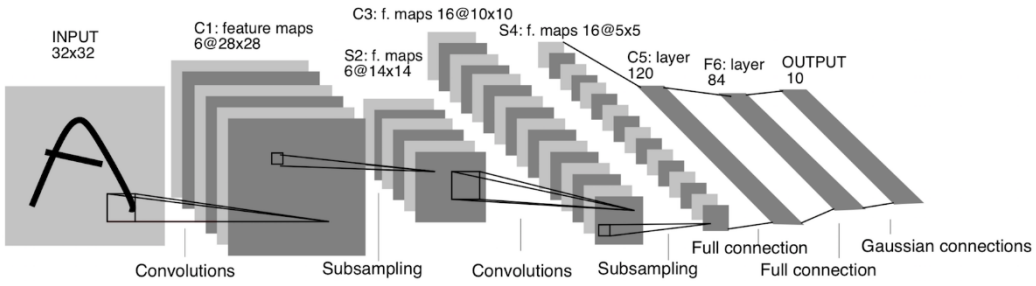

Рисунок 7 –LeNet

2.9.2. AlexNet

В 2012 году была предложена более глубокая и широкая архитектура СНС, по сравнению с существовавшими до нее популярными архитектурами. Название – AlexNet дано в честь одного из основных разработчиков данной архитектуры – Алекса Крижевского (рисунок 8)[12].

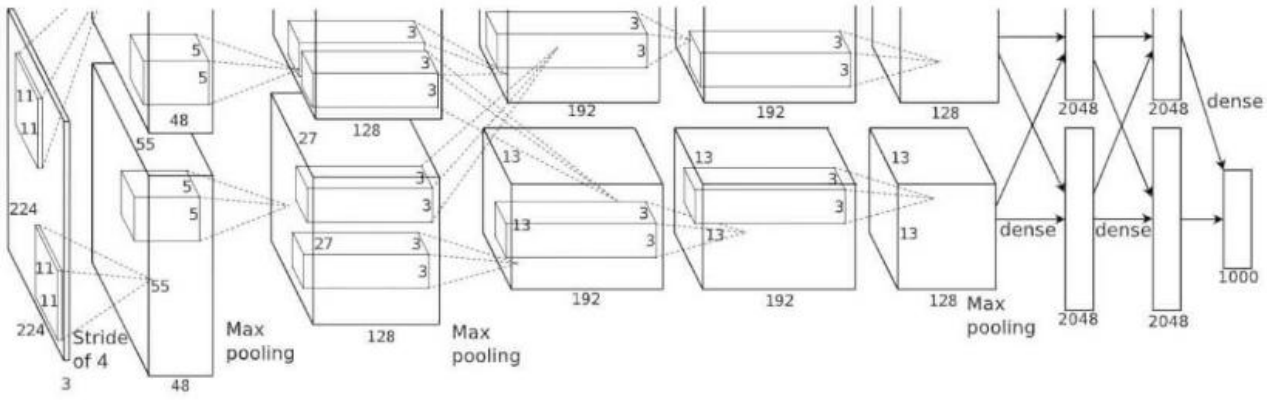

Рисунок 8 – Архитектура AlexNet

Первый сверточный слой данной нейронной сети выполняет свертку и пулинг с нормализацией отклика (LRN), где используются 96 различных фильтров размером 11×11.Формируется карта подвыборки с 3×3 фильтрами и с шагом, размер которого равен 2. Те же операции выполняются на втором уровне с фильтрами 5×5. Фильтры

3×3 используются в третьем, четвертом и пятом сверточных слоях. Дальше идут два полносвязных слоя, используется новая функция отсева (dropout), за всем эти идет последний слой, имеющий функцию активации soft-max.

Данная архитектура хорошо зарекомендовала себя для решения задач визуального распознавания объектов и для задач классификации изображений.

Так же с помощью данной архитектуры в 2012 году было выиграно соревнование ImageNet Large Scale Visual Recognition Challenge[11].

2.9.3. ResNet

ResNet входит в класс ультра-глубоких сетей, так как разработана с большим количеством слоев: 34, 50, 101, 152 и даже 1202.

Получила большее распространение архитектура с 50 слоями, которая называется соответственно ResNet50 содержит 49 слоев свертки и 1 полносвязный слой на выходе.

Основная особенность ResNet – Остаточный блок. Он представляет собой несколько слоев свертки с функциями активации, которые преобразуют выходной сигнал x в $F(x)$. В результате такой связи данный блок учит, как выходной сигнал отличается от выходного (рисунок 9)[11].

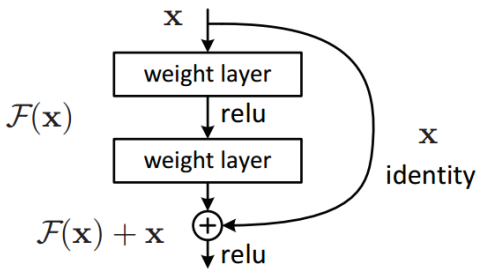

Рисунок 9 – Остаточный блок

3. ИССЛЕДОВАНИЯ

3.1. ИССЛЕДОВАНИЕ ЗАВИСИМОСТИ ТОЧНОСТИ КЛАССИФИКАЦИИ ОТ ВЫБОРА ОПТИМИЗАТОРА

Данное исследование было проведено для выбора лучшего оптимизатора. В исследовании участвовали следующие оптимизаторы : SGD, Adam, Adadelta. Исследование проведено на нейронной сети с следующими параметрами: количество ядер – 64, количество эпох – 10, количество сверточных слоев – 2. Результаты данного исследования приведены в таблице 1.

На графиках (рисунок 10) видно, что быстрее всего сходится нейронная сеть, использующая оптимизатор Adam. Худший результат показал оптимизатор Adadelta.

Таблица 1 – Результат исследования сетей с разными оптимизаторами

Оптимизатор	SGD	Adam	Adadelta
Точность	0.76	0.989	0.053
Ошибка	0.23	0.034	3.352

Из проведенного исследования можно сделать вывод, что лучший результат работы нейросети достигается при использовании оптимизатора Adam.

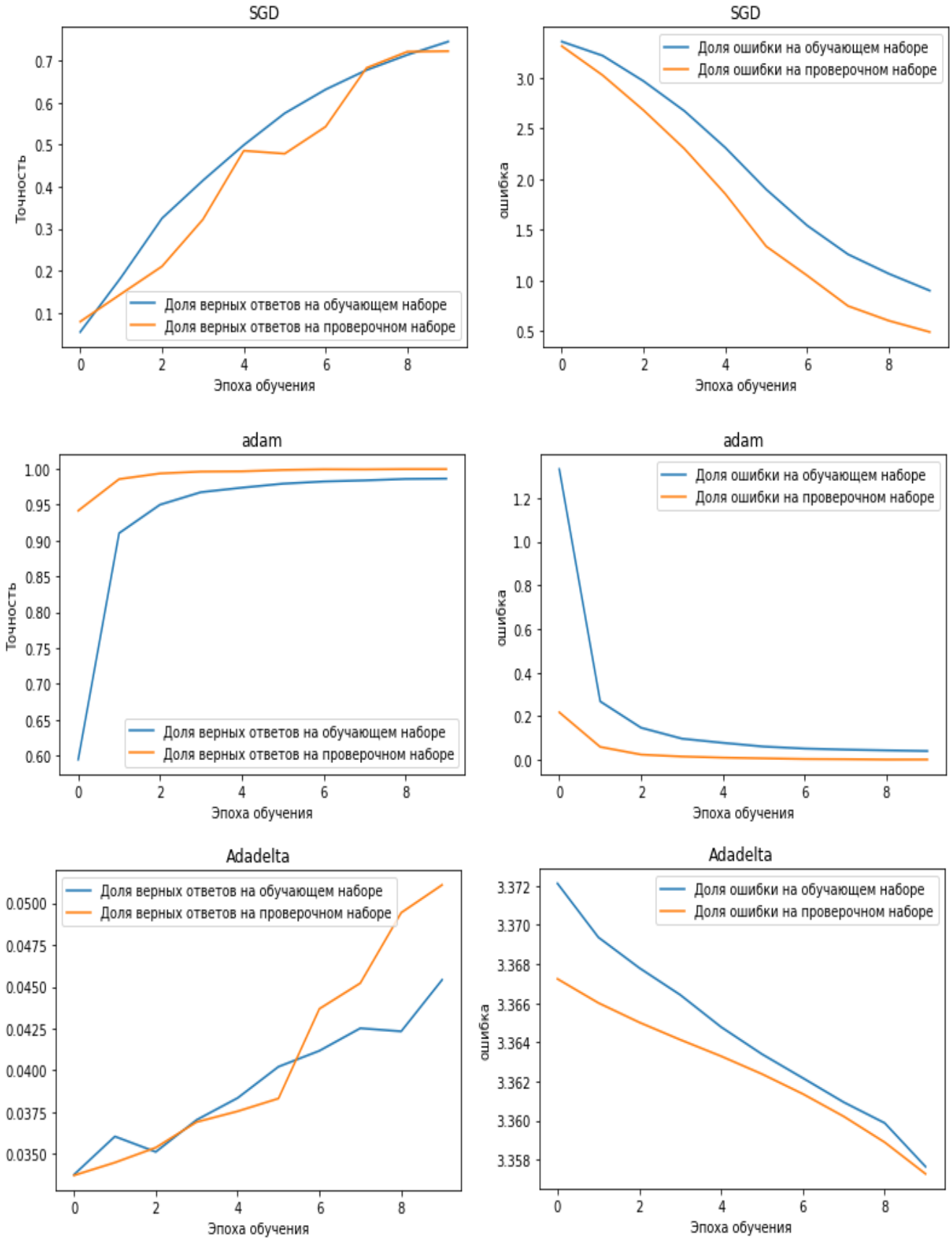

Рисунок 10 – Ошибка и точность сетей с разными оптимизаторами

3.2. ИССЛЕДОВАНИЕ ВЛИЯНИЯ КОЛИЧЕСТВА СВЕРТОЧНЫХ СЛОЕВ НА КАЧЕСТВО РАБОТЫ НЕЙРОННОЙ СЕТИ

Суть данного исследования заключалась в выборе оптимального количества сверточных слоев в нейронной сети. Исследование проведено на нейронной сети с следующими параметрами: количество ядер – 64, количество эпох –, оптимизатор – Adam.

Как можно заметить на рисунке 11, разница в результатах не критична, но самый оптимальный результат выдает нейронная сеть, в архитектуру которой включено 2 сверхточных слоя.

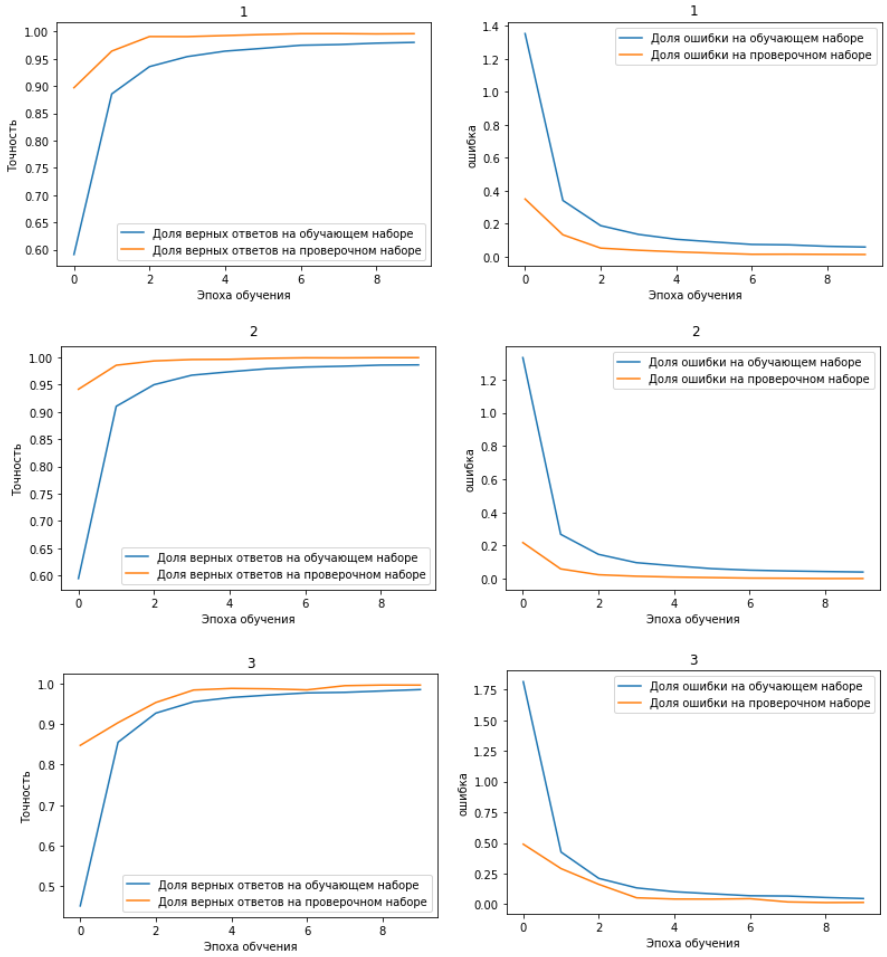

Рисунок 11 – Зависимости точности обучения от количества слоев свертки

3.3. ИССЛЕДОВАНИЕ ВЛИЯНИЯ ЗНАЧЕНИЯ СКОРОСТИ ОБУЧЕНИЯ НА РЕЗУЛЬТАТ РАБОТЫ НЕЙРОННОЙ СЕТИ

Скорость обучения – это такой параметр, который отвечает за величину коррекции весов на каждой эпохе обучения. Значения данного параметра варьируются от 0 до 1.

Например, в алгоритме обратного распространения ошибки (формула 16)

$$\Delta\omega = -\mu \frac{dE}{d\omega} \quad (16)$$

μ - коэффициент скорости обучения.

Было проведено исследование, в котором изменялась скорость обучения нейронной сети

Анализируя полученные данные на рисунке 12, можно сказать, что нейронная сеть не обучается, если скорость обучения задается равной 0.01.

При значениях 0.001 и 0.0001 нейронная сеть обучается почти с одинаковой точностью, но время обучения при меньшем значении коэффициента скорости обучения значительно увеличивается (табл. 2).

Таблица 2 – Результат исследования сетей с разной скоростью обучения

Скорость обучения	0.01	0.001	0.0001
Точность	0.03	0.989	0.976
Ошибка	3.3	0.034	0.038
Время обучения	18с / эпоха	25с / эпоха	51с / эпоха

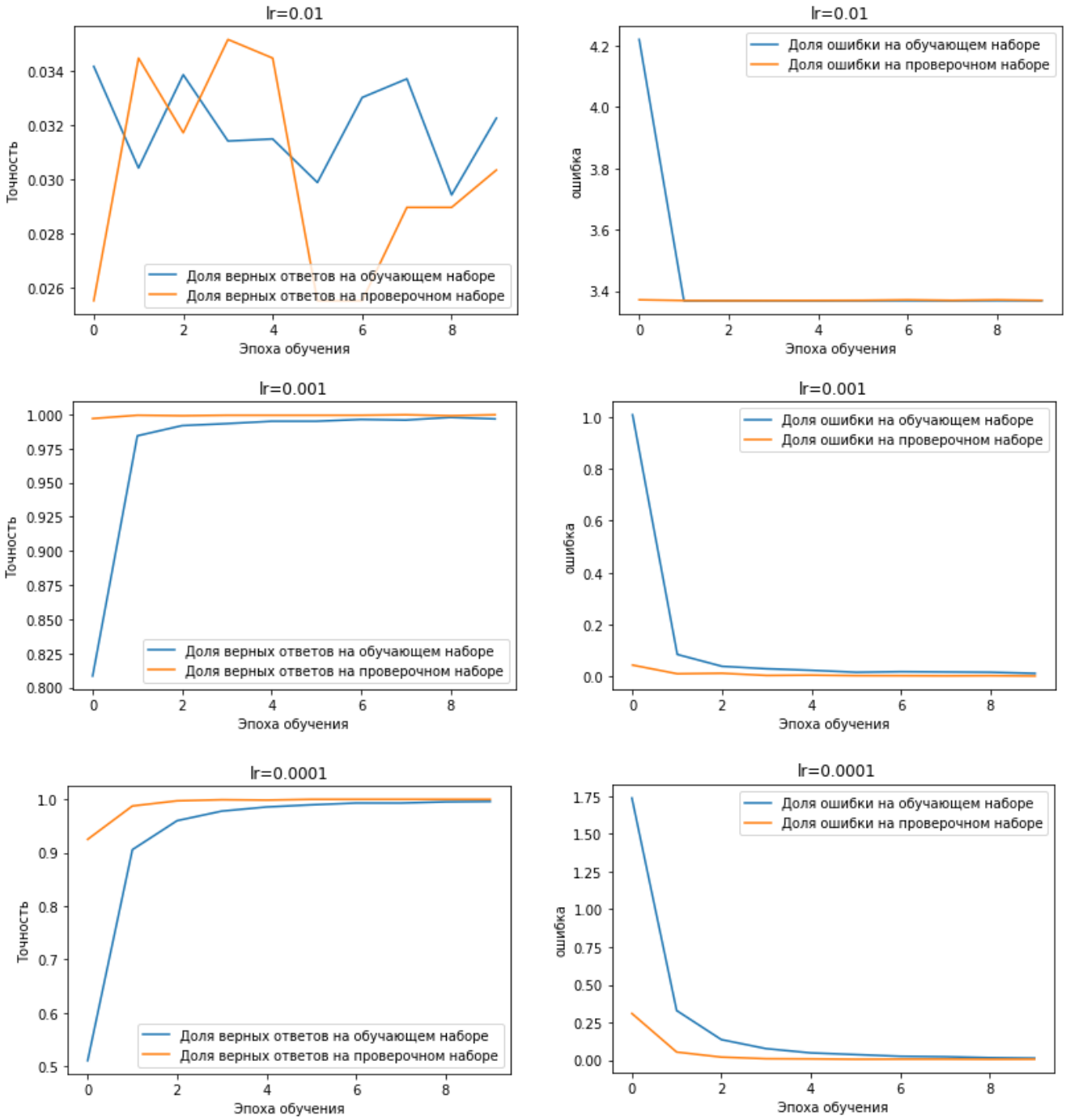

Рисунок 12 – Графики зависимости точности обучения от скорости обучения

4. ПРАКТИЧЕСКАЯ ЧАСТЬ

4.1. ПОДБОР ДАННЫХ

В качестве обучающих и тестовых данные был выбран набор данных ASL Alphabet с платформы Kaggle. ASL Alphabet включает в себя набор из 87000 изображений 200x200 пикселей.

Состоит из 29 классов, 26 из которых английские буквы A-Z и 3 класса SPACE, DELETE и NOTHING.

Основное преимущество этого набора данных в том, что в нем содержаться снимки с разной освещенностью, что в последующем помогает работать с входными данными разного качества (рисунок 14).

Рисунок 13 – Пример буквы «B» из набора ASL Alphabet

Также был отснят собственный набор данных, который включает в себя набор из 29000 изображений 64x64 пикселей (рисунок 13).

Состоит из такого же набора классов, как и набор данных ASL Alphabet.

Рисунок 14 – Пример буквы «B» из собственного набора

4.2. АУГМЕНТАЦИЯ ДАННЫХ

Аугментация данных – это методика создания дополнительного обучающего набора данных из уже имеющихся данных.

Так как для лучших результатов необходимы большие объемы данных, было решено дополнить обучающую выборку новыми данными при помощи аугментации.

Суть данного метода заключается в копировании и изменении исходных данных.

Изменять данные можно следующим образом:

- Изменение цвета
- Отражение по горизонтали
- Отражение по вертикали
- Добавление шума
- Замена фона
- Повороты
- Сжатие и растяжение и т.д.

Такие методы как повороты не подходят для успешного решения данной задачи, потому что из-за угла наклона руки может измениться и значение показываемого жеста. Пример такого поворота представлен на рисунке 16.

Рисунок 15 – Буква «U»(слева) и «H»

В результате аугментации мы имеем в своем распоряжении дополнительный набор данных, включающий в себя подобные изображения (рисунок 16).

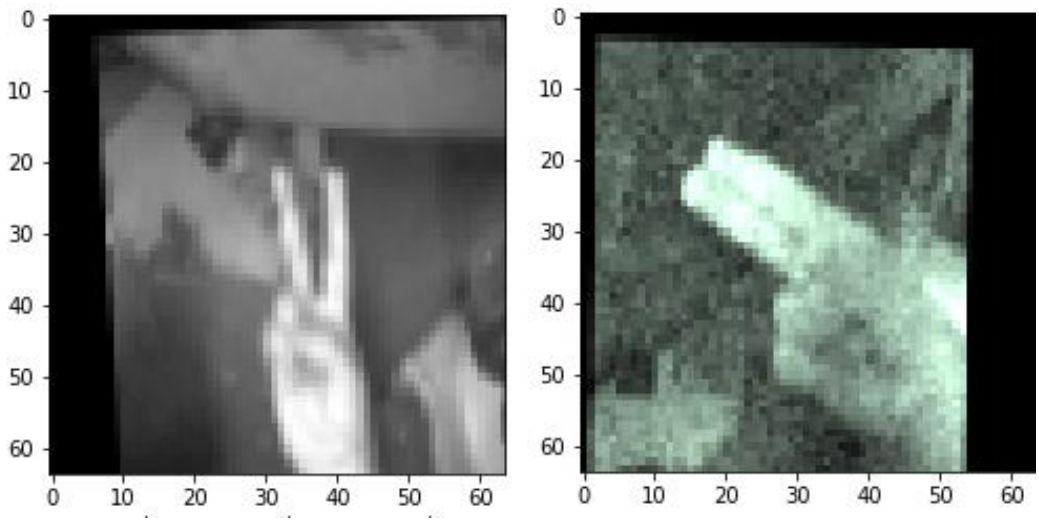

Рисунок 16 – Изображения из аугментированного набора

4.3. ПРЕДОБРАБОТКА ДАННЫХ

После того, как была получена достаточно большая выборка обучающих изображений, необходимо перевести изображения в формат, подходящий для обработки нейросетью.

Для этого требуется провести несколько действий:

1. Считать изображение с помощью метода *imread* библиотеки *OpenCV*.

Данный метод возвращает трехмерный массив 3*ширина*длина, каждый элемент данного массива отвечает за интенсивность красного, зеленого или же синего канала, значения варьируются от 0 до 255.

2. Если изображение было взято из набора *ASL Alphabet*, необходимо изменить длину и ширину(сделать их равными 64) данного изображения с помощью метода *resize* библиотеки *OpenCV*.

3. Дальше необходимо провести нормализацию данных – разделить каждый элемент массива на 255, чтобы все значения входили в интервал от 0 до 1.

4.4. ВЫБОР АРХИТЕКТУРЫ

Было принято решение отказаться от готовой архитектуры и разработать собственную, подходящую для решения поставленной задачи. Выбор архитектуры проводился экспериментальным способом.

Эксперимент заключался в следующем: проводились изменения параметров нейронной сети и отслеживалась точность работы нейронной сети.

В заключении работы над экспериментом было решено остановиться на следующей архитектуре:

- 4 сверточных слоя
- 2 слоя подвыборки
- 2 полносвязных слоя
- 2 слоя регуляризации

Схематически, данная архитектура представлена на рисунке 17.

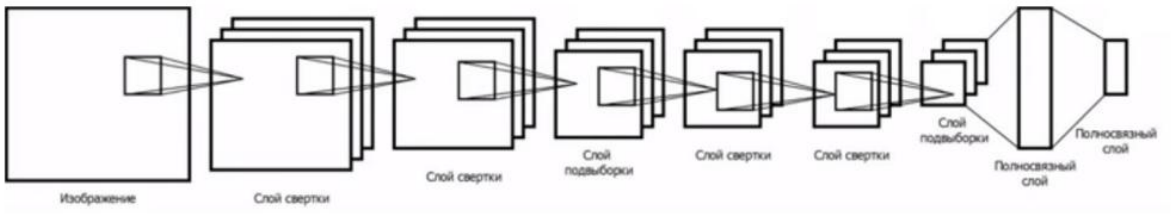

Рисунок 17 – Выбранная архитектура нейронной сети

4.5. ОБУЧЕНИЕ НЕЙРОННОЙ СЕТИ

Как только была выбрана архитектура нейронной сети, следующей задачей стало обучение.

В данном случае используются изображения, разрешение которых приводится к 64x64. Обучение проходило на облачном сервисе Google Colab для быстрого его завершения.

Обучение проходило на модуле Keras из библиотеки TensorFlow, главной задачей являлось достижение минимального значения ошибки.

В результате появилась сеть, способная распознавать изображения жестов.

4.6. ОПИСАНИЕ РАБОТЫ ПРОГРАММЫ

Для функционирования приложения импортируем обученную сеть в формате H5.

H5 – формат, поддерживающий хранение большого объема цифровой информации. Хорошо подходит для сохранения обученных нейросетей.

Для работы с нейросетью были разработаны следующие методы:

1. Test_image

На вход этого метода подается изображение, которое переводится в разрешение 64x64.

Следующим действием является перевод изображения в трехмерную матрицу, значениями которой является интенсивность пикселей в разных каналах изображения.

2. Predict

На вход поступает трехмерная матрица интенсивности пикселей.

Дальше с помощью встроенного метода модели predict выполняется анализ принадлежности изображения к одному из 29 исходных классов.

Для использования программы был создан интерфейс с помощью PyQt (рисунок18).

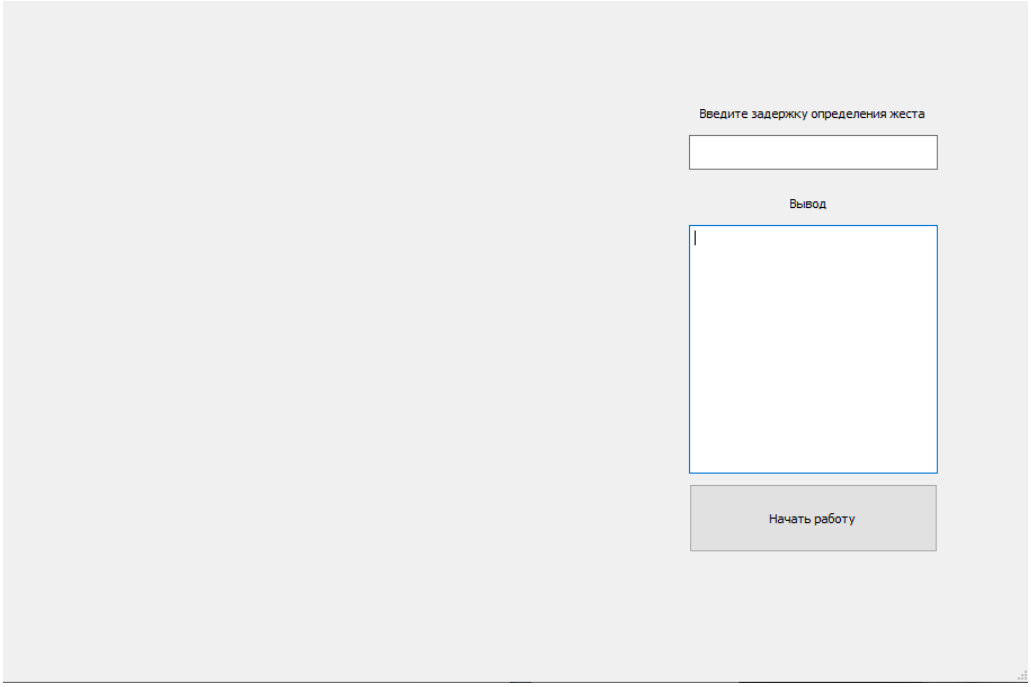

Рисунок 18 – Приложение

При нажатии на кнопку “начать работу” запускается новое окно, на котором выводится поток с видеокамеры (рисунок 19).

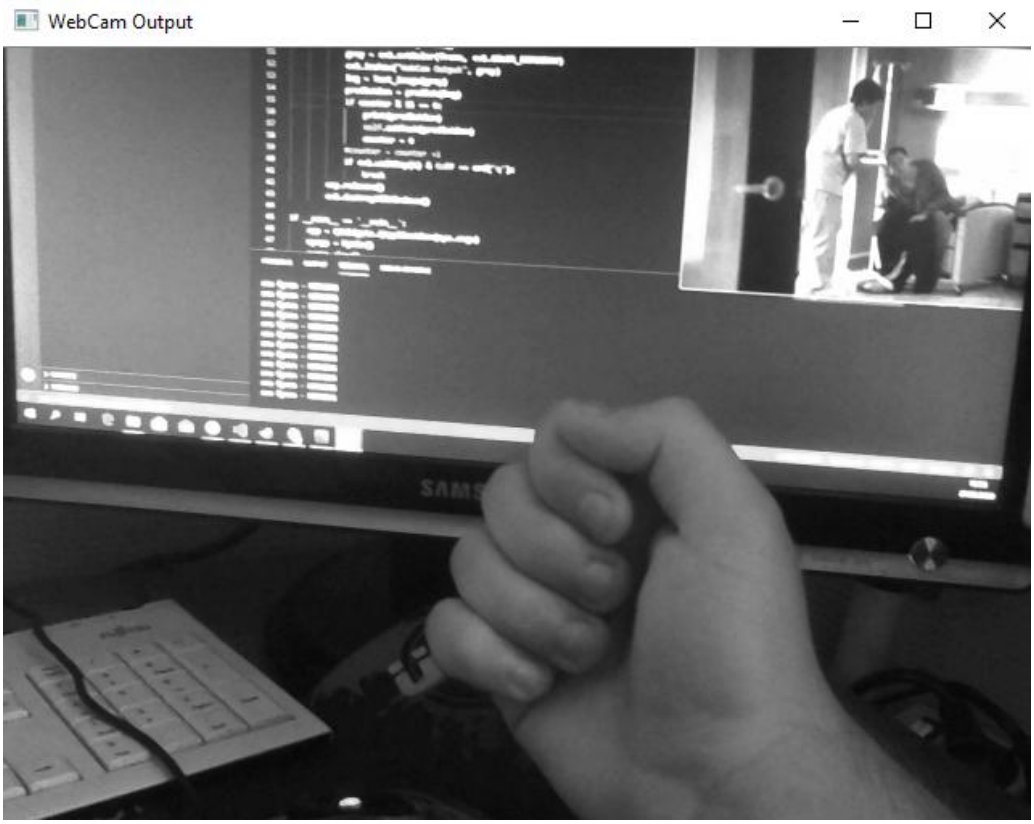

Рисунок 19 – Вывод с видеокамеры

Если показывать жесты, то в соответствующем окне будет выводиться результат. Пример работы программы показан на следующих рисунках.

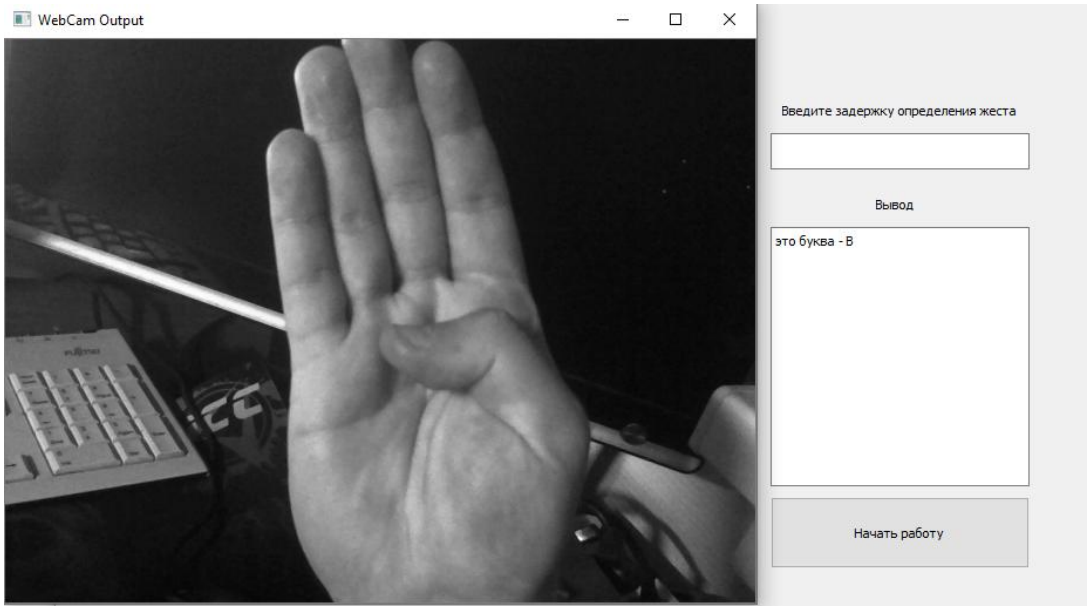

Рисунок 20 – Пример работы программы

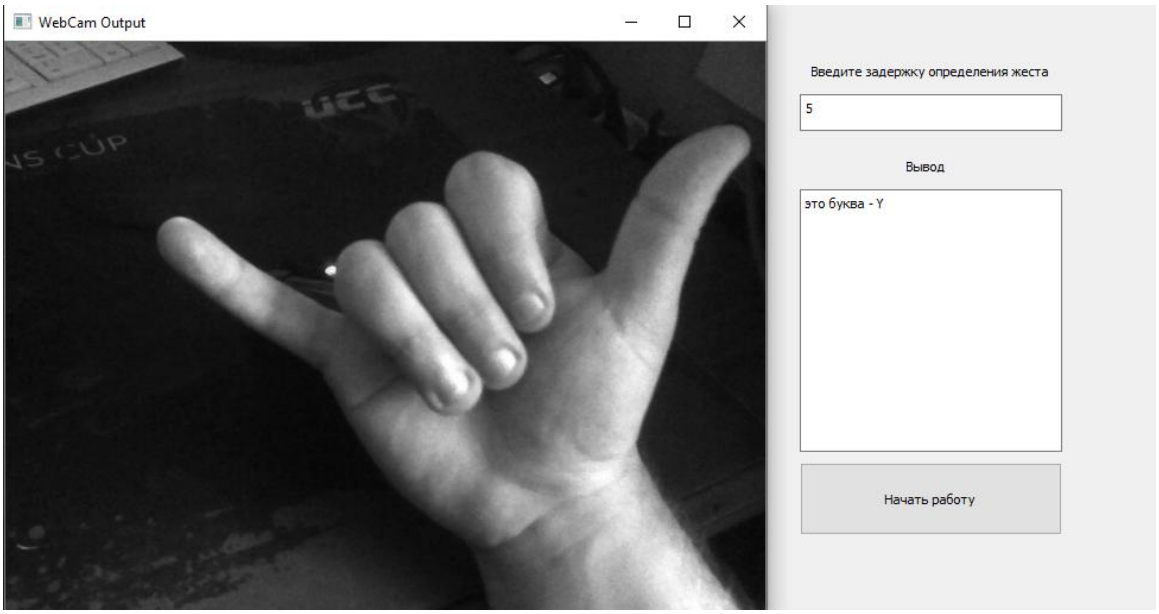

Рисунок 21 – Пример работы программы

Рисунок 22 – Пример работы программы

Файловая иерархия приложения

1. design.py - содержит в себе информацию о интерфейсе программы.
2. main.py - файл, содержащий в себе методы обработки интерфейса
3. webcam.py - Содержит методы обработки изображения.

Для работы приложения необходимо установить:

1. Python 3.7
2. Библиотеки: Tensorflow, matplotlib, opencv, skimage

Рекомендуемые системные требования:

1. ОС: Windows 7 или новее
2. Оперативная память: 400 MB ОЗУ
3. Процессор: Inter core i3-4170
4. Камера, подключаемая к ПК

4.7. ВХОДНЫЕ И ВЫХОДНЫЕ ДАННЫЕ

Входными данными является видео, передающееся с веб-камеры. Выходными данными является текст, описывающий жест, показанный на камеру.

4.8. ВЫЗОВ И ЗАГРУЗКА

Для запуска данного приложения необходимо перейти в директорию с файлом main.py и выполнить команду «python main.py»

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной бакалаврской работы было разработано программное обеспечение для компьютерного перевода жестов дактильного алфавита на письменный язык.

Так же были разобраны и решены следующие задачи:

- Проведен экскурс в искусственные нейронные сети
- Проведен разбор методов работы нейронной сети
- Были выбраны средства для решения поставленной задачи
- Разобрана готовая база изображений жестов
- Был отснят собственный набор данных
- Была выбрана подходящая архитектура нейронной сети
- Была обучена и протестирована нейронная сеть
- Было разработано приложение, позволяющее работать с получившейся нейронной сетью
- Проведено тестирование приложения

Созданное приложение может упрощать процесс общения, если одна из сторон не знает дактильный жестовый язык, так же данное приложение может выступать как тренажер в освоении той же жестовой азбуки. Приложение не является ресурсозатратным и может быть запущено даже на слабых машинах.

СПИСОК ЛИТЕРАТУРЫ

1. Функция активации [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Функция_активации (дата обращения: 13.04.2020).

2. Метод обратного распространения ошибки [Электронный ресурс]. – Режим доступа:ru.wikipedia.org/wiki/Метод_обратного_распространения (дата обращения: 19.04.2020).

3. Сверточная нейронная сеть, часть 1 [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/348000/ (дата обращения: 19.05.2020).

4. Шолле Ф. Глубокое обучение на Python / Ф. Шолле // – СПб. : Питер – 2018 – 400 с.

5. Выбор моделей и критерии качества [Электронный ресурс]. – Режим доступа: http://www.machinelearning.ru/wiki/images/1/1c/Sem06_metrics.pdf (дата обращения: 03.06.2020).

6. Николенко С. Глубокое обучение / С. Николенко, А. Калдурин, Е. Архангельская // – СПб. : Питер – 2019 – 480 с.

7. Этем Алпайдин / Машинное обучение: новый искусственный интеллект / Пер. с англ. – Издательская группа Точка, 2017 – 232 с

8. Машинное обучение [Электронный ресурс] / Wikipedia. – URL. https://ru.wikipedia.org/wiki/Машинное_обучение (дата обращения 09.04.2020)

9. LeNet-5 [Электронный ресурс] / medium. – URL. https://medium.com/@congyuzhou/lenet-5-своими-руками-b60ae3727cd3 (дата обращения 21.04.2020)

10. Компьютерное зрение. Лекция для Малого ШАДа Яндекса [Электронный ресурс] / Habr. – URL. https://habr.com/company/yandex/blog/203136/ (дата обращения 27.04.2020)

11. Архитектуры нейросетей [Электронный ресурс] / Habr. – URL. https://habr.com/ru/company/nix/blog/430524/ (дата обращения 28.04.2020)

12. AlexNet [Электронный ресурс] / neurohive. – URL. https://neurohive.io/ru/vidy-nejrosetej/alexnet-svjortochnaja-nejronnaja-set-dlja-raspoznavanija-izobrazhenij/ (дата обращения 30.05.2020)

13. Распознавание образов [Электронный ресурс] / Искусственный интеллект Ai. – URL. https://intellect.ml/svertochnaya-nejronnaya-setconvolutional-neural-network-cnn-6013 (дата обращения 17.04.2020)

14. Применение нейронных сетей в задачах распознавания образов / Зеленков. – it-claim: Lection_text, 2000. – 17 стр

15. Обзор свёрточных нейронных сетей для задачи классификации изображений / О. С. Сикорский. – cyberleninka: obzor-svyortochnyhneyronnyh.. , 2018. – 8 стр .

16. Набор данных ASL Alphabet [Электронный доступ]. – Режим доступа: https://www.kaggle.com/grassknoted/asl-alphabet (дата обращения: 03.04.2020).

17. Глубокое обучение для новичков: распознаем изображения с помощью сверточных сетей [Электронный ресурс] / Habr. – URL. https://habr.com/company/wunderfund/blog/314872/ (дата обращения 24.04.2020)

18. ImageNet Classification with Deep Convolutional Neural Networks / Alex Krizhevsky [et al]. – NIPS: 4824, 2012. – 9 p

19. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift / Sergey Ioffe [et al]. – arXiv:1502.03167, 2015. – 11 p.

ПРИЛОЖЕНИЕ

Текст программы

```
import zipfile
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation,Fla
tten, Conv2D,MaxPooling2D

from tensorflow.keras import utils
import os
import os, cv2, skimage
from skimage.transform import resize
from sklearn.model_selection import train_test_split
import tensorflow as tf

from keras.preprocessing.image import ImageDataGenerator
from keras.utils.np_utils import to_categorical

from keras.models import Model
from keras.callbacks import TensorBoard, ModelCheckpoint, ReduceLR
OnPlateau


from google.colab import files
!mkdir ~/.kaggle
!mv kaggle.json ~/.kaggle
!kaggle datasets download -d grassknoted/asl-alphabet
z = zipfile.ZipFile('/content/asl-alphabet.zip', 'r')
z.extractall()
with zipfile.ZipFile("/content/drive/My Drive/Dataset/dataset.zip"
,"r") as zip_ref:
  zip_ref.extractall()
import string
Alphabet=[x for x in list(string.ascii_uppercase)]
Path_to_200_train = "/content/asl_alphabet_train/asl_alphabet_trai
n/"
batch_size = 128
imageSize = 64
target_dims = (imageSize, imageSize, 3)
num_classes = 29

#train_len = 87000
train_len = 21212
train_dir = path_to_my_set

def get_data(folder):
```

```
    """
    Load the data and labels from the given folder.
    """
    X = np.empty((train_len, imageSize, imageSize, 3), dtype=np.fl
oat32)
    y = np.empty((train_len,), dtype=np.int)
    cnt = 0

    for folderName in os.listdir(folder):
        if not folderName.startswith('.'):
            if folderName in ['A']:
                label = 0
            elif folderName in ['B']:
                label = 1
            elif folderName in ['C']:
                label = 2
            elif folderName in ['D']:
                label = 3
            elif folderName in ['E']:
                label = 4
            elif folderName in ['F']:
                label = 5
            elif folderName in ['G']:
                label = 6
            elif folderName in ['H']:
                label = 7
            elif folderName in ['I']:
                label = 8
            elif folderName in ['J']:
                label = 9
            elif folderName in ['K']:
                label = 10
            elif folderName in ['L']:
                label = 11
            elif folderName in ['M']:
                label = 12
            elif folderName in ['N']:
                label = 13
            elif folderName in ['O']:
                label = 14
            elif folderName in ['P']:
                label = 15
            elif folderName in ['Q']:
                label = 16
            elif folderName in ['R']:
                label = 17
            elif folderName in ['S']:
                label = 18
            elif folderName in ['T']:
                label = 19
            elif folderName in ['U']:
```

```
                label = 20
            elif folderName in ['V']:
                label = 21
            elif folderName in ['W']:
                label = 22
            elif folderName in ['X']:
                label = 23
            elif folderName in ['Y']:
                label = 24
            elif folderName in ['Z']:
                label = 25
            elif folderName in ['del']:
                label = 26
            elif folderName in ['nothing']:
                label = 27
            elif folderName in ['space']:
                label = 28
            else:
                label = 29
            for image_filename in os.listdir(folder + folderName):
                img_file = cv2.imread(folder + folderName + '/' +
image_filename)
                if img_file is not None:
                    img_file = skimage.transform.resize(img_file,
(imageSize, imageSize, 3))
                    img_arr = np.asarray(img_file).reshape((-
1, imageSize, imageSize,3))

                    X[cnt] = img_arr
                    y[cnt] = label
                    cnt += 1


    return X,y

X_train,y_train = get_data(Path_to_200_train)
X_train, X_test, y_train, y_test = train_test_split(X_train, y_tra
in, test_size=0.1)


y_trainHot = to_categorical(y_train, num_classes=num_classes)
y_testHot = to_categorical(y_test, num_classes=num_classes)
def save_full_data(x,y):
  path = "/content/drive/My Drive/datafiles/"
  num =1
  shag = int(x.shape[0]/10)
  srez =0
  srez1 = 0
  for i in range(10):
    srez +=shag
    newx = x[srez1:srez]
```

```
    newy = y[srez1:srez]
    np.save(path + str(i) +"X64", newx)
    np.save(path + str(i) +"Y64", newy)
    srez1 +=shag
def load_all_data():
  path = "/content/drive/My Drive/datafiles/"
  zX = np.load(path+str(0)+"X64.npy")
  zY = np.load(path+str(0)+"Y64.npy")
  for i in range(1,10):
    x = np.load(path+str(i)+"X64.npy")
    y = np.load(path+str(i)+"Y64.npy")
    print(i)
    zX = np.concatenate((zX,x))
    zY = np.concatenate((zY,y))
  return zX,zY
MX_train,MY_trainHot= load_my_data(data_path,0)
My_trainHot=to_categorical(MY_trainHot, num_classes=num_classes)
X_train = np.concatenate((newX, X_train))
X_test = np.concatenate((newX_test,X_test))
y_trainHot = np.concatenate((newY,y_trainHot))
y_testHot = np.concatenate((newY_test,y_testHot))
def balanced_sample_maker(X, y, sample_size, random_seed=None):
    """ return a balanced data set by sampling all classes with sa
mple_size
        current version is developed on assumption that the positi
ve
        class is the minority.

    Parameters:
    ===========
    X: {numpy.ndarrray}
    y: {numpy.ndarray}
    """
    uniq_levels = np.unique(y)
    uniq_counts = {level: sum(y == level) for level in uniq_levels
}

    if not random_seed is None:
        np.random.seed(random_seed)

    # find observation index of each class levels
    groupby_levels = {}
    for ii, level in enumerate(uniq_levels):
        obs_idx = [idx for idx, val in enumerate(y) if val == leve
l]
        groupby_levels[level] = obs_idx
    # oversampling on observations of each label
    balanced_copy_idx = []
    for gb_level, gb_idx in iter(groupby_levels.items()):
        over_sample_idx = np.random.choice(gb_idx, size=sample_siz
e, replace=True).tolist()
```

```
        balanced_copy_idx+=over_sample_idx
    np.random.shuffle(balanced_copy_idx)

    return (X[balanced_copy_idx, :], y[balanced_copy_idx], balance
d_copy_idx)
newX , newY , BCI = balanced_sample_maker(MX_train,MY_train,1000)
model = Sequential()
model.add(Conv2D(32, kernel_size=(5,5), activation = 'relu', input
_shape=(64, 64 ,3) ))
model.add(Conv2D(32,kernel_size=(5,5), activation = 'relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Conv2D(64, kernel_size=(5,5), activation = 'relu', input
_shape=(64, 64 ,3) ))
model.add(Conv2D(64,kernel_size=(5,5), 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(29,activation="softmax"))
opt = tf.keras.optimizers.Adam(learning_rate=0.0001)
model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])
print(model.summary())

history1=model.fit(zX,zY,batch_size=128,epochs=10,validation_split
=0.3, shuffle=True)
plt.plot(history1.history['accuracy'],
         label='Доля верных ответов на обучающем наборе')
plt.plot(history1.history['val_accuracy'],
         label='Доля верных ответов на проверочном наборе')
plt.xlabel('Эпоха обучения')
plt.title('lr=0.001')
plt.ylabel('Точность')
plt.legend()
plt.show()
plt.plot(history1.history['loss'],
         label='Доля ошибки на обучающем наборе')
plt.plot(history1.history['val_loss'],
         label='Доля ошибки на ыпроверочном наборе')
plt.xlabel('Эпоха обучения')
plt.title('lr=0.001')
plt.ylabel('ошибка')
plt.legend()
plt.show()
from __future__ import absolute_import, division, print_function,
unicode_literals
```

```
import os

import tensorflow as tf

from tensorflow import keras

print(tf.version.VERSION)
model.save('/content/drive/My Drive/Sigh/my_model_fullDATA.h5')
```

main.py

```
from cam import *

from PyQt5 import QtCore, QtGui, QtWidgets

import tensorflow,skimage,cv2

import numpy as np

from skimage.transform import resize

import matplotlib.pyplot as plt

import os

import string

import sys

import time


model = tensorflow.keras.models.load_model("my_modelBalanced.h5")

Alphabet=[x for x in list(string.ascii_uppercase)]

Alphabet.append("DEL")

Alphabet.append("NOTHING")

Alphabet.append("SPACE")

def num_to_letter(num):

    letter = Alphabet[num]

    return letter


def Test_image(path_to_image):
```

```
    img_file =path_to_image
    imageSize = 64
    img_file = cv2.resize(img_file, (64,64))
    img_file = skimage.transform.resize(img_file, (imageSize,
imageSize, 3))
    img_arr = np.asarray(img_file).reshape((-1, imageSize,
imageSize,3))
    return img_arr


def predict(img_arr):
    Pletter = num_to_letter(np.argmax(model.predict(img_arr)))
    predict = "это буква - " + Pletter
    return predict
class MyWin(QtWidgets.QMainWindow):
    def __init__(self,parent=None):
        QtWidgets.QWidget.__init__(self,parent)
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)


        self.ui.pushButton.clicked.connect(self.MyFunction)
    def setText(self,prediction):
        self.ui.textEdit.setText("")
        self.ui.textEdit.setText(prediction)
    def settime(self):
        timer = self.ui.textEdit_2.toPlainText()
        try:
            timer = int(timer)
        except:
            timer =1
```

```
        return timer

    def MyFunction(self):
        cap = cv2.VideoCapture(0)
        counter =0
        start_time = time.perf_counter()
        while(True):
            ret, frame = cap.read()
            gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
            cv2.imshow("WebCam Output", gray)
            img = Test_image(gray)
            prediction = predict(img)
            current_time = time.perf_counter()
            if -start_time+current_time >= self.settime() :
                print(prediction)

                self.setText(prediction)
                start_time = time.perf_counter()
            #counter = counter +1
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break

        cap.release()
        cv2.destroyAllWindows()

if __name__ == '__main__':
    app = QtWidgets.QApplication(sys.argv)
    myapp = MyWin()
```

```
myapp.show()

sys.exit(app.exec_())
```