



**Институт  
интеллектуальных кибернетических систем  
Кафедра № 22 «Кибернетика»**

Направление подготовки 01.03.02 Прикладная математика и информатика

**Пояснительная записка**

к учебно-исследовательской работе студента на тему:

Разработка нейросетевого метода и веб-сервиса для распознавания визуальной капчи

---

Группа

Б17-501

Баранова Д.Д.

Студент

(подпись)

(ФИО)

Руководитель

(0-5 баллов)

(подпись)

(ФИО)

Научный консультант

(0-5 баллов)

(подпись)

(ФИО)

Оценка  
руководителя

(0-15 баллов)

Оценка  
консультанта

(0-15 баллов)

Итоговая оценка

(0-100 баллов)

ECTS

Председатель

(подпись)

(ФИО)

(подпись)

(ФИО)

(подпись)

(ФИО)

(подпись)

(ФИО)

**Москва 2020**

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
**«Национальный исследовательский ядерный университет «МИФИ»**

**Институт интеллектуальных кибернетических систем**



**КАФЕДРА КИБЕРНЕТИКИ**

**Задание на УИР**

Студенту гр.	Б17-501		Барановой Д.Д.
	(группа)		(фамилия)

**ТЕМА УИР**

Разработка нейросетевого метода и веб-сервиса для распознавания визуальной капчи

**ЗАДАНИЕ**

№ п/п	Содержание работы	Форма отчетности	Срок исполнения	Отметка о выполнении Дата, подпись рук.
1	<b>Аналитическая часть</b>			
1.1	Классификация и сравнительный анализ типов визуальной капчи	Подраздел ПЗ	30.02.2020	
1.2	Обзор методов машинного обучения, используемых для распознавания капчи	Подраздел ПЗ	10.03.2020	
1.3	Обзор веб-сервисов для распознавания капчи и принципов их работы	Подраздел ПЗ	20.03.2020	
1.4	Обзор библиотек для распознавания капчи	Подраздел ПЗ	04.04.2020	
1.5	<i>Оформление расширенного содержания пояснительной записи (РСПЗ)</i>	Текст РСПЗ	17.04.2020	
2	<b>Теоретическая часть</b>			
2.1	Постановка задачи распознавания капчи	Подраздел ПЗ	17.04.2020	
2.2	Описание метода распознавания капчи	Описание метода	20.04.2020	
2.3	Модель нейронной сети, используемая для распознавания капчи	Описание модели	25.04.2020	
2.4	Метрики качества нейросетевой модели	Подраздел ПЗ	30.04.2020	
3	<b>Инженерная часть</b>			
3.1	Выбор языка и фреймворков для программной реализации веб-сервиса	Выбранный язык	01.05.2020	
3.2	Программная реализация моделей и алгоритмов	Исходный код	05.05.2020	
4	<b>Технологическая и практическая часть</b>			
4.1	Описание обучающих выборок и способа их получения	Набор изображений	06.05.2020	
4.2	Экспериментальное исследование влияния архитектурных параметров нейросетевых моделей на точность распознавания	Подраздел ПЗ	10.05.2020	

4.3	Экспериментальное исследование влияния объёма обучающей выборки на точность распознавания	Подраздел ПЗ	20.05.2020	
4.4	Выводы по результатам экспериментальных исследований	Подраздел ПЗ	01.06.2020	
	<i>Оформление пояснительной записки (ПЗ) и иллюстративного материала для доклада.</i>	Текст ПЗ, презентация	01.06.2020	

## ЛИТЕРАТУРА

•	The Recognition of CAPTCHA Min Wang,Tianhui Zhang,Hao Song,Wenrong Jiang October 4, 2013
•	Breaking Visual CAPTCHAs with Naïve Pattern Recognition Algorithms Jeff Yan, Ahmad Salah El Ahmad
•	Breaking Text-based CAPTCHAs using Average Vertical Partition* XIYANG LIU, YANG ZHANG, JING HU, MENGYUN TANG AND HAICHANG GAO
•	Breaking CAPTCHAs with Convolutional Neural Networks Martin Kopp, Matej Nikl, and Martin Holena
•	CAPTCHA Breaking with Deep Learning, Nathan Zhao, Yi Liu, Yijun Jiang. Autumn 2017
•	CAPTCHA Recognition with Active Deep Learning Fabian Stark, Caner Hazırbaş, Rudolph Triebel, and Danie Cremers

Дата выдачи задания:		Руководитель			Trofimov A.G.
					(ФИО)
« 02 » февраля 2020 г.		Студент			Баранова Д.Д.
					(ФИО)

## Реферат

Пояснительная записка содержит 81 страницу, 50 рисунков, 5 таблиц, 7 формул.

Количество использованных источников – 21.

Ключевые слова: распознавание капчи, нейронная сеть, веб-сервис.

Целью данной работы является разработка нейросетевого метода и веб-сервиса для распознавания визуальной капчи.

В первом разделе рассматриваются различные существующие подходы к распознаванию капчи, классификация разных типов капчи, использующиеся методы машинного обучения для распознавания, существующие библиотеки для работы с изображениями и методами машинного обучения, а также обзор и сравнение существующих веб-сервисов.

В втором разделе приводится описание разработанного метода распознавания капчи, этапы обработки и распознавания, модель используемой нейронной сети.

В третьем разделе приводится описание разработки веб-сервиса, выбор фреймворков для разработки.

В четвёртом разделе приводятся результаты экспериментальные исследования метода распознавания капчи, оценивается влияние архитектуры нейросети и объема обучающей выборки на точность распознавания.

В заключении подводятся итоги проведённой работы с кратким описанием результатов по каждому разделу.

## Содержание

<b>Введение .....</b>	<b>4</b>
<b>Раздел 1. Анализ существующих подходов к распознаванию капчи.....</b>	<b>5</b>
1.1 Классификация и сравнительный анализ типов визуальной капчи.....	5
1.2 Обзор методов машинного обучения, используемых для распознавания капчи.....	7
1.3 Обзор веб-сервисов для распознавания капчи и принципов их работы .....	17
1.4 Обзор библиотек для распознавания капчи.....	18
1.5 Выводы .....	21
1.6 Постановка задачи курсового проекта.....	21
<b>Раздел 2. Разработка нейросетевого метода распознавания капчи.....</b>	<b>23</b>
2.1 Постановка задачи распознавания капчи .....	23
2.2 Описание метода распознавания капчи.....	25
2.3 Модель нейронной сети, используемой для распознавания капчи.....	29
2.4 Метрики качества нейросетевой модели.....	31
2.5 Метрики качества метода распознавания.....	32
2.6 Выводы .....	33
<b>Раздел 3. Разработка веб-сервиса для распознавания капчи.....</b>	<b>34</b>
3.1 Выбор языка и фреймворков для программной реализации веб-сервиса.....	34
3.2 Программная реализация моделей и алгоритмов.....	36
3.3 Выводы .....	45
<b>Раздел 4. Экспериментальные исследования метода распознавания капчи .....</b>	<b>46</b>
4.1 Описание обучающих выборок и способа их получения.....	46
4.2 Экспериментальное исследование влияния архитектурных параметров нейросетевых моделей на точность распознавания.....	47
4.3 Экспериментальное исследование влияния объёма обучающей выборки на точность распознавания.....	57
4.4 Выводы по результатам экспериментальных исследований .....	61
<b>Заключение .....</b>	<b>62</b>
<b>Литература.....</b>	<b>63</b>
<b>Приложения.....</b>	<b>65</b>

## Введение

Капча (англ. “CAPTCHA”) — это компьютерный тест, используемый для того, чтобы определить, кем является пользователь системы: человеком или компьютером. Ставится задача, которую легко решает человек, в то время как для машины эта задача должна быть сложной.

Поскольку проблемы интернет-безопасности обостряются, очень важно провести исследование существующих типов капчи как элемента киберзащиты. Расшифровка капчи веб-сайта означает, что он может подвергаться вредоносным атакам со стороны компьютеров, поэтому надежность и практичность капчи стали важной гарантией безопасности веб-сайтов. Разработка веб-сервиса для распознавания поможет выявить слабые места в исследованных типах капчи и, возможно, разработки более надежных способов защиты веб-сайтов от автоматических взломов. Также сервис позволит взламывать существующие виды капчи при интеграции системы в интернет браузеры.

Целью данной работы является построение веб-сервиса, выполняющего автоматическое распознавание капчи на изображении. В рамках работы решаются следующие задачи:

- Получение размеченной обучающей выборки.
- Изучение и разработка методов обработки изображения.
- Выбор и разработка оптимальной архитектуры нейросети.
- Разработка веб-сервиса.

В первом разделе приводится описание и сравнительный анализ различных уже существующих подходов к распознаванию капчи.

Во втором разделе приводится описание метода распознавания капчи и исследование метрик качества разработанной модели.

В третьем разделе приводятся требования и проект разрабатываемой системы, описание реализованных функций и модулей.

В четвёртом разделе приводятся результаты экспериментальных исследований влияния параметров архитектуры выбранной модели нейронной сети и влияния объема обучающей выборки на точность работы, в результате чего получены оптимальные параметры и оптимальный объем выборки.

В заключении подводятся итоги проведённой работы с кратким описанием результатов по каждому разделу.

# **Раздел 1. Анализ существующих подходов к распознаванию капчи**

Сейчас существует несколько платных веб-сервисов для распознавания капчи, спроектированных на основе использования различных методов машинного обучения для распознавания текста на изображении. Успешность и скорость их распознавания во многом зависит от вида капчи, поданной на вход.

Капча классифицируется на несколько видов исходя из своей сложности. Самая распространенная – текстовая. Однако существует также звуковой тип капчи (строка, которую необходимо ввести, проговорит робот при нажатии на иконку “наушники или звук”), поведенческий вид капчи (упор на взаимодействие с пользователем, к примеру поворот картинки в нужное положение), логический (решить пример или выбрать конкретную картинку из нескольких) [1]. В данной работе будет рассматриваться текстовая капча.

## **1.1 Классификация и сравнительный анализ типов визуальной капчи**

Рассмотрим классификацию текстовой капчи. Её можно сравнивать по выполнению своих двух функций: анти-распознавания и анти-сегментации. Обе эти функции направлены на усложнение выполнения отдельных этапов процесса распознавания капчи.

Одним из этапов процесса распознавания, является изоляция различных объектов друг от друга, а также от фона. Такое разделение изображения называется сегментацией. К функции анти-сегментации могут относиться наложения дополнительных линий и кривых, наложение символов друг на друга и т.д.

К функции анти-распознавания можно отнести использование необычного шрифта, использование дополнительных элементов на фоне, которые не включены в алфавит и могут быть ошибочно распознаны как символ ответа, использование искажений, т.е. все те методы, которые отдаляют объект от его обычного вида.

В зависимости от характерных особенностей капчи необходимо разрабатывать свой подход к предварительной обработке капчи перед процессом распознавания [4].

Основные особенности текстовой капчи:

1. Большой набор символов. Чем больше набор символов в алфавите, тем больше общее количество строк. Необходимо увеличивать количество классов, на которые метод распознавания делит символы (усложняется распознавание).

2. Искажения и перекрытия. Символы с искажениями и перекрытием сложнее поддаются сегментации на отдельные символы. Необходимо разрабатывать специальные методы сегментации и поворота.

3. Различия по размеру, ширине, углу, расположению и шрифту. Различные преобразования могут снизить точность распознавания. Необходимо разработать методы масштабирования, посимвольного поворота, разнообразить обучающую выборку объектами с различными шрифтами.

4. Строки с нефиксированной длиной, строки из нескольких слов. Увеличивают вероятность ошибки распознавания строки. Необходимо установить предел расстояния между символами, после превышения которого алгоритм разделял бы строку ответа на несколько слов.

5. Пустые символы. По сравнению со сплошными символами, полый шрифт меньше. Он эффективно противостоит распознаванию. Помочь может заливка полостей и избавление от контуров.

6. Цвет и форма сложных фонов аналогичных цветам символов. Если изображения соответствуют этим условиям, шум трудно удалить. Это может снизить точность распознавания. Необходимо разработать алгоритм определения границы окончания фона и начала символа.

В большинстве случаев одна капча комбинирует в себе несколько особенностей для усложнения и сегментации, и распознавания. Многообразие особенностей приводит к тому, что практически невозможно создать универсальный алгоритм обработки изображения перед распознаванием, который мог бы избавить от характерных особенностей любой тип капчи.

В таблице 1 расположены наиболее популярные различные типы капчи в порядке увеличения их сложности и повышения уровня защиты. Перечисленные особенности повышают эффективность капчи и в то же время ставят серьезные задачи перед исследованиями способов взлома капчи.

Таблица 1. Сравнение существующих видов текстовой капчи

Тип	Пример	Сайт	Особенности
Solid CAPTCHA		Discuz!	Независимые символы, текстурный фон, помехи
		Slashdot	Большое количество помех в виде линий и шумовых точек
		Gimpy	Иногда несколько строк, перекрытие, искажение, градиент на фоне

		Google	Нефиксированная длина, искажение, перекрытие
		Microsoft	Двойная строка, нефиксированная длина, неравномерная толщина, наклон, перекрытие
Hollow CAPTCHA		QQ	Полый шрифт, тени, сложный фон
		Sina	Полый шрифт, перекрытие, пересекающие линии
		Yandex	Полый дырявый шрифт, искажение, перекрытие, пересекающие линии
Three-dimensional CAPTCHA		Scihub	Полый шрифт, тени, пересекающие линии, шум
		Teabag	Шрифт из сетки, выпячивание, искажение, смешивание фона и символов
		Parc	Разный цвет, тени, вращение, разный размер, специальные символы
Animation CAPTCHA		Program generating	Несвязанные символы, поворот, шум из символов
		Hcaptcha	Пересечение символов, искаженных трансформацией

## 1.2 Обзор методов машинного обучения, используемых для распознавания капчи

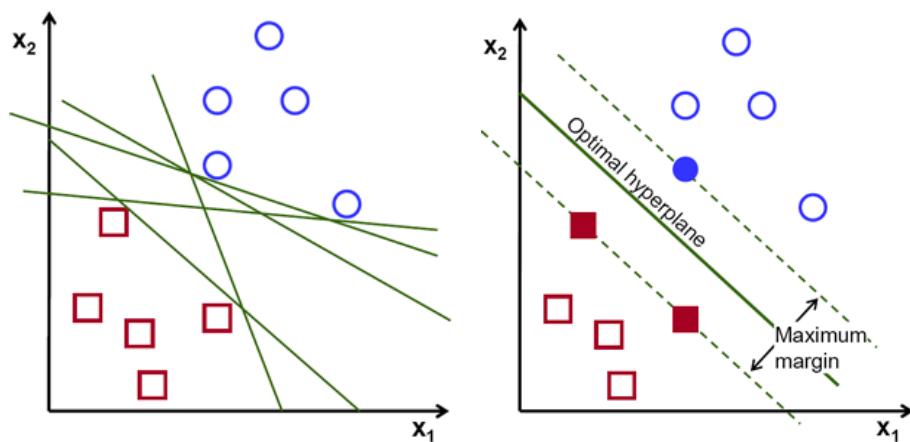
Методы распознавания капчи в большинстве случаев используют алгоритмы машинного обучения для правильной классификации символов CAPTCHA. В соответствии с хронологическим порядком, их можно разделить на две категории: традиционные методы (SVM и KNN) и методы, основанные на нейронных сетях [2].

### 1.2.1 Метод опорных векторов (Support Vector Machine, SVM)

Пусть имеется  $X$  — пространство объектов (например,  $\mathbb{R}^n$ ) и множество  $Y$  — классы этих объектов (например,  $Y = \{-1, 1\}$ ). Данна обучающая выборка: данных  $(x_1, y_1), \dots, (x_m, y_m)$ . Требуется построить функцию  $F : X \rightarrow Y$  (классификатор), сопоставляющий класс  $y$  произвольному объекту  $x$ .

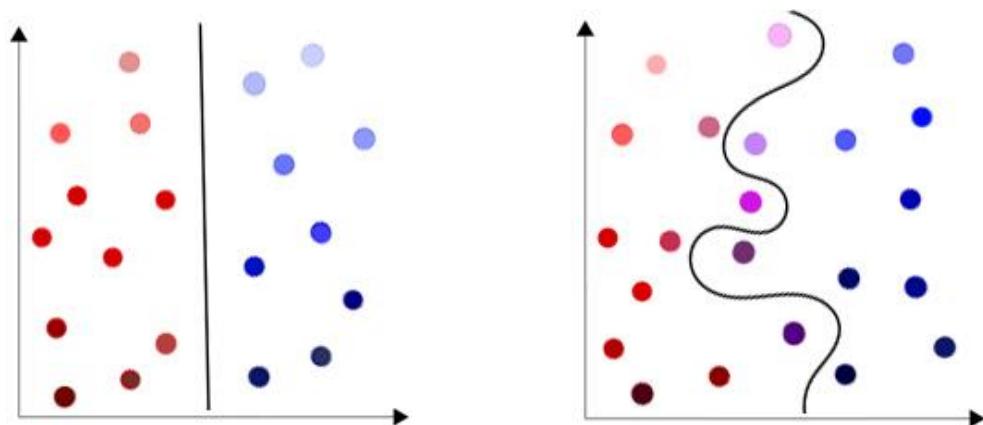
Основная идея метода опорных векторов (Support Vector Machine) заключается в построении гиперплоскости, разделяющей оптимальным способом объекты обучающей выборки, представленные точками в многомерном пространстве  $X$ . Построение гиперплоскости (функции, которая её задает) работает в предположении, что чем больше расстояние (зазор) между разделяющей гиперплоскостью и объектами разделяемых классов, тем меньше будет ошибка классификатора.

Более формально, нужно отыскать оптимальную гиперплоскость, разделяющую множество точек, для которых  $y_i = 1$  и множество точек, для которых  $y_i = -1$ , причем расстояние от неё до ближайших точек из обоих классов должно быть максимальным.



*Рис.1 Выбор оптимальной гиперплоскости для разделения выборки на кластеры*

Однако не всегда объекты можно разделить линейно. В этом случае их нельзя разделить гиперплоскостью и объекты называются линейно неразделимыми (линейно несепарабельными). К примеру, на рисунке 2 изображены линейно разделимые и линейно не разделимые множества объектов при классификации на классы красный и синий (в качестве признаков могут выступать яркость цвета и величина Blue в модели RGB).



*Рис.2 Линейно разделимые и линейно не разделимые множества*

В случае линейной неразделимости в методе делается допущение, что некоторые из точек будут классифицированы неверно. Метод опорных векторов способен эффективно решать задачу нелинейной классификации, используя неявное отображение  $\varphi(\vec{x})$  данных  $(\vec{x}_1, y_1), \dots, (\vec{x}_n, y_n)$ , в пространство признаков большей размерности, в предположении, что там данные будут линейно разделимы.

Нужно отметить, что при реализации метода опорных векторов существуют несколько недостатков, среди которых:

- применимость только для двух классов
- не существует общего подхода к автоматическому выбору функции  $\varphi(\vec{x})$  (переводящей объекты к пространство большей размерности) в случае линейной неразделимости классов.

В исследовании [13] для задачи распознавания капчи был использован метод SVM для попарной классификации объекта (символа). Для  $M$  классов, было использовано  $\frac{M*(M-1)}{2}$  классификаторов.

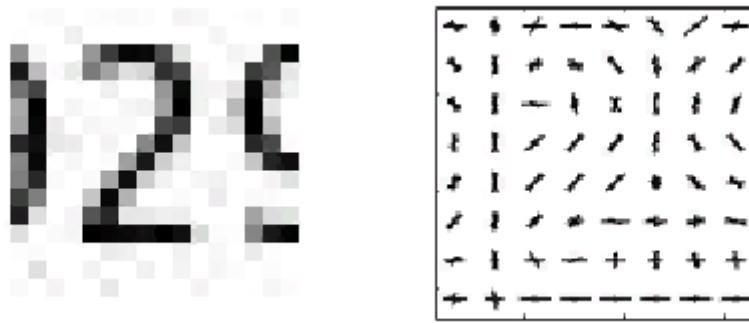
Для случайно выбранных капч после процесса их предварительной обработки и сегментации (Рис.3) система показала 67,5% точности распознавания одиночного символа и 20% точности распознавания целой капчи.



*Рис.3 Примеры распознаваемых сегментированных символов*

Для сегментации был выбран метод проекции. Его работа заключается в следующем. Будем суммировать количество черных пикселей в каждом столбце изображения. По полученному массиву суммы построим гистограмму. Будем считать, что область между вершиной и впадиной является отдельным символом. Мы можем установить порог сегментации ( $Threshold = 1$ ) и определить столбец, значение которого ниже порога, в качестве разделительной линии. Определив левую и правую границу символа, мы можем провести вертикальную сегментацию по этим границам и горизонтальную сегментацию, чтобы ограничить символ меньшей областью. Не указано, что было взято в качестве признаков для построения объектов в пространстве признаков. После распознавания с помощью SVM производилась склейка некоторых часто неверно распознаваемых символов (к примеру, кавычки “(“ и “)” в букву “о”).

В исследованиях [18][19] в качестве признаков берутся значения количества черных пикселей в каждом столбце и каждой строке изображения. В исследовании [5] для построения пространства признаков используются признаки, сформированные на основе HOG (Histogram of Oriented Gradients) дескриптора. Техника основана на построении локальных гистограмм направлений градиентов яркости (интенсивности) в локальных областях изображения. Суть метода состоит в том, что изображение плотной равномерной сеткой разбивается на области, для каждой из которых строится локальная гистограмма направлений градиентов яркости. Совокупность построенных гистограмм будет являться дескриптором объекта. Последним этапом в распознавании является классификация дескрипторов при помощи SVM.

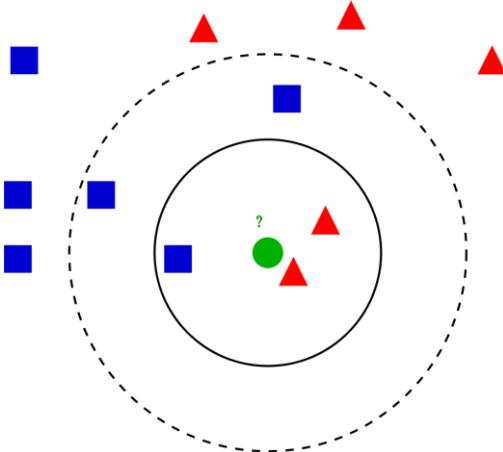


*Рис.4 Пример преобразования изображения в HOG-представление*

### 1.2.2 Метод k-ближайших соседей (K-Nearest Neighbors, KNN)

Метод k-ближайших соседей (KNN) создает k-групп из набора объектов таким образом, чтобы члены группы были наиболее однородными. Алгоритм стремится минимизировать суммарное квадратичное отклонение точек кластеров (групп) от центров этих кластеров. Другими словами, это итеративный алгоритм, который делит данное множество пикселей на k кластеров точки, которых являются максимально приближенными к их центрам, а сама кластеризация происходит за счет смещения этих же центров.

Метод обычно используется для кластеризации, т.е. разбиения выборки на k кластеров (групп), но может быть использован и для классификации. В этом случае объект присваивается тому классу, который является наиболее распространённым среди k ближайших соседей данного элемента, классы которых уже известны. Алгоритм KNN находит наиболее распространенный класс среди k ближайших соседей. Затем считается количество соседей, принадлежащих одному классу, и выигрывает класс, которому принадлежит наибольшее количество объектов среди k соседей.



*Рис.5 Пример классификации методом k-средних*

Рассмотрим пример классификации k-ближайших соседей. Пусть имеется 2 класса объектов (синие квадраты и красные треугольники), расположенных в некотором пространстве признаков. Тестовый объект (зелёный круг) должен быть классифицирован как синий квадрат (класс 1) или как красный треугольник (класс 2). Если  $k = 3$ , то он классифицируется как 2-й класс, потому что внутри меньшего круга 2 треугольника и только 1 квадрат. Если  $k = 5$ , то он будет классифицирован как 1-й класс (3 квадрата против 2 треугольников внутри большего круга).

Основной вопрос при использовании данного алгоритма — выбор числа  $k$ . При  $k = 1$  алгоритм теряет устойчивость и начинается себя неадекватно вести при появлении шумов, при  $k$ , близком к числу векторов обучающей выборки, точность становится избыточной и алгоритм вырождается. [11] [14]

В исследовании [5] приводится описание использования метода для классификации рукописных символов. Классификатор тренировался при различных значениях  $k$  от 1 до 30 с шагом 2, после чего производилось сравнение точности распознавания при каждом  $k$ . Для вычисления точности было произведено распознавание случайно выбранных символов из выборки. На рисунке изображены полученные результаты.

```

k=1, accuracy=99.26%
k=3, accuracy=99.26%
k=5, accuracy=99.26%
k=7, accuracy=99.26%
k=9, accuracy=99.26%
k=11, accuracy=99.26%
k=13, accuracy=99.26%
k=15, accuracy=99.26%
k=17, accuracy=98.52%
k=19, accuracy=98.52%
k=21, accuracy=97.78%
k=23, accuracy=97.04%
k=25, accuracy=97.78%
k=27, accuracy=97.04%
k=29, accuracy=97.04%

```

*Рис.6 Точность распознавания в зависимости от коэффициента k*

Метод показал лучшую точность при значениях  $k < 17$ .

Для распознавания капчи в исследовании [10] также был использован метод KNN. Значение k выбиралось аналогичным образом. Обучение было произведено на выборке в размере 1500 образцов символов. Процент успешности лучшей атаки варьируется от 5,0% до 74,0% для разных типов капчи.

Вид капчи	Показатель успеха
Baidu	54.2%
Taobao	24.2%
Google Street View	5.0%
Google Map	74.0%
Sina	37.6%
Weibo	31.8%
Amazon	51.6%
QQ	46.4%
eBay	47.2%
Microsoft	15.2%
Yahoo!	20.4%
Facebook	12.4%
Wikipedia	37.4%

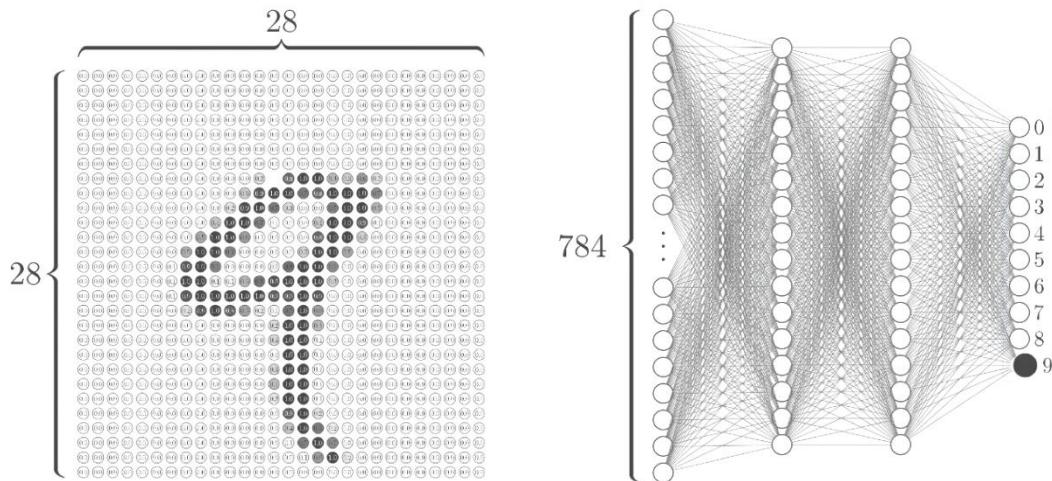
*Рис.7 Точность распознавания в зависимости от вида капчи*

### 1.2.3 Методы распознавания на основе нейросетей

Рассмотрим работу многослойного перцептрона (MLP) для задачи распознавания символов.

Пусть для упрощения наш алфавит состоит только из цифр. Пусть в качестве входных данных будут выступать квадратные изображения размером  $h = 28$  пикселей. Тогда на вход сети будет подаваться  $h * h$  значений — в нашем случае это  $28 * 28 = 784$ . Количество

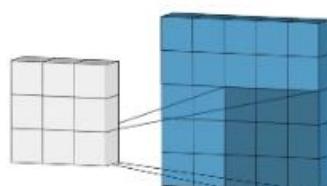
нейронов внутреннего слоя такое же и равно 784. Чтобы распознавать цифры, мы создаем сеть с десятью выходами, и единица будет на выходе, соответствующем нужной цифре. Получим сеть, представленную на рисунке 8.



*Рис.8 Построение многослойного перцептрона для цифры 9*

Рассмотрим работу свёрточной сети. Она обязана своим названием архитектуре, которая построена только из локально связанных слоев, таких как свертка, объединение и развертывание.

В обычном перцептроне, который представляет собой полносвязную нейронную сеть, каждый нейрон связан со всеми нейронами предыдущего слоя, причём каждая связь имеет свой персональный весовой коэффициент. В свёрточной нейронной сети в операции свёртки используется лишь ограниченная матрица весов (ядро) небольшого размера, которую «двигают» по всему обрабатываемому слою (в самом начале — непосредственно по входному изображению). Это формирует матрицу значений, которая подается на вход следующему слою сети. Её интерпретируют как графическое кодирование какого-либо признака, например, наличие наклонной линии под определённым углом. Тогда следующий слой, получившийся в результате операции свёртки такой матрицей весов, показывает наличие данного признака в обрабатываемом слое и её координаты, формируя так называемую карту признаков.



*Рис.9 Демонстрация свертки*

К преимуществам свёрточной нейронной сети относят:

1. Один из лучших алгоритмов по распознаванию и классификации изображений.

2. Меньшее количество настраиваемых весов (по сравнению с полносвязной нейронной сетью - перцептроном), так как одно ядро весов используется целиком для всего изображения, вместо того, чтобы делать для каждого пикселя входного изображения свои персональные весовые коэффициенты. Это подталкивает нейросеть при обучении к обобщению демонстрируемой информации, а не попиксельному запоминанию каждой показанной картинки в мириадах весовых коэффициентов, как это делает перцепtron.

3. Устойчивость к повороту и сдвигу распознаваемого изображения.

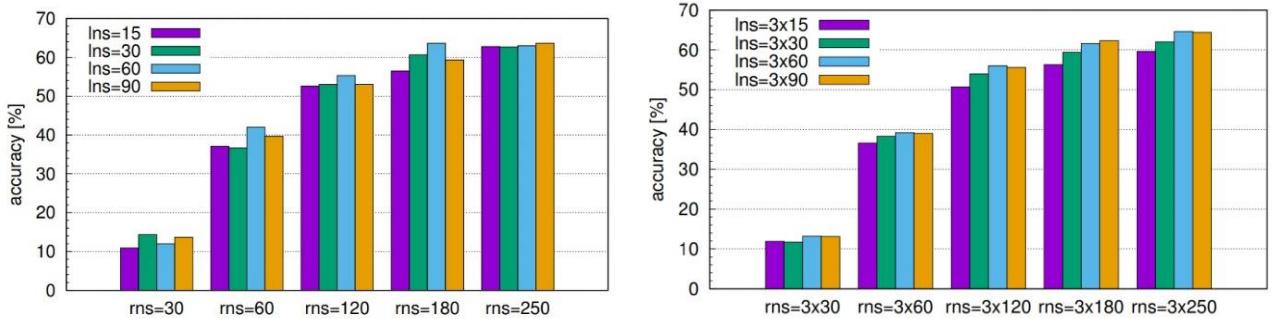
4. Обучение при помощи классического метода обратного распространения ошибки.

Недостатком является сложность использования как следствие большого количества настраиваемых параметров (количество слоев, размерность ядра, шаг сдвига ядра и т.д.).

Аналогично схеме построения перцептрана для распознавания цифры 9, в исследовании [12] строится сеть из 36 выходных слоев, способная классифицировать алфавит размером 36 (цифры 0–9 и заглавные буквы A – Z). Для хорошей работы нейронная сеть требует большего количества данных, чем предыдущие классические методы (порядка миллиона для очень глубокой сети). И многослойный перцептрон, и сверточные нейронные сети могут страдать от большого объема вычислений на этапе обучения. Данная сеть была обучена на выборке размера 100 000 для каждого символа. Были исследованы 11 видов капчи. В них присутствуют случайные линии и дополнительные объекты, закрывающие символы, неровные или полупрозрачные края символов. Этап предварительной обработки не описан.

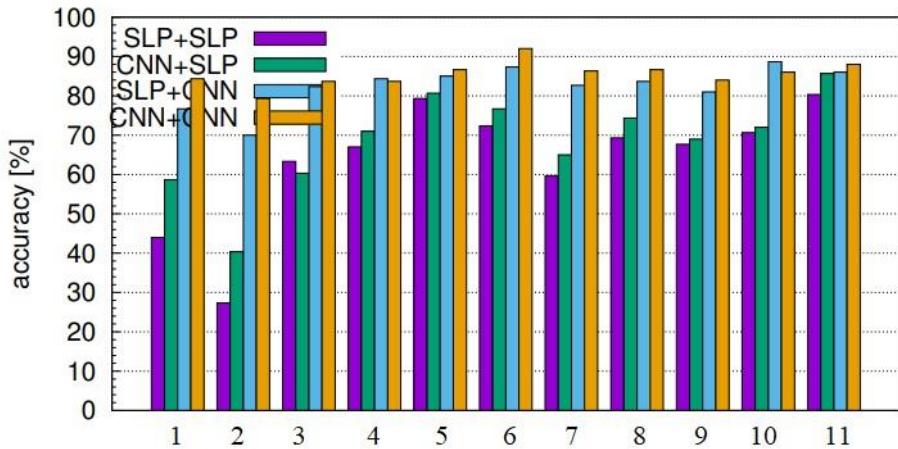
Сеть использовалась как для распознавания, так и для сегментации. Были протестированы перцептрон с одним скрытым слоем (SLP), персептрон с тремя скрытыми слоями (MLP) и сверточные нейронные сети.

В первом эксперименте проверялось влияние количества скрытых нейронов SLP. Количество скрытых нейронов, использованных для сети локализации (сегментации), было  $l_{ns} = \{15, 30, 60, 90\}$ , а количество нейронов для сети распознавания было  $r_{ns} = \{30, 60, 120, 180, 250\}$ . Результаты, изображенные на рис. 10(а), показывают точность распознавания для 1000 изображений капчи (все символы должны быть правильно распознаны). Следующий эксперимент был таким же, но вместо SLP использовался MLP с тремя скрытыми слоями. Результаты, изображенные на рис. 10(б), показывают, что добавление большего количества скрытых слоев не повышает точность локализации и распознавания. Поэтому остальные эксперименты проводились с использованием SLP, так как его можно обучать быстрее.



*Рис.10(а), 10(б) Сравнение точности в зависимости от количества слоев сетей*

После подбора оптимальной архитектуры было проведено сравнение точности распознавания на 11 типах капчи. Все капчи состояли из 5 символов. Распознавание всех символов корректно означало 100% точность. Результаты приведены на рисунке 11. CNN явно превосходит SLP в роли сети распознавания.



*Рис.11 Сравнение точности в зависимости от типа сети*

В исследовании [3] для решения задачи распознавания символов сравнивалась работа MLP и CNN (три различные архитектуры). Для распознавания был использован chars74k dataset. Исследование показало, что использование CNN дает в 2 раза более точный результат распознавания, чем использование MLP при классификации символов (43% для MLP, 88-90% для CNN).



*Рис.12 Примеры символов в chars74k dataset*

После сравнения работы методов Support Vector Machine, MLP и Convolutional Neural Network в исследовании [7] по распознаванию капчи было замечено, что CNN показывает себя наиболее успешно в распознавании по сравнению с другими методами. Она достигает большей скорости распознавания, а также лучшего показателя успеха среди этих методов.

Также были произведены сравнения KNN и CNN [6]. KNN достигает более высоких показателей успеха в большинстве капч, чем CNN, но в большинстве случаев CNN работает быстрее (Таблица 2).

*Таблица 2. Сравнение показателей работы CNN и KNN*

Вид капчи	Точность		Скорость	
	KNN	CNN	KNN	CNN
Yahoo!	5.0%	5.2%	28.56	23.81
Baidu	44.2%	46.6%	2.81	2.21
Wikipedia	23.8%	20.4%	3.74	2.90
QQ	56.0%	22.4%	4.95	4.61
Microsoft	16.2%	8.6%	12.59	6.64
Amazon	25.8%	20.2%	13.18	8.68
Taobao	23.4%	20.4%	4.64	5.25
Sina	9.4%	4.4%	4.83	5.21
Ebay	58.8%	32.6%	5.98	5.50

В существующих исследованиях [8][9] широко используется метод CNN с хорошей точностью распознавания (больше 90% точности распознавания на капчах Gimpy, Google).

Таким образом, среди методов машинного обучения, используемых для распознавания капчи, самым широко используемым является CNN (Convolutional Neural Network). Она показывает хорошее сочетание точности (точнее всех других методов. Чуть менее точная, чем KNN) и скорости (быстрее KNN), хорошую устойчивость к искажениям и смещениям символов.

### **1.3 Обзор веб-сервисов для распознавания капчи и принципов их работы**

Сервисы для распознавания изображений и капч могут использовать в своей основе человеческий труд или решения OCR (оптического распознавания символов). Использующиеся методы автоматического распознавания авторы не раскрывают.

Ниже приведена таблица сравнения существующих сервисов распознавания капч.

*Таблица 3. Сравнение существующих веб-сервисов распознавания капчи*

Сервис	Метод	Показатель успеха	Стоимость	Время ответа	Ограничения
Anticaptcha	Человек	90%	~1.5\$ /1000	7 сек	Не решает капчи с большим количеством символов, ограниченный размер изображения
Image Typerz	Человек	95%	~2\$ /1000	10 сек	Ограниченный размер изображения
Rycaptcha	Человек	99%	42₽ /1000	20 сек	Разная стоимость и показатель успеха в зависимости от времени суток
EndCaptcha	Человек	90%	12\$ /3000	8-13 сек	Не решает капчи с большим количеством символов
Bypass Captcha	Человек	95%	14\$ /2000	8-13 сек	Ограничение на количество (нет многопоточности)
2captcha	Человек	85%	\$1 / 1000	14 сек	
Best Captcha Solver	Человек	85%	~1\$ /1000	14 сек	Не решает капчи с большим количеством символов
CaptchaSniper	OCR	-	57\$	-	Поддерживаются определенные типы капч, установка ПО
Captcha Tronix	OCR	60%	14\$ /месяц	0.68 сек	Поддерживаются определенные типы капч
AZcaptcha	OCR	70%	25\$ /месяц	1-5 сек	Поддерживаются определенные типы капч
Deathbycaptcha	Человек + OCR	80%	~1.5\$ /1000	11 сек	Поддерживаются определенные типы капч

Как можно видеть, веб-сервисы, использующие человеческий труд в своей основе, дают больший процент верно распознанных капч. Хорошим показателем успеха для человека является 75% и выше.

Таким образом, у методов распознавания, не использующих человеческий труд в основе, существуют много ограничений на типы капч и показатель успеха обычно ниже. Однако их время работы и многопоточность позволяет составить конкуренцию человеку.

#### 1.4 Обзор библиотек для распознавания капчи

Задача распознавания капчи не является совершенно новой, вследствие чего существует несколько готовых библиотек с открытым исходным кодом для решения этой или похожих задач. Библиотеки содержат реализацию некоторых методов обработки изображений, реализацию различных методов машинного обучения, а также функции генерации обучающих выборок.

Для обработки изображений используются библиотеки PIL (Pillow) и OpenCV. Для реализации необходимых методов машинного обучения используются библиотеки Scikit-learn, Keras и TensorFlow, способные также оптимально и просто построить нейронную сеть.

В библиотеке OpenCV реализованы функции для автоматической бинаризации изображения, выделения контуров, размытия и удобной работы с отдельными пикселями. К примеру, следующий код позволяет сгладить исходное изображение и затем выделить края.

```
1. import cv2
2. img = cv2.imread("input.jpg")
3. # разглаживание изображения
4. img = cv2.medianBlur(img, 5)
5. #выделение границ
6. edges = cv2.Canny(img, 100, 200)
7. cv2.imwrite('output.png', edges)
```



Рис.13 Результат работы кода

Существует несколько библиотек машинного обучения. Scikit-learn содержит реализации практически всех возможных преобразований, а также основные алгоритмы машинного обучения (метод опорных векторов, многослойный перцептрон, метод к-средних и т.д.). В данной библиотеки реализованы методы разбиения датасета на тестовый и обучающий, вычисление основных метрик над наборами данных.

```
1. from sklearn.neural_network import MLPClassifier  
2. mlp = MLPClassifier(hidden_layer_sizes=(10, 10, 10), max_iter=1000)  
3. mlp.fit(X_train, Y_train)  
4. predictions = mlp.predict(X_test)
```

Рассмотрим пример кода для построения модели перцептрана с помощью этой библиотеки. При построении модели можно задать некоторые параметры. *hidden\_layer\_sizes* - этот параметр позволяет задать количество слоев и количество нейронов, которые мы хотим иметь в нейронной сети. Каждый элемент в кортеже представляет собой число узлов в *i*-й позиции, где *i*-индекс кортежа. Таким образом, длина кортежа обозначает общее количество скрытых слоев в сети. В данном примере имеем 3 скрытых слоя по 10 узлов в каждом. *max\_iter* - обозначает количество эпох. В данном примере количество эпох = 1000. Метод *fit* производит обучение сети, где *X\_train* – обучающая выборка, *Y\_train* – целевое значение (класс, к которому относится объект *X\_train*). Метод *predict* возвращает предсказанный класс объекта *X\_test*. Также дополнительно можно задать скорость обучения сети, функцию активации и т.д.

TensorFlow хороша для сложных проектов, таких как создание многослойных нейронных сетей. Она используется для распознавания голоса или картинок и приложений для работы с текстом, таких как Google Translate, например. В отличие от Scikit-learn, библиотека позволяет создать сверточную нейронную сеть.

В отличие от TensorFlow, библиотека Keras является более высокоуровневой. TensorFlow более многофункционален и гибок, однако может быть не так прост в использовании. Keras - это высокоуровневый API, построенный на TensorFlow. Он более удобен и прост в использовании по сравнению с TF. С другой стороны, с помощью TF можно получить больший контроль над своей сетью и лучше ее настроить. Как и низкоуровневые библиотеки, TensorFlow способна предоставить лучшую настраиваемость и гибкость. Высокоуровневые библиотеки понятнее и проще, требуют меньше строк кода, однако не позволяют построить очень специфичную нейронную сеть.

Существует реализованный бесплатный модуль Python-tesseract, представляющий собой готовый инструмент для оптического распознавания (OCR) текста. Его работа также

основана на использовании нейронных сетей. Некоторые проекты [16] по распознаванию капчи используют его для своей работы.

В реализованных отдельными пользователями библиотеках [15][17] при решении задачи распознавания капчи можно найти собственные методы для предобработки входного изображения. К примеру, функции поиска масок для удаления дополнительных линий фона, функции усреднения цвета на изображении, обнаружение черных пикселей. В библиотеке [15] происходит поиск и выделение отдельных линий, пересекающих капчу. Эти линии постоянны, и их легко получить с помощью усреднения 20000 капч обучающей выборки и отсечения по заданному порогу.

```
1. def generateMask():
2.     N = 20000
3.     arr = np.zeros((HEIGHT, WIDTH), np.float)
4.     #усреднение капч
5.     for name in IMG_NAMES:
6.         img = np.array(Image.open(name), dtype=np.float)
7.         arr = arr + img / N
8.     #отсечение ярких полос по границе 230
9.     arr = arr.point(lambda x:255 if x > 230 else 0)
10.    return arr
```

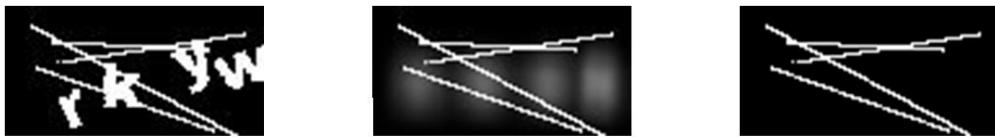


Рис.14 Результат работы кода (капча, усредненное изображение, маска линий)

Однако готовые библиотеки, разработанные при решении аналогичной задачи другими пользователями, мало полезны, так как они в основном содержат реализацию модели машинного обучения для распознавания капчи и мало полезных функций для предварительной обработки изображений и сегментации.

Библиотека TensorBoard позволяет визуализировать работу TensorFlow. В частности, показать, как меняются со временем обучения веса и значение функции ошибки. Она позволяет отслеживать потери и точность, может помочь определить момент переобучения, при котором нейронная сеть начинает терять точность при продолжении обучения. Один из способов быстрого обнаружения проблем с построенной сетью - это графическая визуализация того, что в ней происходит в режиме реального времени (например, каждые 100 итераций). Можно выбрать, какие параметры должны отображаться, и TensorBoard будет

поддерживать визуализацию этих значений в реальном времени во время обучения. К примеру, на рисунке 15 можно видеть уменьшение значения функции потерь в зависимости от номера эпохи. Можно заметить, как быстро уменьшаются потери при обучении (рыжая линия) и проверке (синяя линия), а затем остаются стабильными. На самом деле, можно было бы прекратить тренировки данной сети после 25 эпох, потому что обучение не сильно улучшилось после этого момента.

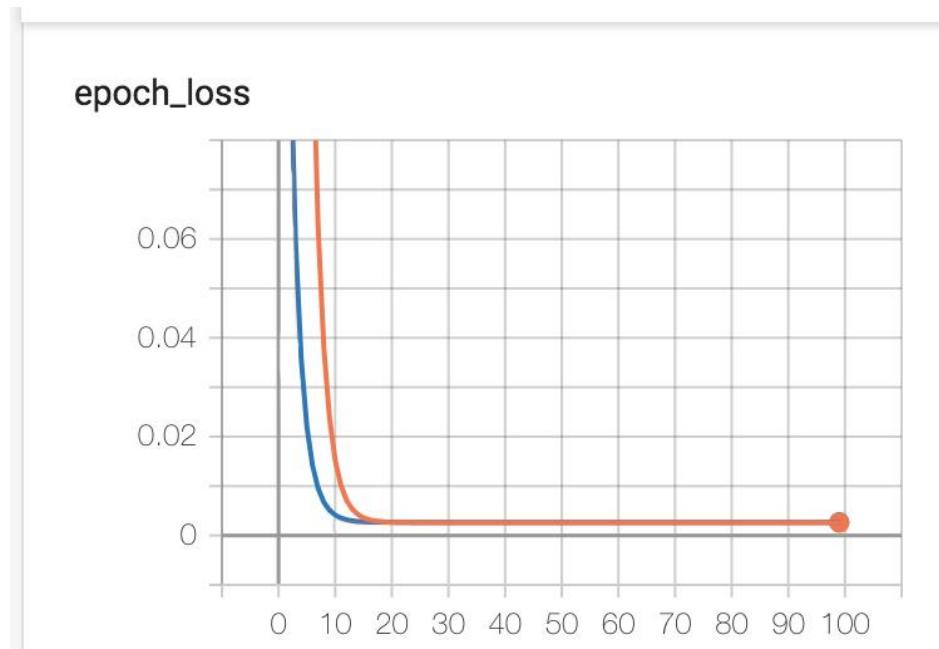


Рис.15 Зависимость функции потерь от количества прошедших эпох обучения

## 1.5 Выводы

Разработка метода распознавания капчи строится исходя из типа и особенностей выбранной капчи. Множество видов капчи различаются по степени сложности их распознавания. Выбор слишком простой капчи может привести к легкому взлому и отсутствии безопасности сайта от кибератак, в то время как выбор слишком сложной капчи может повлечь затруднения для человеческого распознавания. В данной работе планируется использование нейросетевых методов, т.к. они могут распознавать более глубокие закономерности в предоставляемых данных.

## 1.6 Постановка задачи курсового проекта

В результате проведенного обзора установлено, что в настоящее время существует несколько веб-сервисов, использующих методы оптического распознавания символов, основанные на различных методах машинного обучения. Капча может сильно отличаться по

своим функциям противодействия взлому, поэтому все алгоритмы взлома капчи основываются на её характерных особенностях и слабых сторонах. В связи с этим, в данном проекте ставится цель создания веб-сервиса для автоматического распознавания типа капчи и распознавания самой капчи исходя из её типа. Для достижения этой цели предполагается решить следующие задачи:

1. Получение размеченной обучающей выборки
2. Изучение и разработка методов обработки изображения
3. Выбор и разработка оптимальной архитектуры нейросети
4. Разработка веб-сервиса

## **Раздел 2. Разработка нейросетевого метода распознавания капчи**

В разделе ставится задача распознавания капчи и приводится описание предлагаемого метода распознавания и используемых метрик качества распознавания. Задача разбивается на несколько подзадач и выделяются этапы процесса распознавания. Приводятся способы решения поставленных подзадач.

### **2.1 Постановка задачи распознавания капчи**

Задача распознавания капчи заключается в поиске по заданному изображению строки символов, которые на ней находятся. Такую строку будем называть ответом. Другими словами, при работе капчи всегда генерируется уникальная пара «изображение – строка», причем строка, соответствующая изображению, всегда единственна. Необходимо определить ту единственную строку, которая будет являться верной для заданного изображения.

Разобьем задачу распознавания капчи на несколько этапов:

- Предобработка
  - Фильтрация
  - Восстановление
- Сегментация
- Распознавание
- Постобработка



*Рис.16 Блок-схема этапов распознавания*

Введем понятие предобработки. На заданном изображении часто может присутствовать шум, дополнительные элементы, искажения. Предобработкой будет являться процесс работы с изображением, в результате которого из исходного изображения получается изображение только черного и белого цвета, причем символы, которые отображают символ, являются черными, а все иные – белыми. В процессе предобработки (фильтрации) изображение очищается от шумов, фона, проходит процесс бинаризации (перевод полноцветного или в градациях серого изображения в монохромное, где присутствуют только два типа пикселей). Также в предобработку может быть включен процесс масштабирования, избавления от искажений, поворотов символов (процесс восстановления), но он не является обязательным элементом предобработки.

Обозначим введенное ранее понятие сегментации. Процесс сегментации включает в себя определение границ каждого символа и нарезку изображения таким образом, чтобы каждому символу на входном изображении соответствовало изображение, полученное из входного, на котором присутствует только этот символ.

Процесс распознавания – это определение класса, к которому можно отнести входящее изображение. Определяя класс, мы однозначно определяем символ алфавита, которому соответствует этот класс.

Процесс постобработки включает в себя восстановление ответа. Он может состоять только из сортировки полученных символов алфавита по координате x, или включать еще несколько различных дополнительных функций.

Решение поставленной задачи в данной работе будет осуществляться для двух типов капчи (Рис 17).

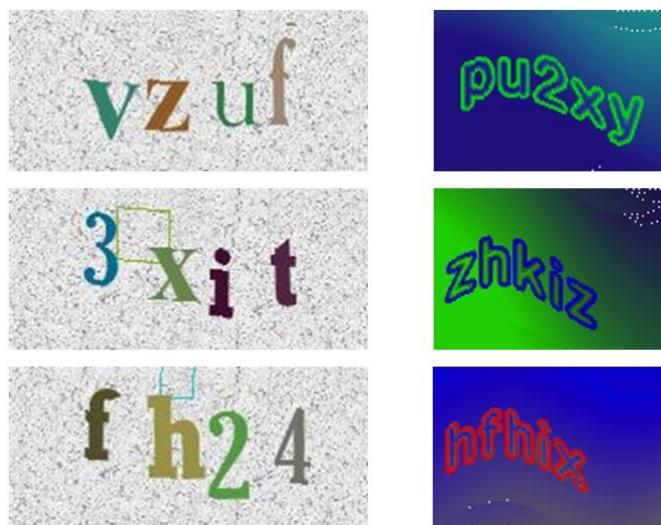


Рис. 17 Типы визуальной капчи для распознавания (слева тип 1, справа тип 2)

## **2.2 Описание метода распознавания капчи**

Опишем подробнее метод классификации, алгоритмы предварительной обработки изображений, методы сегментации и функцию классификации. Так как рассматривается несколько видов капчи для распознавания, рассмотрим отдельно методы работы с каждым из них.

Реализованный метод распознавания осуществляется для двух выбранных видов капчи и включает в себя различные методы для предварительной обработки изображений. Поэтому на самом первом шаге будет происходить классификация полученного сервисом изображения. Если изображение слишком велико (ширина  $> 700$  пикселей или высота  $> 600$  пикселей), то изображение не классифицируется как не являющееся капчей (изображения капчи обычно меньше) и не проходит дальнейшую обработку. Если эта проверка не выдала ошибки, изображение поступает на классифицирующую функцию, которая решает, каким типом визуальной капчи оно является. На этом шаге предполагается, что изображение точно принадлежит к какому-то одному из видов капчи.

### **2.2.1 Классификация**

Задача функции классификации состоит в принятии решения, к какому типу капчи отнести полученное изображение. На выходе имеем номер класса – класс 1 или класс 2 (см. рисунок 17). Можно заметить, что капче первого типа присущ зашумленный фон, причем большинство пикселей имеют белый, черный цвет или цвет оттенков серого. В то время как второй тип капчи представляет собой яркое цветное изображение с градиентом на фоне. Работа классификатора как раз основывается на найденном различии. Будем подсчитывать пиксели оттенков серого в полученном изображении, а исходя из полученного числа однозначно определять класс капчи. В зависимости от класса капчи начнем процесс предобработки.

### **2.2.2 Предобработка**

#### **2.2.2.1 Первый тип капчи**

Для избавления от части фона в изображении будем вычитать первый пиксель изображения, всегда являющийся частью фона, из остальных пикселей, а также средний цвет пикселя на изображении. Этот нехитрый способ даёт неплохой результат. Полученное изображение немного осветляется. Во избежание выхода за границы допустимых значений необходимо следить за получающимися значениями пикселей.

Применим алгоритмы *dilate* и *eroding*. Данные функции обычно применяются вместе, позволяя избавляться от мелкого шума на изображении. Принцип работы функций основывается на свертке изображения с некоторым ядром. Для функции *dilate* используется

функция максимизации. Проходя по изображению, ядро находит наибольшее значение пикселя, который в него попадает, и выдает это значение, в результате чего яркие области «растут». Для функции *eroding* наоборот, используется функция минимизации, в следствие чего светлые области изображения (обычно фон) становятся тоньше, в то время как темные зоны «расширяются». Грубо говоря, они делают шрифт тоньше или толще, поэтому после применения функций друг за другом, многие шумы пропадают за счет работы функции *dilate*. В результате работы получим изображение с меньшим количеством шумов на фоне, однако, часть необходимой информации (пикселей, изображающих символ) может быть потеряна. Применим полученное изображение (маску) для того, чтобы затемнить исходное. «Сложим» два изображения, увеличивая значения пикселей исходного в тех местах, где присутствуют черные пиксели маски. Таким образом, потеря информации не произойдет и пиксели символов увеличат значение. Затем применим к изображению пороговую функцию, позволяющую все пиксели со значениями ниже заданного покрасить в белый цвет, выше – в черный цвет.



*Рис. 18 Вычитание пикселей фона, наложения маски и применение пороговой функции*

На изображении до сих пор присутствуют элементы шума. Избавимся от них с помощью реализации двумерной системы непересекающихся множеств (DSU). С его помощью найдем связанные множества размера меньше 15 пикселей и, считая их шумом, покрасим в белый.

### 2.2.2.2 Второй тип капчи

Сложность второго типа капчи заключается в наличие градиента на фоновой части изображения. Количество различных цветов очень велико. Попытка находить определенное значение разницы цвета между соседними пикселями для выявления границы начала символов и окончания фона к успеху не привела. Также на изображении присутствуют некоторые белые точки небольших размеров, являющиеся своеобразным шумом, которые могут дать большую разницу с соседними пикселями. Для избавления от большого разнообразия цветов был реализован алгоритм k-средних (k-means). Будем разбивать множество различных цветов на  $k$  кластеров и находить наилучшее разбиение. Инициализация происходит случайными цветами. После завершения работы алгоритма имеем  $k$  различных цветов, которые лучше всего разбивают множество цветов исходного

изображения на группы. Затем происходит подсчет количества пикселей, принадлежащих каждому из  $k$  цветов. Если на изображении присутствует больше  $presents$  процентов пикселей данного цвета, то этот цвет классифицируется как фон и закрашивается белым. Значения  $k = 10$  и  $presents = 15$  подобраны эмпирически.



Рис. 19 Примеры предобработки трудных образцов капчи из тестовой выборки

Однако алгоритм на некоторых итерациях может относить буквы к фону. Чтобы этого избежать, значение  $presents$  пришлось немного занизить, а значение  $k$  завысить. Это повлекло за собой то, что иногда в углах изображения могут оставаться цвета градиента. Для очистки изображения используются функции *clear\_corners* и *bfs*. Первая функция проходит по кромке изображения и проверяет наличие там цвета, отличного от белого. Если этот цвет найден, то *bfs* рекурсивно красит все соседние символы того же обнаруженного цвета в белый цвет.

После предварительной обработки получаем черную строку символов на белом фоне. Можно заметить, что строка искажена с помощью какой-то синусоидальной функции. Были попытки подобрать коэффициенты для функции синуса таким образом, чтобы максимально выпрямить искаженную строку, однако на некоторых примерах функция лишь усугубляла искажение, поэтому было принято решение отказаться от использования функции восстановления и подавать на вход нейронной сети немного искаженные символы. Сегментация капчи этого типа производилась аналогичным образом. В конце предобработки также происходит масштабирование полученных изображений к размеру (28x28).

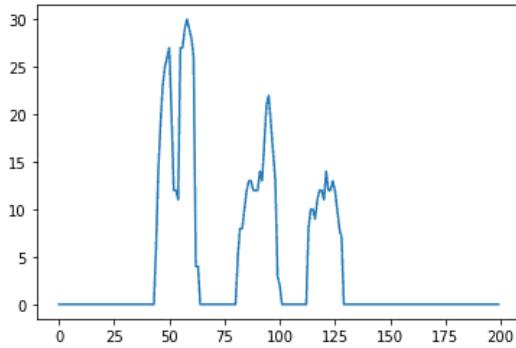


Рис. 20 Пример успешной и неудачной работы функции восстановления

Результат распознавания отображается на форме ответа реализованного веб-сервиса. Внешний вид формы ответа и результат работы представлен на рисунке.

### 2.2.3 Сегментация

Избавившись от шума, начнем процесс сегментации. В его основе будет лежать построение гистограмм количества черных пикселей в каждом «столбце» пикселей. Рассмотрим такую гистограмму для выбранного примера первого типа капчи, прошедшего этап предобработки.



*Рис.21 Гистограмма распределения черных пикселей по ширине изображения*

Заметны три значительных пика, разделенные нулевыми минимумами. Каждый пик отражает наличие буквы, каждый нулевой минимум соответствует расстоянию между символами. Будем считать, что буква начинается тогда, когда значение «столбика» перестаёт быть нулевым, и заканчивается тогда, когда столбик снова принимает значение нуля. Конечно, в некоторых случаях мы можем получить разделение буквы на несколько частей или, наоборот, соприкасающиеся буквы будут давать один сегмент вместо двух. Это вырожденные случаи. Чтобы избежать ошибки, будем объединять разрезы слишком маленькой ширины и проверять разрезы слишком большой ширины. После сегментации необходимо масштабирование изображения. Приведем каждое изображение к размеру (28x28), так как нейронная сеть принимает на вход изображение именно этого размера.

### 2.2.4 Распознавание и постобработка

После этого начинает происходить распознавание. Процесс распознавания разделен на несколько ранее описанных этапов. Это позволяет более четко определять ошибки, которые могут возникнуть в процессе распознавания и привести к неверному ответу, и способы их предотвращения в будущем.

Этап распознавания основывается на использовании свёрточной нейронной сети, которая получает на вход изображение размера (28x28) пикселей и выдает вектор из 28

элементов, где  $i$  –  $i$ -ый элемент ( $i = 0, \dots, 27$ ) отражает вероятность отнесения распознанного символа к  $i$  классу.

Процесс постобработки состоит из сбора ответов нейронной сети и формирование единой строки ответа.

### 2.3 Модель нейронной сети, используемая для распознавания капчи

При распознавании капчи будет использоваться свёрточная нейронная сеть. Для описания математической модели введем обозначения.

Под  $l \in [1; L]$  будем понимать рассматриваемый в данный момент слой нейронной сети, где  $L = 2a + 2, a \in \mathbb{Z}^+$  – количество скрытых слоёв в сети. За  $N^l$  обозначим количество карт признаков (фильтров) на слое  $l$ , а за  $f_l(\cdot)$  – функцию активации рассматриваемого слоя  $l$ . Также, под переменной  $\psi_n^l$  будем понимать  $n$ -ую карту признаков на слое  $l$ .

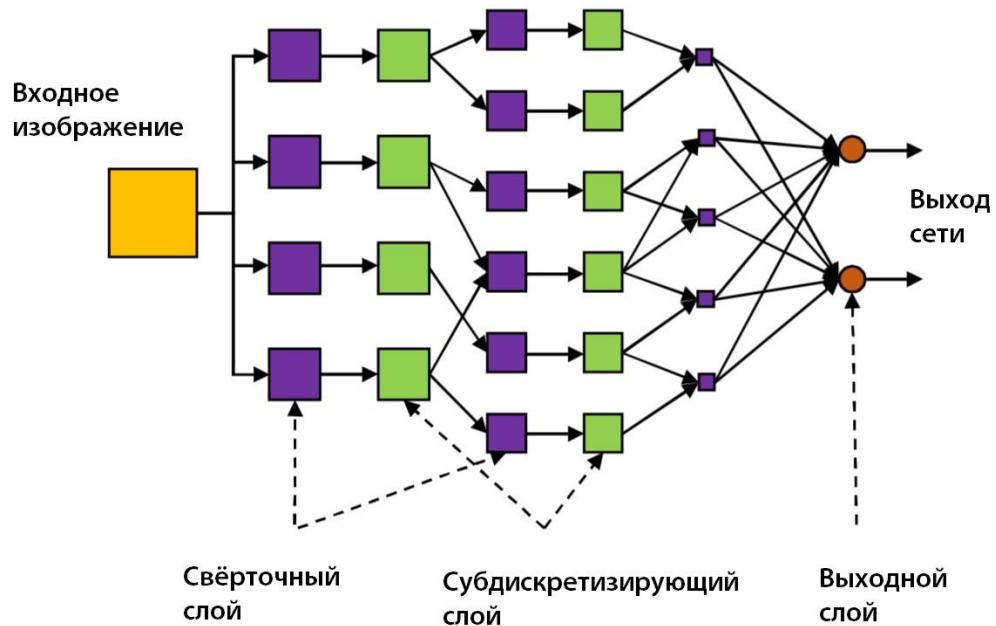


Рис. 22 Архитектура свёрточной нейронной сети

Введём в рассмотрение свёрточный слой  $l$ . В подобной архитектуре нейронной сети  $l$  принимается нечётным числом, то есть  $l = 1, 3, \dots, 2a + 1$ . Тогда для карты признаков  $n$  будет иметь место следующее:

- $w_{m,n}^l(i,j)$  – свёртка, применяемая к карте признаков  $m$  слоя  $(l - 1)$ , на слое  $l$  с картой признаков  $n$ ;
- $b_n^l$  – пороговые значения, присоединяемые к карте признаков  $n$  на слое  $l$ ;

- $V_n^l$  – список всех уровней слоя  $(l - 1)$ , которые соединяются с картой признаков  $n$  слоя  $l$ ;

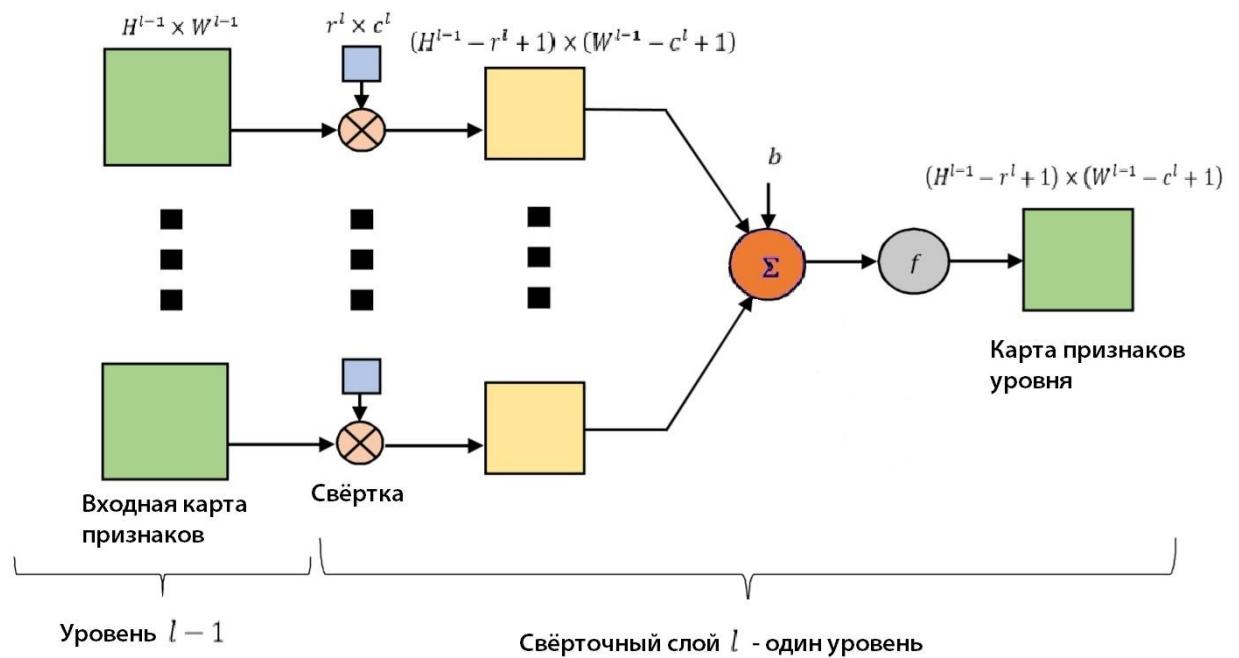
Таким образом, карта признаков  $n$  свёрточного слоя  $l$  будет вычисляться следующим образом:

$$y_n^l = f_l \left( \sum_{m \in V_n^l} y_m^{l-1} \otimes w_{m,n}^l + b_n^l \right), \quad (1)$$

где под оператором  $\otimes$  понимается математическая операция двумерной свёртки.

Предположим, что размер входных карт признаков  $y_m^{l-1}$  равен  $H^{l-1} \times W^{l-1}$ , а размер применяемой к ним свёртки  $w_{m,n}^l$  равен  $r^l \times c^l$ , тогда размер выходной карты признаков  $y_m^l$  вычисляется как:

$$(H^{l-1} - r^l + 1) \times (W^{l-1} - c^l + 1). \quad (2)$$



*Rис. 23 Схема свёрточного слоя  $l$*

Введём в рассмотрение субдискретизирующий слой  $l$  (пуллинг-слой). В свёрточной нейронной сети  $l$  принято принимать чётный числом, то есть  $l = 2, 4, \dots, 2a$ . Для карты признаков  $n$  введем следующие обозначения:  $w_{m,n}^k$  – фильтр, применяемый к  $n$  на слое  $l$ , и  $b_n^l$  – добавочное пороговое значение.

Далее разделим карту признаков  $n$   $(l - 1)$ -ого слоя на непересекающиеся блоки размером  $2 \times 2$  пикселя. Затем просуммируем значения четырёх пикселей в каждом блоке и в результате получим матрицу  $z_n^{l-1} = \{z_n^{l-1}(i, j)\}$ , элементами которой будут являться

соответствующие значения сумм. Таким образом, формула для вычисления значений элементов матрицы будет иметь следующий вид:

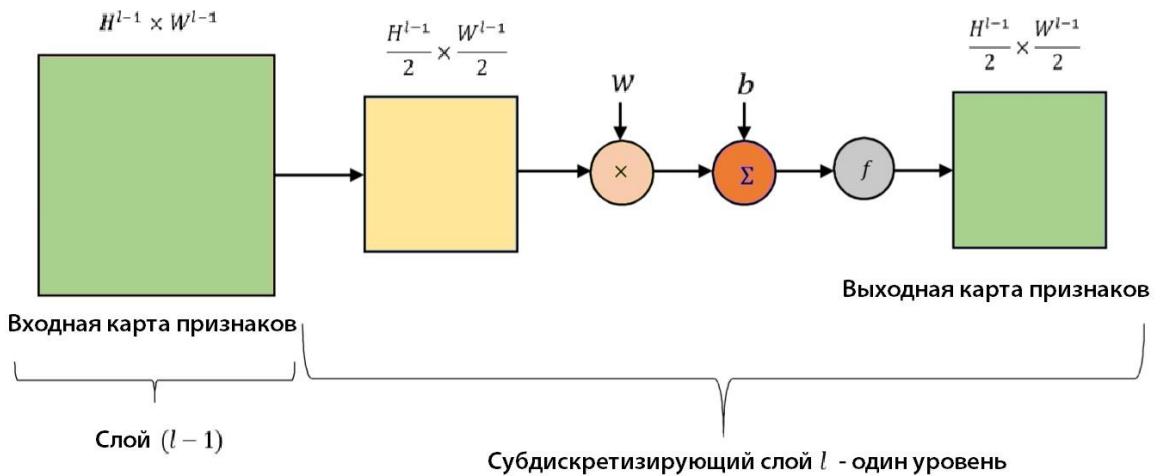
$$z_n^{l-1} = y_n^{l-1}(2i-1, 2j-1) + y_n^{l-1}(2i-1, 2j) + y_n^{l-1}(2i, 2j-1) + y_n^{l-1}(2i, 2j) \quad (3)$$

Карта признаков  $n$  субдискретизирующего слоя  $l$  вычисляется, как:

$$y_n^l = f_l(z_n^{l-1} \times w_{m,n}^l + b_n^l). \quad (4)$$

Благодаря представленным выше рассуждениям, становится возможным посчитать размер  $H^l \times W^l$  карты признаков  $y_n^l$  субдискретизирующего слоя  $l$ :

$$H^l = \frac{H^{l-1}}{2}, W^l = \frac{W^{l-1}}{2}. \quad (5)$$



*Рис. 24 Схема субдискретизирующего слоя  $l$*

Выходной слой  $L$  состоит из единичный нейронов. Примем за  $N^L$  – количество нейронов на данном слое. Как и при рассмотрении предыдущих слоёв, обозначим за  $w_{m,n}^L$  – фильтр, применяемый к карте признаков  $m$  последнего свёрточного слоя для получения перехода к нейрону  $n$  выходного слоя. Пусть  $b_n^L$  – пороговое значение, добавляемое к нейрону  $n$ .

Пользуясь введёнными обозначениями, получаем формулу для подсчета значения выходного нейрона  $n$ :

$$y_n^L = f_L(\sum_{m=1}^{N^{L-1}} y_m^{L-1} \otimes w_{m,n}^L + b_n^L). \quad (6)$$

Таким образом, выходом свёрточной нейронной сети является вектор следующего вида:

$$\mathbf{y} = [y_1^L, y_2^L, \dots, y_{N^L}^L]. \quad (7)$$

## 2.4 Метрики качества нейросетевой модели

В процессе обучения нейронной сети происходит оптимизация сразу нескольких параметров, которые и отражают качество обучения нейронной сети.

Такими параметрами является функция потерь (loss) - ошибка распознавания на конкретном примере, которая напрямую влияет на обучение, а также точность распознавания, которая показывает способность сети распознавать символы, но не влияющая на обучение.

Существует несколько метрик для оценки качества нейронной сети, однако для оценки качества многоклассовой классификации существует только точность распознавания (accuracy). Возможно также отслеживание точности для каждого отдельного класса (mean average precision, mAP).

Еще одной характеристикой, которая может рассматриваться при обучении сети, является время её обучения. Так как распознавание будет осуществлять уже обученная сеть, эта характеристика при проектировании архитектуры нейронной сети не будет учитываться.

## 2.5 Метрики качества метода распознавания

Под точностью метода распознавания будем понимать среднюю точность распознавания одного символа капчи. Другими словами, точность распознавания является отношением числа правильно распознанных символов к общему числу символов в зашифрованной в изображении строке.

Ошибка на каком-то этапе влечет ошибки на всех следующих этапах. Самый ранний этап, повлекший последовательность ошибок и неверный результат распознавания, считается пройденным с ошибкой.

В таблице 4 представлены формулы вероятностей ошибок на различных этапах процесса распознавания. Значения  $n_1, n_2, n_3$  равны количеству символов, которые обработались ошибочно на соответствующем этапе. Значения  $P_1, P_2, P_3$  равны вероятности того, что ошибка обработки данного символа появилась впервые на соответствующем этапе. Значение  $n$  отражает объем всей тестовой выборки (все символы капчи), значение  $P$  означает вероятность корректной работы всех этапов, которая привела к верному результату распознавания. Для расчета  $P$  приведены две формулы.

Таблица 4. Вероятность ошибки на разных этапах распознавания

Ошибка при предобработке	Ошибка при сегментации	Ошибка при распознавании	Символ распознан верно
$n_1$	$n_2$	$n_3$	$n - n_1 - n_2 - n_3$

$P1 = \frac{n1}{n}$	$P2 = \frac{n2}{n - n1}$	$P3 = \frac{n3}{n - n1 - n2}$	$P = 1 - P1 - (1 - P1) * P2 - (1 - P1) * (1 - P2) * P3$ $P = \frac{n - n1 - n2 - n3}{n}$
---------------------	--------------------------	-------------------------------	---

## 2.6 Выводы

В данном разделе была сформулирована задача распознавания визуальной капчи, выделены этапы и подзадачи.

Распознавание будет включать в себя процесс предварительной обработки, сегментации, распознавания свёрточной нейронной сетью и постобработки ответа.

Также была подробно рассмотрена модель свёрточной нейронной сети и виды слоёв, из которых она состоит. В качестве метрики качества построенной нейронной сети выбрана точность распознавания.

## **Раздел 3. Разработка веб-сервиса для распознавания капчи**

На данном этапе производился выбор и изучение языка и фреймворков для реализации веб-сервиса, произведена сама реализация и проверена работа полученного веб-сервиса.

### **3.1 Выбор языка и фреймворков для программной реализации веб-сервиса**

Фреймворт – это программная платформа, каркас, предназначенный для облегчения разработки программного проекта и объединения его компонентов. Для реализации веб-сервисов существует множество фреймворков, облегчающих разработку и создание сложных сайтов и веб-приложений.

Реализация нейросети и дополнительных методов и алгоритмов происходила на языке Python, поэтому в качестве языка для создания веб-сервиса также был выбран Python. Рассмотрим самые распространенные Python-фреймворки.

#### **1. Django**

Является самым популярным на данный момент фреймвортом на языке Python. Этот инструмент удобно использовать для разработки сайтов, работающих с базами данных. Django подходит для разработки высоконагруженных веб-приложений. Это возможно благодаря архитектуре MVC (модель-представление-контроллер / model-view-controller). [20]

Архитектура позволяет организовывать блоки кода, отвечающие за решение разных задач.

Компоненты MVC:

- Модель (model) — этот компонент отвечает за данные, а также определяет структуру приложения. Здесь содержится основная логика приложения, определяется список решаемых задач, происходит манипуляция данными.
- Представление (view) — этот компонент отвечает за взаимодействие с пользователем. То есть код компонента view определяет внешний вид приложения и способы его использования.
- Контроллер (controller) — этот компонент отвечает за связь между model и view. Код компонента controller определяет, как сайт реагирует на действия пользователя. По сути, это мозг MVC-приложения, который содержит организационную логику для самого приложения.

В Django реализован мощный движок шаблонов и собственный язык разметки.

Фреймворт обладает множеством достоинств, таких как расширяемость, быстрое и удобное взаимодействие с базами данных, и множеством недостатков, главный из которых – избыточность возможностей для небольших проектов. По этой причине на фоне Django эффективно выделяется фреймворт Flask, являющийся так называемым микрофреймвортом.

## 2. Flask

Фреймворк представляет собой минималистичный каркас, предоставляющий лишь самые базовые возможности. Ключевое отличие состоит в гибкости и простоте, которая позволяет выбирать, как реализовывать те или иные вещи.

Flask позволяет разделять большой, сложный проект на набор независимых модулей (blueprint), что позволяет создавать фрагменты приложения независимо.

В отличие от Django Flask не имеет модели данных, позволяющих связывать таблицы базы данных с классами на языке программирования. Приложения для Flask в основном представляют собой одностраничные приложения. Это хороший выбор для небольших и средних сайтов.

В январе 2017 в StackOverflow насчитывалось 2631 вопросов о Django, и лишь 575 о Flask. Оба фреймворка набирают популярность, если брать StackOverflow в качестве мерил[21].

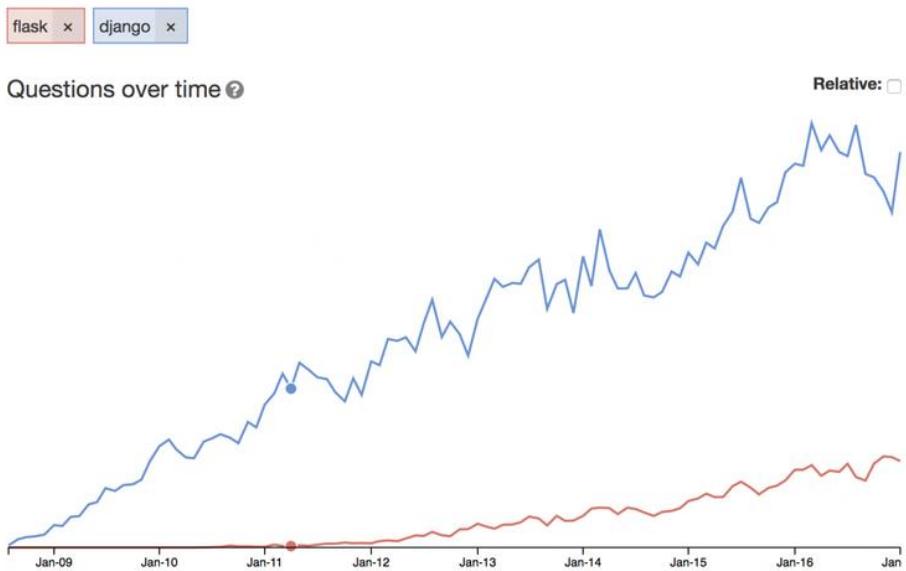


Рис.25 Количество вопросов, связанных с фреймворками, с 2009 по 2017гг.

Таким образом, простота поставленной задачи и небольшая необходимая функциональность веб-сервера позволяет не рассматривать Django как подходящий вариант фреймворка для использования. Готовая основа, предоставленная фреймворком Django, в данном случае может лишь бесполезно расширять сервис, что позволяет сделать выбор в пользу Flask.

Однако работа с фреймворком требует больших временных ресурсов для знакомства с его устройством, принципами работы, настройки, запуска и особенностями работы с ним. В данной работе веб-сервис должен представлять собой простую модель из

двух страниц (форм) – форма отправки и форма ответа. Создание сайта возможно без использования фреймворка, поэтому был сделан выбор в пользу написания исходного кода с нуля на чистом языке программирования без использования каких-либо платформ, а именно – создание динамических страниц при помощи CGI-скриптов (исполняемых файлов, которые выполняются веб-сервером, когда в URL запрашивается соответствующий скрипт), что позволит сэкономить временной ресурс для более качественной реализации метода распознавания. Ограниченный список требований к работе веб-сервиса позволяет максимально ускорить его разработку. Форма отправки реализована на HTML. Передача изображения в форму ответа происходит в виде POST-запроса.

### **3.2 Программная реализация моделей и алгоритмов**

#### **3.2.1 Спецификация требований**

- Для решения задачи необходимо реализовать веб-сервис, который принимает изображение пользователя и возвращает строку-ответ для полученного изображения.
- Веб-сервис может обладать минимальным наполнением и функционалом.
- Пользователь должен видеть этапы работы метода распознавания, чтобы оценивать корректность работы метода и в случае ошибки находить этап распознавания, который её вызвал.
- Критическими параметрами системы является правильность её работы и время её работы.

#### **3.2.2 Архитектура системы**

Веб-сервис представляет собой локальный хост, при запуске которого начинается работа сервера. После запуска сервер ожидает отправку изображения пользователем. Если хост запущен, то сервер доступен для работы.

Чтобы осуществить распознавание полученного сервером изображения, оно сохраняется на сервер и последовательно обрабатывается. Выполнение этапов распознавания осуществляется с помощью последовательного запуска скриптов. Каждый из них ищет сохраненное изображение, открывает его, обрабатывает и сохраняет результат, либо возвращает ответ. Работа сервиса зависит от целостности проекта, так как полученное изображение, с которым последовательно работают скрипты для формирования ответа, должно находиться в определенных подкаталогах проекта.

Для работы скрипта по распознаванию символов сегментированные изображения также сохраняются на сервер. Перед каждым новым циклом работы программы сохраненные ранее изображения удаляются.

Для отслеживания этапов распознавания пользователем ему будет представлено обработанное и сегментированное изображение и результат распознавания каждого изображения нейронной сетью. Таким образом, пользователь сможет определить, на каком этапе распознавания происходила неправильная работа, повлекшая ошибку.

Чтобы предотвратить возможную ошибку пользователя при выборе и отправке изображения, установлена проверка размера полученного сервером изображения. Так как никаких требований к безопасности веб-сервиса не было установлено, то работа по выявлению потенциальных программных уязвимостей выполнено не было.

Архитектура проекта создана с целью визуализации работы метода распознавания.

В следствие локальности сервиса делается допущение, что на машине пользователя возможен запуск скриптов. Это допущение может повлечь необходимость в установке дополнительных программных компонентов или библиотек на том устройстве, на котором будет происходить запуск веб-сервиса.

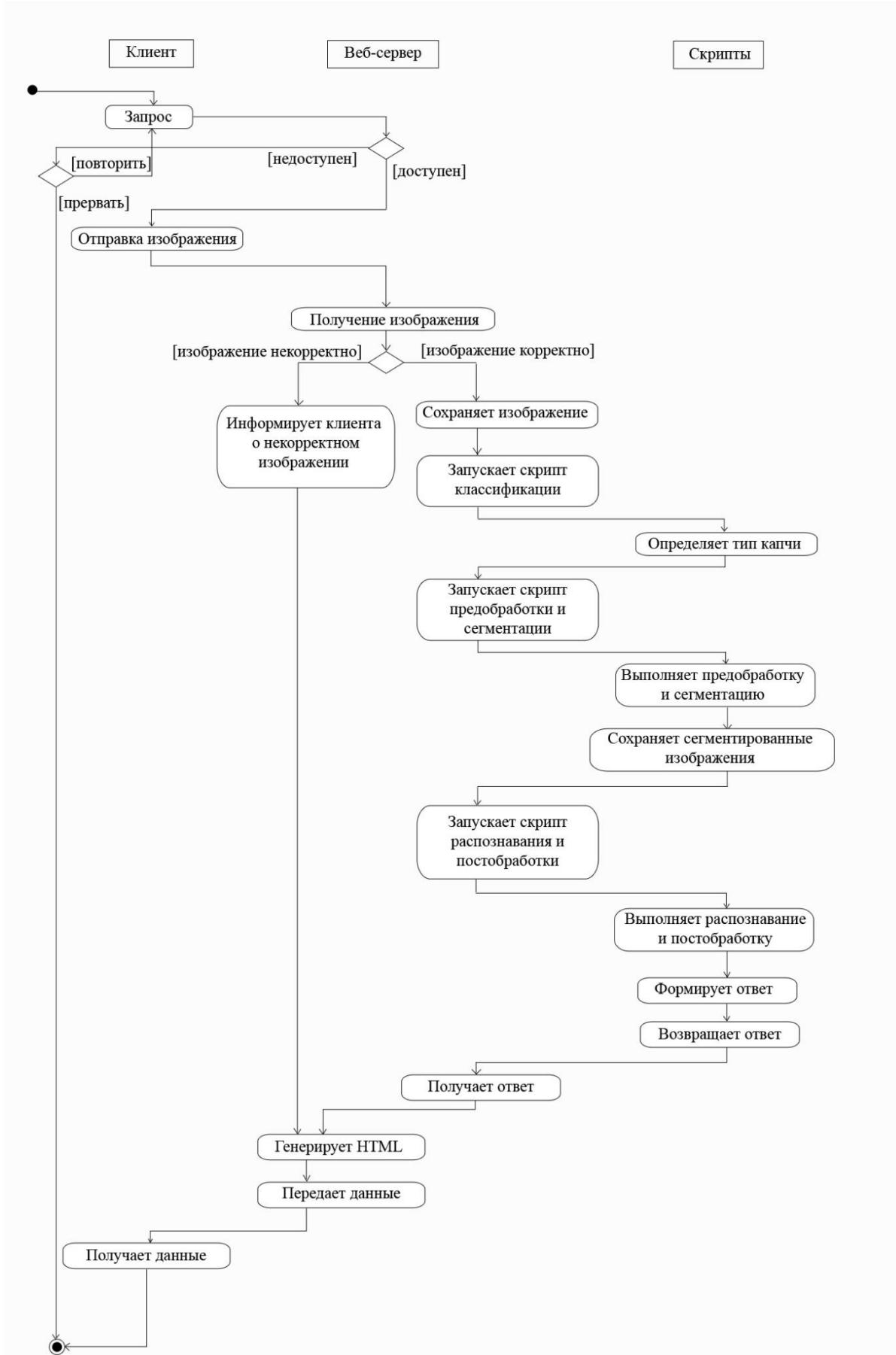


Рис.26 Диаграмма деятельности (активности) UML

### 3.2.3 Описание функционала

Возможности системы состоят в распознавании определенных типов капчи. Предполагается, что капча имеет определенные характеристики, соответствующие характеристикам поддерживаемых капч. Другими словами, полученное изображение капчи обладать определенным размером и форматом. У первого типа капчи размер изображение составляет 200x90 пикселей, формат “.jpeg”. Второй тип капчи имеет размер 130x90 пикселей и формат “.png”. Предоставление изображения капчи в других форматах может повлечь неверную работу алгоритма предобработки и стать причиной появления ошибки при распознавании. Слишком большой размер может стать причиной долгой обработки или отказа системы воспринимать изображение как корректное (содержащее капчу).

Функционал системы заключается в способности производить распознавание одного изображения при одном цикле работы.

### 3.2.4 Описание программной реализации

Для программной реализации выбран язык Python 3. Для работы с изображениями используются библиотеки Python OpenCV и Pillow. Библиотека OpenCV обладает встроенными функциями фильтрации и уже реализованными алгоритмами работы с изображениями, которые могут облегчить работу. Библиотека Pillow предоставляет удобные инструменты для реализации собственных алгоритмов и настройки их работы под конкретную задачу. Для построения, обучения и использования сверточных нейронных сетей используются библиотеки Keras и Tensorflow.

Далее будет представлена документация к каждому реализованному скрипту. Диаграммы структуры скрипта показывают вложенность функций, реализованных в соответствующем скрипте. Предполагается, что функции располагаются друг под другом, символизируя порядок вызова. Другими словами, первая вызванная функция будет располагаться в верхней части изображения, последняя – в нижней части. Изображение одной функции внутри другой означает вызов второй в процессе выполнения первой. С увеличением степени вложенности функции цвет блока функции становится темнее.

#### 3.2.4.1 Классификатор

- `def Classifier(image)` – соотносит изображение с одним из рассматриваемых классов. Функция принимает на вход массив пикселей, подсчитывает количество серых пикселей в нем и на основании полученного числа определяет класс, к которому стоит отнести изображение. Границным значением при принятии решения является 5000 пикселей. В процессе работы вызывает функцию `Grey_scale(pixel)`. Возвращает номер класса.

- `def Grey_scale(pixel)` – определяет, принадлежит ли цвет принятого на вход пикселя оттенкам серого. Если значения разницы трех каналов пикселя ниже граничного `border = 25`, функция возвращает `True`, иначе `False`.

### 3.2.4.2 Предобработка и сегментация первого типа капчи

- `def get_path_to_save(path)` – производит обработку полученной строки `path` для формирования пути сохранения обработанных, сегментированных и отмасштабированных изображений символов.

• `def open_color(path)` – загружает изображение по указанному пути `path`. Возвращает загруженное изображение. Для работы необходимо подключить библиотеку OpenCV.

• `def image_show(image)` – отображает полученное изображение. Функция использовалась в процессе написания кода и отслеживания процесса предобработки.

• `def noize_line(image)` – очищение изображения от шума и выполнение аддитивной пороговой операции для изображения с помощью OpenCV. Последовательно применяет функции реализованные функции `dilate` и `erode`. Функция `dilate` применяет операцию максимизации, что приводит к тому, что яркие области в изображении "растут" (поэтому происходит расширение фона и уменьшение символов и областей шума). Функция `erode` работает аналогично, но применяет операцию минимизации, в следствие чего светлые области изображения становятся тоньше, в то время как темные зоны становятся больше. Символы возвращаются к исходной толщине, но шум, потерянный после функции `dilate`, исчезает.

• `def cut_luz(image)` – выполняет пороговую операцию по заданному значению. Пиксели со значением выше заданного окрашиваются в белый.

• `def no_background(image)` – вычитает значение первого пикселя изображения из последующих.

• `def dominant_color(image)` – определяет преобладающий цвет на изображении. Использует встроенные функции библиотеки OpenCV.

• `def preproseed(image)` – объединяет несколько ранее описанных функций. Из изображения вычитается значения преобладающего цвета и цвета первого пикселя с помощью функции `dominant_color(image)` и `no_background(image)`. После этого определяется преобладающий цвет и выполняется пороговая операция. Полученное таким образом изображение «накладывается» на исходное с помощью функции `addWeighted` библиотеки OpenCV. Затем применяются функции `cut_luz(image)`, `noise_line(image)`. Функция возвращает обработанное изображение.

- class DSU\_2D – реализует двумерную систему непересекающихся множеств.

Содержит два массива d, p. Класс включает в себя три метода – init(self), get(self, x, y), uni(self, a, b). При создании экземпляра класса функция init заполняет массивы d, p начальными значениями. Функция get(self, x, y) возвращает представителя элемента, функция uni(self, a, b) выполняет объединение множеств.

- def generate\_steps(delta, flag) – создает и возвращает массив пар, заполненный значениями от -delta до delta с шагом 1. Значение flag определяет, будет ли входить значение (0, 0) в массив.

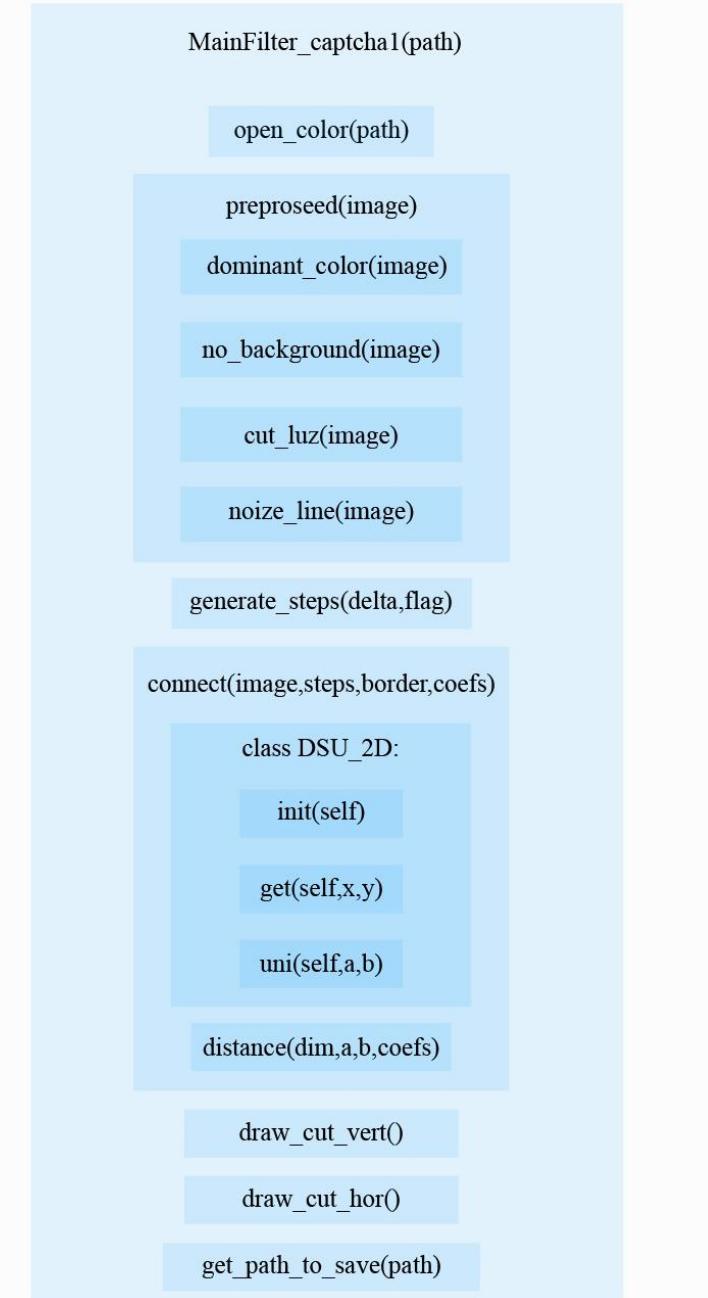
- def distance(dim, a, b, coefs) – вычисляет разницу между элементами массивов a и b, умноженную на элементы массива весов coefs. Переменная dim показывает размерность массивов. Возвращает просуммированную величину разницы.

- def connect(image, steps, border, coefs) – выполняет создание экземпляра класса DSU\_2D на массиве image. После этого для каждого элемента (x, y) массива image и каждого элемента пар (dx, dy) массива steps вычисляется разница между значениями пикселей (x, y) и (x + dx, y + dy) с помощью вызова функции distance(dim, a, b, coefs). Если значение разницы меньше border, то пиксели (x, y) и (x + dx, y + dy) объединяются в одну компоненту связности класса DSU\_2D с помощью функции uni((x, y), (x + dx, y + dy)). Функция возвращает реализованный экземпляр класса.

- def MainFilter\_captcha1(path) – объединяет описанные ранее функции. Эта функция вызывается в качестве основной функции предобработки, сегментации и масштабирования капчи первого типа. Функция загружает изображение, расположенное по указанному пути с помощью функции open\_color(path), обрабатывает его с помощью функции preproseed(image), выделяет связные области с помощью создания структуры данных DSU (Disjoint Set Union / система непересекающихся множеств) и удаляет их. Связанная область классифицируется как шум и удаляется, если ее размер не превосходит 15 пикселей. Затем происходит сегментация изображения. Сегментация происходит описанным ранее способом. Каждое изображение масштабируется с помощью функции resize(size) библиотеки OpenCV и сохраняется по пути, полученному функцией get\_path\_to\_save(path). Формат для сохранения “.jpeg”.

- def draw\_cut\_vert() – рисует на изображении линии серого цвета (170, 170, 170), отображающие границу будущих вертикальных разрезов. Функция использовалась в процессе написания кода для сегментации.

- def draw\_cut\_hor() – рисует на изображении линии серого цвета (170, 170, 170), отображающие границу будущих горизонтальных разрезов. Функция использовалась в процессе написания кода для сегментации.



*Рис.27 Диаграмма структуры скрипта предобработки первого типа капчи*

### 3.2.4.3 Предобработка и сегментация второго типа капчи

- def get\_path\_to\_save(path) - производит обработку полученной строки path и формирует путь для сохранения обработанных, сегментированных и отмасштабированных изображений символов.
- def generate\_steps(delta, flag) – аналогично, формирует массив пар, состоящий из значений от -delta до delta с шагом = 1.
- def connect(image, steps) – связывает области изображения с одинаковыми цветами с помощью обхода в ширину. Возвращает массив из списка пикселей одного цвета.

Вызывает внутреннюю функцию `connect_bfs(x, y, color)`, реализующую обход в ширину на изображении.

- `def clear_corners(image)` – обходит кромку изображения шириной в 3 пикселя и ищет пиксели не белого цвета. Если такой пиксель найден, то окрашивает в белый цвет этот пиксель и всю найденную связанную с ним область того же цвета. Для поиска связанной области вызывается внутренняя функция `clear_bfs(x, y, color)`, реализующая обход в ширину.
- `def k-means(list_of_pixels, k)` – реализует алгоритм k-средних. Внутренняя функция `get_class(color)` определяет номер класса  $i$ ,  $i = 1..k$ , к которому принадлежит цвет `color`. Принимает на вход массив пар «ключ, значение», где в качестве ключа выступает цвет, в качестве значения – количество пикселей этого цвета на изображении, и значение `k`. Это множество пар разбивается на `k` классов, проинициализированных на первом шаге случайными цветами. После этого для каждого пикселя вычисляется ошибка по каждой компоненте цвета. К величине `error` прибавляется среднее значение ошибки по каждой компоненте цвета в каждом классе. Центры кластеров вмещаются на среднее значение ошибки по каждой компоненте цвета. Если величина `error` равна нулю, алгоритм прекращает работу. Возвращает словарь, где каждому цвету исходного изображения ставится в соответствие цвет класса, к которому был отнесен данный цвет с помощью алгоритма.
- `def filter_image(path)` – загружает изображение и применяет к нему алгоритм k-средних с помощью функции `k-means(list_of_pixels, k)`. Окрашивает изображение в соответствие с выбранными средними цветами. Определяет процентное соотношение каждого из средних цветов на изображении. Если какого-то цвета присутствует более 15 процентов, то он включается в список фоновых цветов. Для избавления от белых точек, белый цвет также считается фоновым. Все пиксели фонового цвета окрашиваются в белый. Функция возвращает перекрашенное изображение.
- `def main_filter_captcha_2(path)` – объединяет в себе все описанные выше функции, а также выполняет сегментацию. Вызывает функцию `clear_corners(filter_image(path))`, затем выполняет пороговое отсечение, разделяя пиксели на белые и небелые. Все небелые окрашивает в черный. Алгоритм k-средних инициализируется случайными цветами, поэтому применяя его к одному и тому же изображению можно получить разный результат. Если на изображении присутствует больше 95% или меньше 5% черных пикселей, оно обрабатывается еще раз. После этого функция `connect(image, steps)` очищает изображение от связных областей, которые по размеру не превышают 10 пикселей. После этого процесс предобработки завершается и происходит процесс сегментации. Изображения обрезаются с помощью функции `crop`, входящей в библиотеку `Pillow`.

масштабируется и сохраняется по пути, полученному функцией `get_path_to_save(path)`. Формат для сохранения “`.jpeg`”.

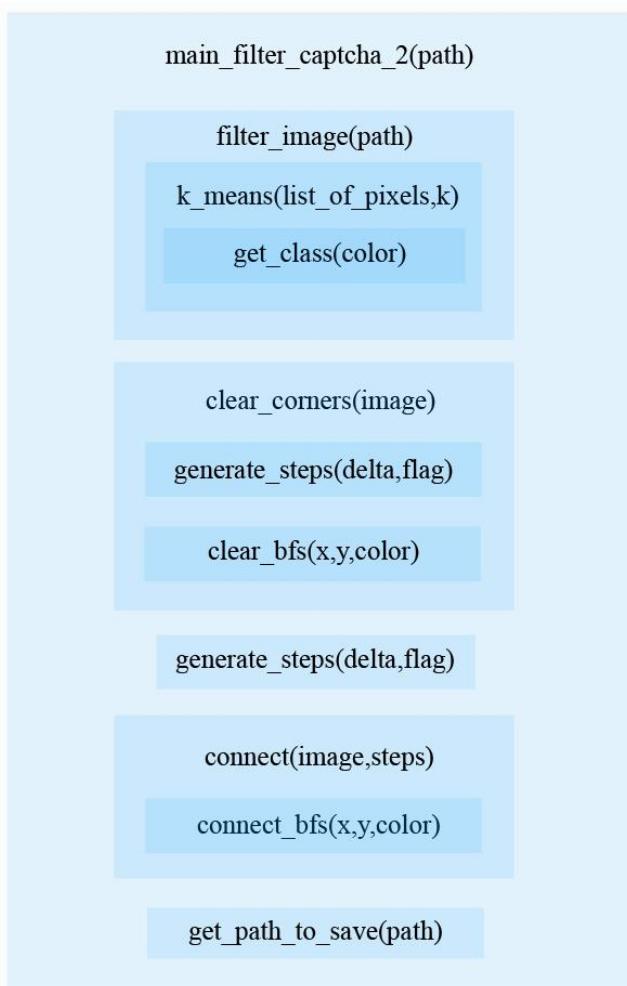


Рис.28 Диаграмма структуры скрипта предобработки второго типа капчи

#### 3.2.4.4 Распознавание и постобработка

- `def recognition1()` – загружает обученную модель нейронной сети и выполняет распознавание всех сохраненных изображений символов. Названия изображений состоят из цифр начиная от нуля. Открытие и распознавание происходит в лексикографическом порядке, поэтому сортировка и дополнительная постобработка не требуется. Возвращает строку ответа.
- `def recognition2()` – работает аналогично функции `recognition1()`. Загружает другую модель свёрточной нейронной сети, обученную на капче второго типа.

#### 3.2.5 Описание тестирования

Для проведения тестирования и оценки качества работы реализованного метода было выполнено распознавание сгенерированной тестовой выборки. В процессе тестирования

было замечено, что в процессе работы алгоритма k-средних в редких случаях результирующее изображение содержит все пиксели в белом цвете, либо все пиксели в черном цвете. Для устранения ошибки было принято решение вызывать алгоритм повторно, так как случайная инициализация центров кластеров может дать более удачный результат.

Достигнутая степень успешного распознавания составляет 91,6 процента для капчи первого типа и 78,2 процента для капчи второго типа. Ошибки в ответах возникают преимущественно из-за ошибок при сегментации. При распознавании второго типа капчи также присутствует небольшая вероятность неправильного распознавания символа нейронной сетью.



Рис. 29 Внешний вид формы ответа веб-сервиса

### 3.3 Выводы

В данном разделе описаны реализованные алгоритмы работы с разными выбранными типами визуальной капчи, описан выбранный способ создания веб-сервиса.

## **Раздел 4. Экспериментальные исследования метода распознавания капчи**

В данном разделе приводятся результаты экспериментальных исследований метода распознавания капчи с помощью конструирования нейронных сетей с различными архитектурными параметрами, а также выбор параметров сети для получения сети с оптимальным показателем работы, таким как точность распознавания. В связи с тем, что результат обучения нейронных сетей во многом определяется размером обучающей выборки, была создана размеченная выборка объёмом 28000 капч (1000 изображений на символ), и на этой выборке продемонстрировано влияние различных параметров архитектуры сети на результат работы. Проведённые исследования позволили определить оптимальную нейросеть по выбору архитектуры.

После этого было произведено сравнение результатов работы на выборках разных объемов для установления лучшего количества обучающих примеров.

В данном разделе также описан способ получения обучающей выборки и её размер.

### **4.1 Описание обучающих выборок и способа их получения**

Свёрточная нейронная сеть для хорошего обучения требует большого объема данных. Опишем способы получения обучающей выборки для каждого класса капчи.

Существует несколько способов создания обучающей выборки. Получение выборки с помощью парсинга сайтов и самостоятельной разметки, разметки с помощью существующих платных сервисов, расширение объема данных на основе небольшого имеющегося объема данных методом дополнительных искажений. Каждый из этих способов влечет некоторые проблемы, связанные либо с большим количеством потраченного времени и сил, либо потраченных денег, либо с плохим обучением нейронной сети на созданных самостоятельно данных. Поэтому в качестве способа получения обучающей выборки был выбран поиск существующих программных компонентов, которые генерируют капчу на реальных веб-страницах. Несколько найденных PHP-файлов были созданы для интеграции в частные сайты для обеспечения защиты от взломов. Файлы были немного изменены для того, чтобы созданное изображение не просто отображалось на локальном хосте при запуске PHP-файлов, но и сохранялось на носитель под названием, идентичным строке-ответу, зашифрованному в изображении. Затем был поставлен счетчик для генерации сразу нескольких изображений при одном запуске. Таким образом было сгенерировано две выборки по 84000 изображений в каждой. Капча второго типа включает в себя 5 символов, капча первого типа переменной длины – от 3 до 4 символов.

После генерации, предобработки, сегментации и масштабирования полученной обучающей выборки получим изображения одиночных символов размера (28x28). Эти изображения будем подавать на вход нейронной сети для обучения. Пример полученных изображений для символа «р» представлены на рисунке ниже.



Рис.30 Пример изображений обучающей выборки. Слева капча 1, справа капча 2

Таким образом была получена выборка, которая не потребовала больших человеческих или материальных ресурсов, способная обеспечить хороший уровень обучения нейронной сети.

## 4.2 Экспериментальное исследование влияния архитектурных параметров нейросетевых моделей на точность распознавания

Рассмотрим влияние архитектуры свёрточной нейронной сети на точность распознавания. Для исследования экспериментальной зависимости точности распознавания от параметров свёрточной нейронной сети было построено и обучено несколько нейросетей с различной архитектурой, а именно - с разным количеством слоёв, фильтров в слое и размерами фильтров.

В качестве оптимизатора использовался класс *Adadelta*, одним из преимуществ которого является самостоятельное адаптивная скорость обучения, что исправляет позволяет производить обучение с различной скоростью и не влечет постепенное затухание обучения, вызванное классическим постоянным снижением скорости обучения на протяжении всего обучения. Для сохранения граничных признаков используется *padding = same*. Активационной функцией является сигмоида. У выходного слоя функцией активации является *softmax*, выдающий вероятности отнесения изображения к определенному классу.

Обучение происходило на выборке размера 36400 (по 1300 на каждый класс), где три четверти является обучающей выборкой (27300 / 975) и одна четверть является контрольной (тестовой) выборкой (9100 / 325).

### 4.2.1 C5\*20-MP2S2-C5\*50-MP2S2-FC300-FC28 (сеть 1)

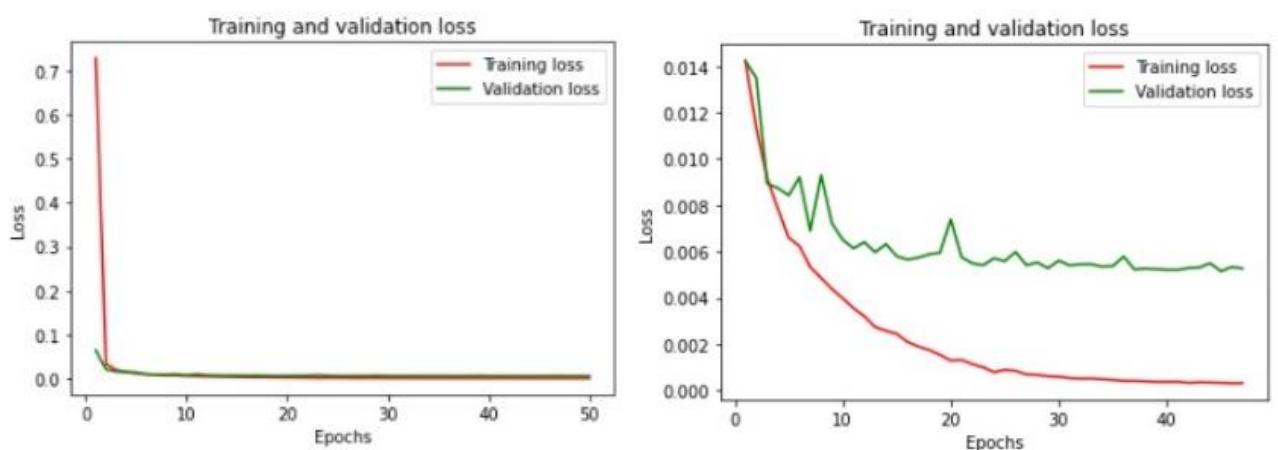
Первая свёрточная нейронная сеть состояла из двух слоев и имела конфигурацию C5\*20-MP2S2-C5\*50-MP2S2-FC300-FC28. Здесь C5\*20 обозначает слой свертки с размером ядра = 5, шагом (stride) = 1 и количеством фильтров = 20; MP2S2 обозначает субдискретизирующий слой (max-pooling layer) с размером ядра 2 и шагом = 2, FC300

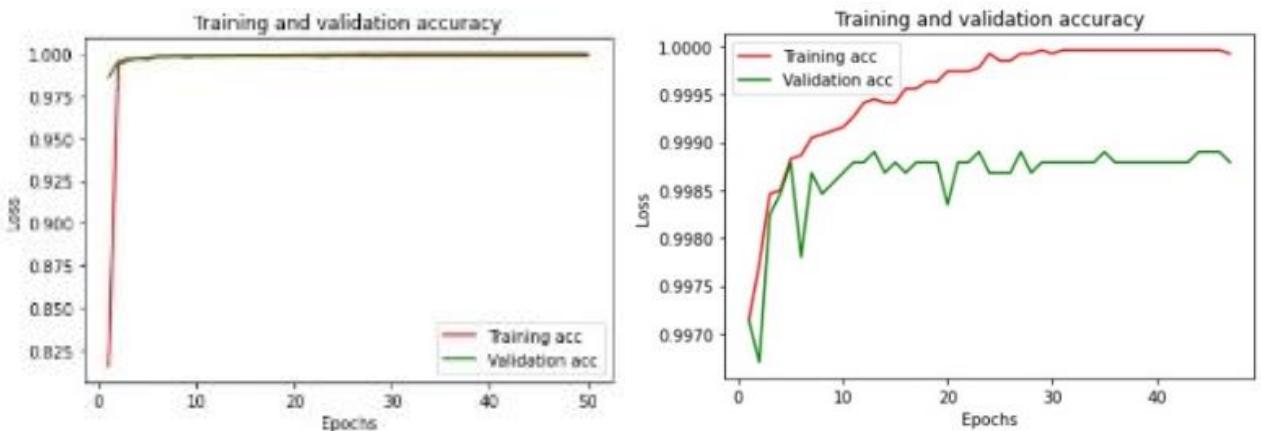
представляет полносвязный слой из 300 нейронов, и выходной слой, выдающий вектор размера (28x1). Обучение происходило в течение 50 эпох с переменным коэффициентом обучения.

Полученные результаты приведены на рисунке 31 и 32. Можно заметить, что после 8 эпохи точность перестает увеличиваться.

```
Epoch 1/50
- 17s - loss: 0.7290 - accuracy: 0.8157 - val_loss: 0.0637 - val_accuracy: 0.9862
Epoch 2/50
- 19s - loss: 0.0331 - accuracy: 0.9938 - val_loss: 0.0204 - val_accuracy: 0.9954
Epoch 3/50
- 17s - loss: 0.0189 - accuracy: 0.9961 - val_loss: 0.0156 - val_accuracy: 0.9970
Epoch 4/50
- 17s - loss: 0.0142 - accuracy: 0.9971 - val_loss: 0.0142 - val_accuracy: 0.9971
Epoch 5/50
- 17s - loss: 0.0113 - accuracy: 0.9977 - val_loss: 0.0135 - val_accuracy: 0.9967
Epoch 6/50
- 18s - loss: 0.0092 - accuracy: 0.9985 - val_loss: 0.0089 - val_accuracy: 0.9982
Epoch 7/50
- 18s - loss: 0.0078 - accuracy: 0.9985 - val_loss: 0.0087 - val_accuracy: 0.9985
Epoch 8/50
- 20s - loss: 0.0066 - accuracy: 0.9988 - val_loss: 0.0084 - val_accuracy: 0.9988
Epoch 9/50
- 18s - loss: 0.0062 - accuracy: 0.9989 - val_loss: 0.0092 - val_accuracy: 0.9978
Epoch 10/50
- 18s - loss: 0.0053 - accuracy: 0.9990 - val_loss: 0.0069 - val_accuracy: 0.9987
Epoch 11/50
- 18s - loss: 0.0049 - accuracy: 0.9991 - val_loss: 0.0093 - val_accuracy: 0.9985
Epoch 12/50
- 17s - loss: 0.0044 - accuracy: 0.9991 - val_loss: 0.0072 - val_accuracy: 0.9986
Epoch 13/50
- 17s - loss: 0.0040 - accuracy: 0.9992 - val_loss: 0.0065 - val_accuracy: 0.9987
Epoch 14/50
- 17s - loss: 0.0036 - accuracy: 0.9993 - val_loss: 0.0061 - val_accuracy: 0.9988
Epoch 15/50
- 17s - loss: 0.0032 - accuracy: 0.9994 - val_loss: 0.0064 - val_accuracy: 0.9988
```

Рис. 31 Обучение сети 1 в течение первых 15 эпох





*Рис.32 Графики функции потерь и точности сети 1*

После 3 эпохи появляется небольшое переобучение (приводя в результате к большой разнице между тренировочной и валидационной потерей) на 3 эпохе имеем точность 0,997 или 99,7%. Это лучший результат, который показала сеть до начала переобучения.

#### 4.2.2 C5\*50-MP2S2-C5\*100-MP2S2-FC300-FC28 (сеть 2)

Рассмотрим следующую нейронную сеть (сеть 2). Увеличим количество фильтров на первом и втором слое и посмотрим, на что это может повлиять. Все остальные параметры оставим без изменения. Имеем конфигурацию второй сети: C5\*50-MP2S2-C5\*100-MP2S2-FC300-FC28.

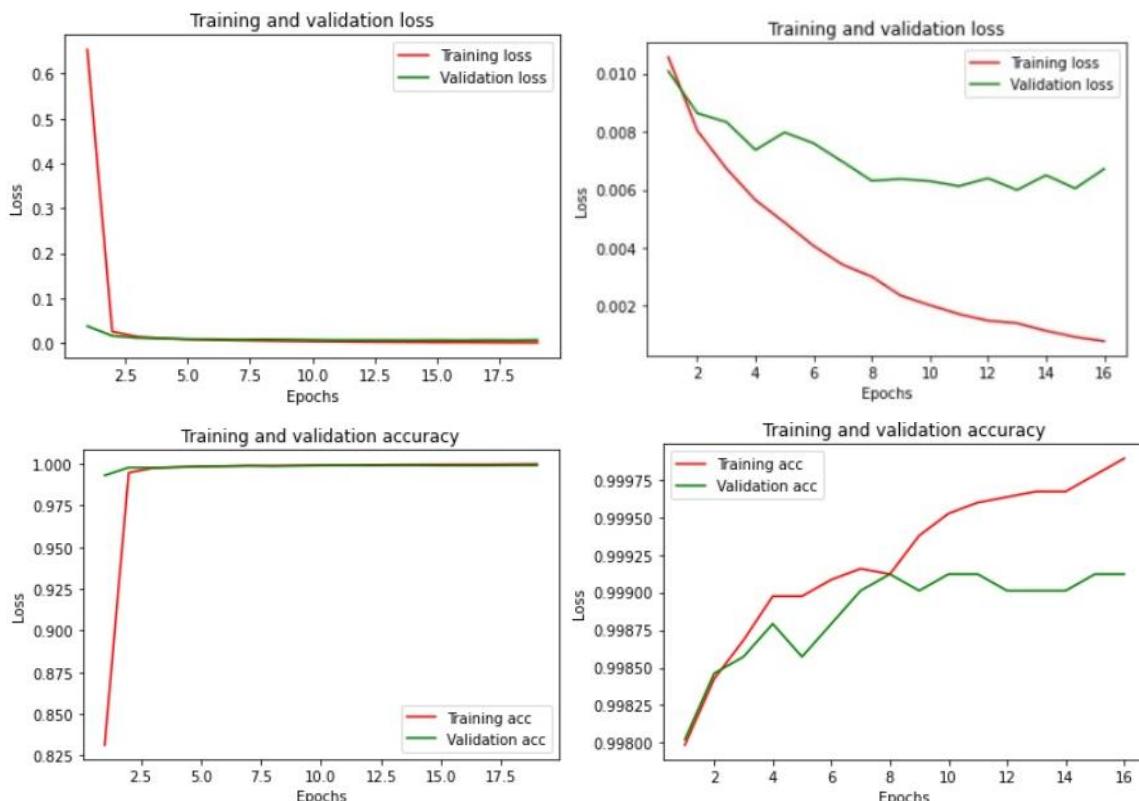
Обучение проводим в течение 20 эпох. После 11 эпохи точность перестает увеличиваться. Функция потерь также перестает уменьшаться после 11 эпохи.

```

Epoch 1/20
- 39s - loss: 0.6534 - accuracy: 0.8313 - val_loss: 0.0376 - val_accuracy: 0.9931
Epoch 2/20
- 41s - loss: 0.0252 - accuracy: 0.9947 - val_loss: 0.0157 - val_accuracy: 0.9978
Epoch 3/20
- 39s - loss: 0.0140 - accuracy: 0.9974 - val_loss: 0.0113 - val_accuracy: 0.9977
Epoch 4/20
- 38s - loss: 0.0106 - accuracy: 0.9980 - val_loss: 0.0101 - val_accuracy: 0.9980
Epoch 5/20
- 41s - loss: 0.0080 - accuracy: 0.9984 - val_loss: 0.0086 - val_accuracy: 0.9985
Epoch 6/20
- 42s - loss: 0.0067 - accuracy: 0.9987 - val_loss: 0.0083 - val_accuracy: 0.9986
Epoch 7/20
- 42s - loss: 0.0056 - accuracy: 0.9990 - val_loss: 0.0074 - val_accuracy: 0.9988
Epoch 8/20
- 38s - loss: 0.0049 - accuracy: 0.9990 - val_loss: 0.0080 - val_accuracy: 0.9986
Epoch 9/20
- 37s - loss: 0.0041 - accuracy: 0.9991 - val_loss: 0.0076 - val_accuracy: 0.9988
Epoch 10/20
- 36s - loss: 0.0034 - accuracy: 0.9992 - val_loss: 0.0070 - val_accuracy: 0.9990
Epoch 11/20
- 36s - loss: 0.0030 - accuracy: 0.9991 - val_loss: 0.0063 - val_accuracy: 0.9991
Epoch 12/20
- 40s - loss: 0.0024 - accuracy: 0.9994 - val_loss: 0.0064 - val_accuracy: 0.9990
Epoch 13/20
- 44s - loss: 0.0020 - accuracy: 0.9995 - val_loss: 0.0063 - val_accuracy: 0.9991
Epoch 14/20
- 39s - loss: 0.0017 - accuracy: 0.9996 - val_loss: 0.0061 - val_accuracy: 0.9991
Epoch 15/20
- 38s - loss: 0.0015 - accuracy: 0.9996 - val_loss: 0.0064 - val_accuracy: 0.9990

```

*Рис. 33 Обучение сети 2 в течение 15 эпох*



*Рис. 34 Графики функции потерь и точности сети 2*

Здесь начало переобучения можно заметить уже после 2 эпохи. К этому моменту сеть добилась 99,78% точности. Обе сети дают одинаково хороший результат.

#### 4.2.3 C9\*15-MP2S2-C9\*15-MP2S2-FC150-FC28 (сеть 3)

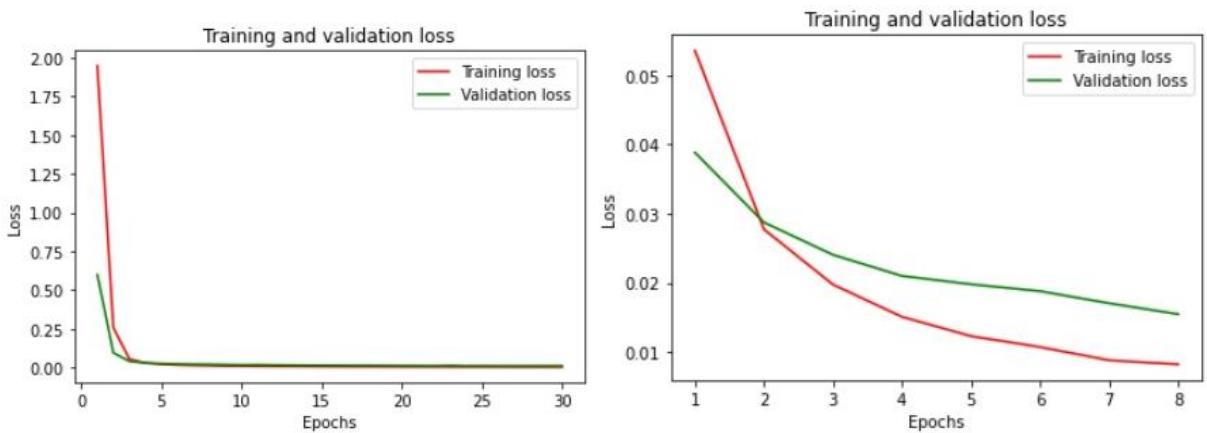
Третья нейронная сеть будет иметь намного меньше фильтров, но больший размер ядер. Сократим вдвое размер полно связного слоя. Конфигурация: C9\*15-MP2S2-C9\*15-MP2S2-FC150-FC28. Данная сеть имеет меньше всего параметров для обучения (28500 против 275 000 и 617 000 в предыдущих сетях).

```

Epoch 1/30
- 12s - loss: 1.9487 - accuracy: 0.5628 - val_loss: 0.5982 - val_accuracy: 0.9379
Epoch 2/30
- 13s - loss: 0.2554 - accuracy: 0.9693 - val_loss: 0.0934 - val_accuracy: 0.9924
Epoch 3/30
- 14s - loss: 0.0536 - accuracy: 0.9948 - val_loss: 0.0388 - val_accuracy: 0.9947
Epoch 4/30
- 12s - loss: 0.0277 - accuracy: 0.9961 - val_loss: 0.0287 - val_accuracy: 0.9954
Epoch 5/30
- 12s - loss: 0.0197 - accuracy: 0.9967 - val_loss: 0.0240 - val_accuracy: 0.9959
Epoch 6/30
- 12s - loss: 0.0151 - accuracy: 0.9975 - val_loss: 0.0210 - val_accuracy: 0.9962
Epoch 7/30
- 12s - loss: 0.0123 - accuracy: 0.9981 - val_loss: 0.0198 - val_accuracy: 0.9966
Epoch 8/30
- 12s - loss: 0.0107 - accuracy: 0.9982 - val_loss: 0.0188 - val_accuracy: 0.9965
Epoch 9/30
- 13s - loss: 0.0088 - accuracy: 0.9985 - val_loss: 0.0170 - val_accuracy: 0.9967
Epoch 10/30
- 13s - loss: 0.0082 - accuracy: 0.9987 - val_loss: 0.0155 - val_accuracy: 0.9969
Epoch 11/30
- 12s - loss: 0.0072 - accuracy: 0.9987 - val_loss: 0.0162 - val_accuracy: 0.9968
Epoch 12/30
- 13s - loss: 0.0066 - accuracy: 0.9989 - val_loss: 0.0145 - val_accuracy: 0.9969
Epoch 13/30
- 12s - loss: 0.0059 - accuracy: 0.9989 - val_loss: 0.0138 - val_accuracy: 0.9971
Epoch 14/30
- 12s - loss: 0.0055 - accuracy: 0.9990 - val_loss: 0.0126 - val_accuracy: 0.9975
Epoch 15/30
- 12s - loss: 0.0047 - accuracy: 0.9990 - val_loss: 0.0126 - val_accuracy: 0.9973

```

*Рис. 35 Обучение сети 3 в течение 15 эпох*



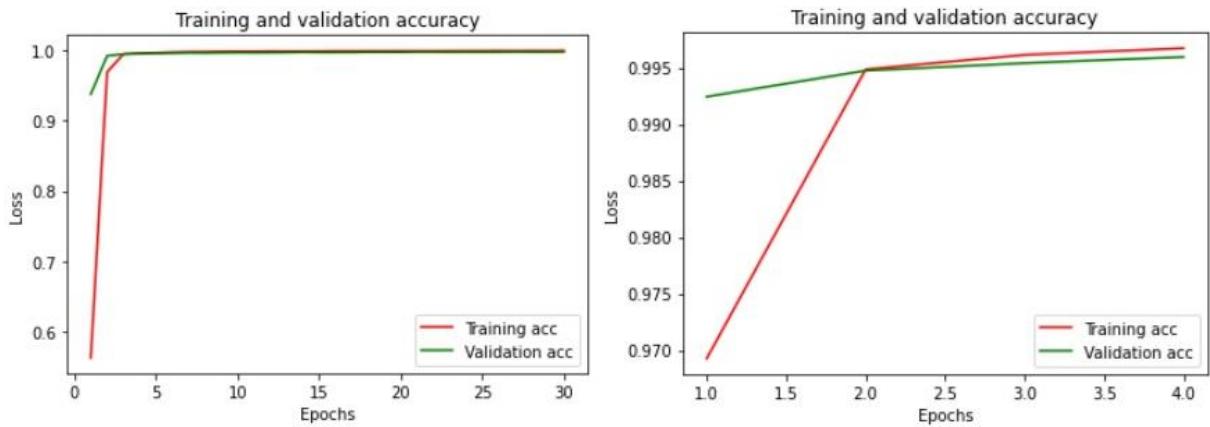


Рис. 36 Графики функции потерь и точности сети 3

Переобучение также начинается после 2 эпохи. Точность нейронной сети 99,24% на втором этапе. Заметим, что несмотря на переобучение, контрольная точность все равно продолжает увеличиваться на всем процессе обучения, а функция потерь продолжает падать.

#### 4.2.4 C9\*30-MP2S2-FC150-FC28 (сеть 4)

Посмотрим, каких показателей смогут добиться сверточные нейронные сети с другим количеством слоёв. Рассмотрим сеть с одним сверточным слоем. Возьмем последнюю нейронную сеть и вместо 2 сверточных слоёв по 15 фильтром сделаем 1 сверточный слой с 30 фильтрами. Конфигурация: C9\*30-MP2S2-FC150-FC28

```

Epoch 1/30
- 19s - loss: 0.2978 - accuracy: 0.9422 - val_loss: 0.0187 - val_accuracy: 0.9976
Epoch 2/30
- 22s - loss: 0.0134 - accuracy: 0.9978 - val_loss: 0.0114 - val_accuracy: 0.9982
Epoch 3/30
- 20s - loss: 0.0079 - accuracy: 0.9986 - val_loss: 0.0095 - val_accuracy: 0.9986
Epoch 4/30
- 20s - loss: 0.0056 - accuracy: 0.9990 - val_loss: 0.0091 - val_accuracy: 0.9986
Epoch 5/30
- 21s - loss: 0.0041 - accuracy: 0.9992 - val_loss: 0.0077 - val_accuracy: 0.9987
Epoch 6/30
- 19s - loss: 0.0034 - accuracy: 0.9992 - val_loss: 0.0072 - val_accuracy: 0.9988
Epoch 7/30
- 19s - loss: 0.0026 - accuracy: 0.9994 - val_loss: 0.0073 - val_accuracy: 0.9987
Epoch 8/30
- 19s - loss: 0.0021 - accuracy: 0.9996 - val_loss: 0.0067 - val_accuracy: 0.9990
Epoch 9/30
- 20s - loss: 0.0017 - accuracy: 0.9996 - val_loss: 0.0067 - val_accuracy: 0.9989
Epoch 10/30
- 20s - loss: 0.0014 - accuracy: 0.9997 - val_loss: 0.0066 - val_accuracy: 0.9986
Epoch 11/30
- 22s - loss: 0.0012 - accuracy: 0.9998 - val_loss: 0.0066 - val_accuracy: 0.9988
Epoch 12/30
- 25s - loss: 9.5442e-04 - accuracy: 0.9999 - val_loss: 0.0066 - val_accuracy: 0.9987
Epoch 13/30
- 21s - loss: 7.9692e-04 - accuracy: 0.9999 - val_loss: 0.0060 - val_accuracy: 0.9987
Epoch 14/30
- 22s - loss: 6.4418e-04 - accuracy: 1.0000 - val_loss: 0.0064 - val_accuracy: 0.9988
Epoch 15/30
- 19s - loss: 5.4034e-04 - accuracy: 1.0000 - val_loss: 0.0062 - val_accuracy: 0.9988

```

Рис. 37 Обучение сети 4 в течение первых 15 эпох

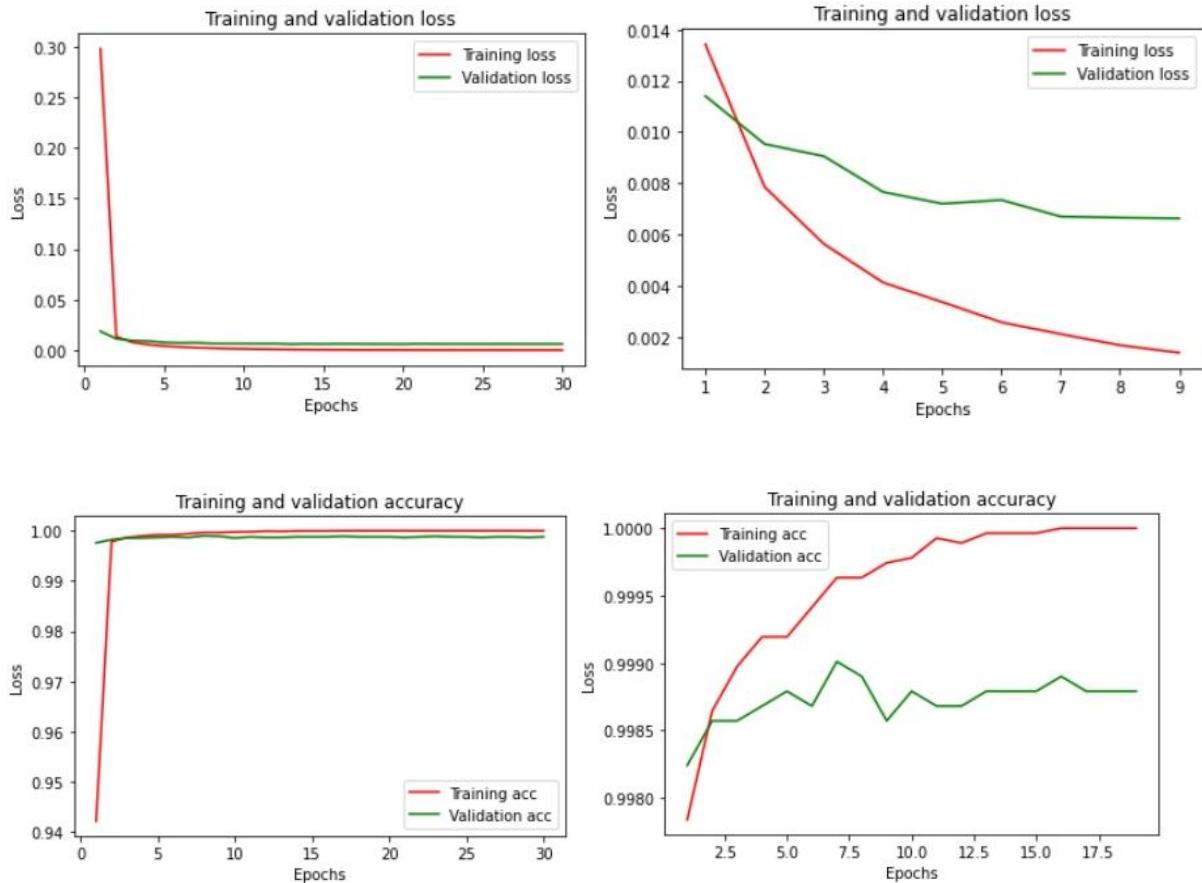


Рис. 38 Графики функции потерь и точности сети 4

После 6 эпохи точность перестает увеличиваться. Функция потерь перестает уменьшаться после 13 эпохи, сохраняя примерно одинаковое значение. Переобучение происходит также после 2 эпохи. Однако эта сеть даёт самую высокую точность в 99,82% после 2 эпохи. Самая высокая точность достигается на 3 эпохе и до самого конца обучения остается примерно на том же уровне.

#### 4.2.5 C3\*70-MP2S2-C3\*100-MP2S2-C3\*150-MP2S2-FC300-FC28 (сеть 5)

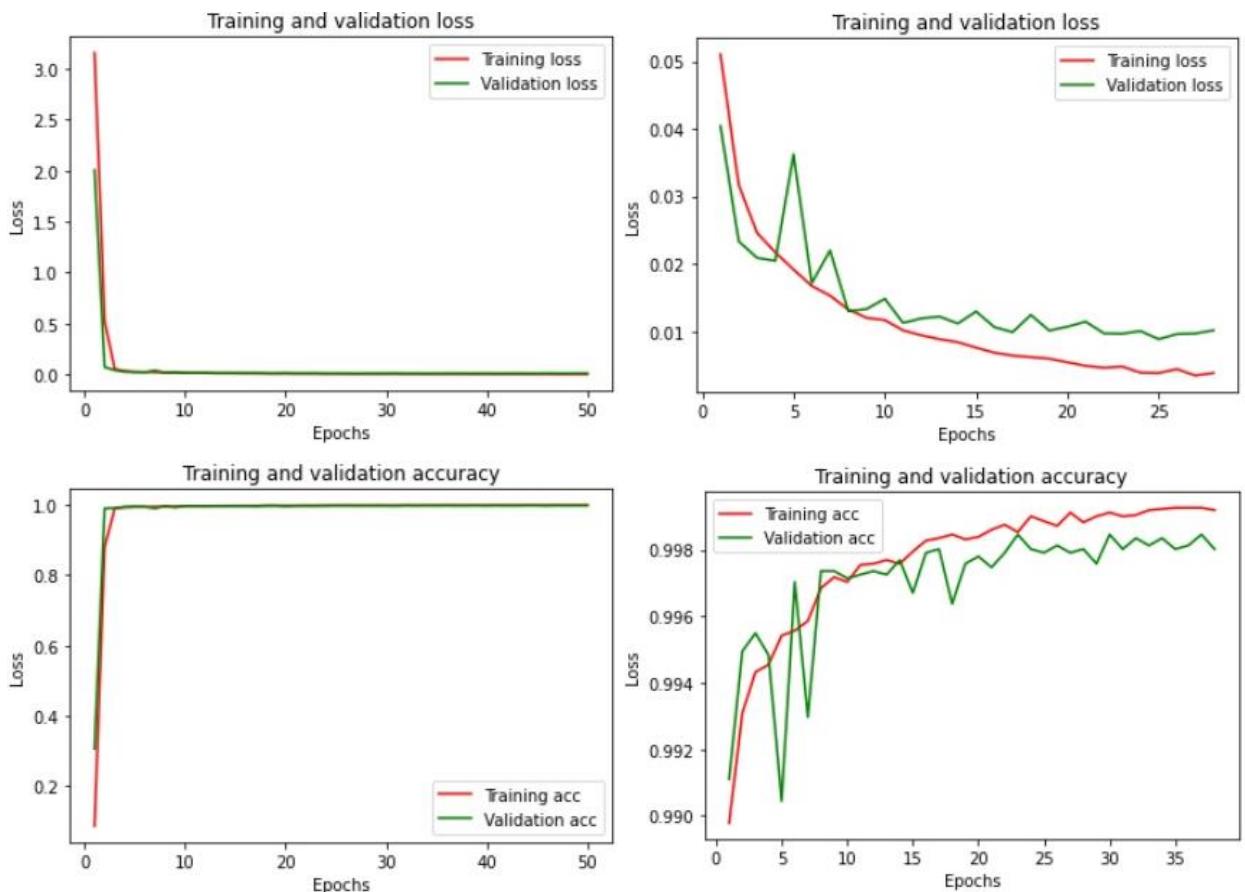
Последняя нейронная сеть будет состоять из 3 сверточных слоёв. Используем ядра небольшого размера (размер ядер уменьшен до 3), шаг оставим равным 1. Количество фильтров увеличено. Конфигурация сети: C3\*70-MP2S2-C3\*100-MP2S2-C3\*150-MP2S2-FC300-FC28

```

Epoch 1/50
- 38s - loss: 3.1573 - accuracy: 0.0860 - val_loss: 2.0061 - val_accuracy: 0.3059
Epoch 2/50
- 37s - loss: 0.5254 - accuracy: 0.8794 - val_loss: 0.0691 - val_accuracy: 0.9902
Epoch 3/50
- 34s - loss: 0.0510 - accuracy: 0.9898 - val_loss: 0.0404 - val_accuracy: 0.9911
Epoch 4/50
- 34s - loss: 0.0317 - accuracy: 0.9931 - val_loss: 0.0233 - val_accuracy: 0.9949
Epoch 5/50
- 39s - loss: 0.0246 - accuracy: 0.9943 - val_loss: 0.0209 - val_accuracy: 0.9955
Epoch 6/50
- 39s - loss: 0.0217 - accuracy: 0.9945 - val_loss: 0.0204 - val_accuracy: 0.9948
Epoch 7/50
- 36s - loss: 0.0192 - accuracy: 0.9954 - val_loss: 0.0362 - val_accuracy: 0.9904
Epoch 8/50
- 34s - loss: 0.0167 - accuracy: 0.9956 - val_loss: 0.0172 - val_accuracy: 0.9970
Epoch 9/50
- 35s - loss: 0.0153 - accuracy: 0.9959 - val_loss: 0.0220 - val_accuracy: 0.9930
Epoch 10/50
- 35s - loss: 0.0132 - accuracy: 0.9968 - val_loss: 0.0130 - val_accuracy: 0.9974
Epoch 11/50
- 36s - loss: 0.0120 - accuracy: 0.9972 - val_loss: 0.0133 - val_accuracy: 0.9974
Epoch 12/50
- 39s - loss: 0.0117 - accuracy: 0.9970 - val_loss: 0.0148 - val_accuracy: 0.9971
Epoch 13/50
- 37s - loss: 0.0101 - accuracy: 0.9975 - val_loss: 0.0112 - val_accuracy: 0.9973
Epoch 14/50
- 37s - loss: 0.0094 - accuracy: 0.9976 - val_loss: 0.0119 - val_accuracy: 0.9974
Epoch 15/50
- 37s - loss: 0.0088 - accuracy: 0.9977 - val_loss: 0.0122 - val_accuracy: 0.9973

```

*Рис. 39 Обучение сети 5 в течение первых 15 эпох*



*Рис. 40 Графики функции потерь и точности сети 5*

После 8 эпохи точность перестает увеличиваться. Функция потерь перестает уменьшаться после 10 эпохи. Переобучение стало появляться только после 12 эпохи. Можно

сделать вывод, что глубина сети увеличивает "вместимость" ("capacity") модели (увеличивает количество изучаемых параметров), а значит сеть дальше настраивает их и процесс переобучения наступает позже.

#### 4.2.6 C5\*50-MP2S2-C5\*100-MP2S2-FC300-D0.5-FC28 (сеть 6)

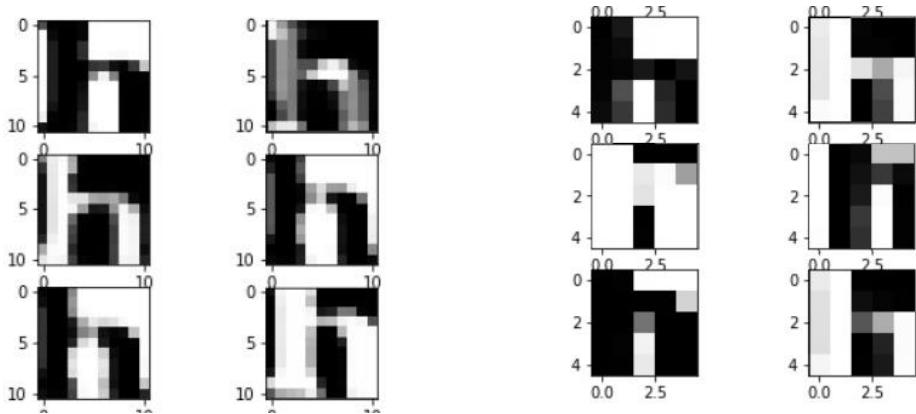
Для борьбы с переобучением на модели с конфигурацией C5\*50-MP2S2-C5\*100-MP2S2-FC300-FC28 был использован дополнительный слой Dropout layer с коэффициентом drop ratio = 0.5. Таким образом, сеть представляла собой: C5\*50-MP2S2-C5\*100-MP2S2-FC300-D0.5-FC28, однако это не дало заметных изменений, и сеть дала аналогичные результаты.

Все построенные свёрточные нейронные сети для капчи типа 1 дали очень высокий показатель точности, что может объясняться достаточно простой поставленной задачей, а также наличием малого разнообразия шрифтов в рассмотренной капче. Для последующего исследования выберем сеть, состоящую из 3 свёрточных слоёв из-за более позднего процесса насыщения данными (позднего начала переобучения).

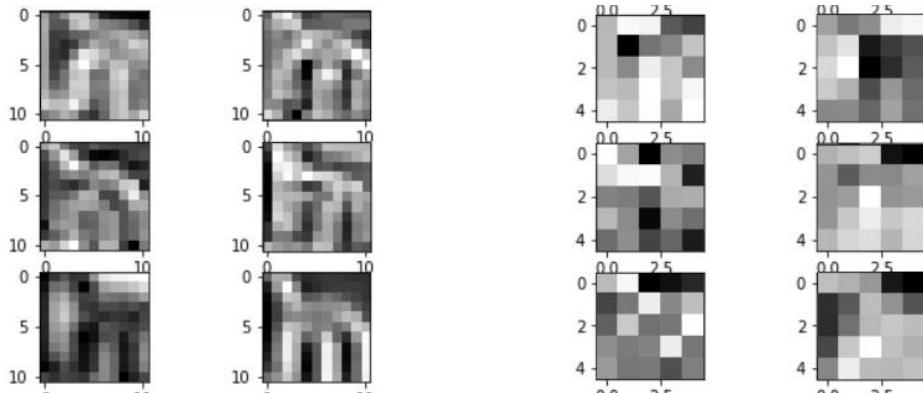
#### 4.2.7 C3\*50-MP2S2-C3\*70-MP2S2-FC300-FC28 (сеть 7)

Построение сети для распознавания второго вида капчи оказалось более интересной задачей. Сначала было принято решение построить сеть с аналогичной архитектурой (сеть, содержащую 3 свёрточных слоя). На удивление, такая сеть никак не обучалась. В то время, как предыдущие сети достигали точности в 99% уже после 2 эпохи, сеть 9 даже после 7 эпох обучения показывала точность чуть выше 3% и минимальное значение функции потерь loss = 3,33. С течением времени и точность, и функция потерь сохраняли свои значения.

Ошибка могла крыться в неправильном перемешивании выборки или меток, однако на самом деле проблема была в архитектуре сети. Особенность символов второго вида капчи заключается в том, что шрифт полый и представляет собой лишь тонкие линии, составляющие контур шрифта. Третий свёрточный слой просто не находил признаки, выделенные первыми двумя слоями. Слои субдискретизации сильно способствовали потере информации, размывая тонкие контуры символов. На рисунке ниже показана визуализация изображения символа после второго свёрточного слоя и второго субдискретизирующего слоя для капчи типа 1 и капчи типа 2. Можно видеть постепенную потерю информации для капчи типа 2.



*Рис. 41 Визуализация изображения после 2-го сверточного слоя и 2-го слоя субдискретизации сети 5 для капчи типа 1*



*Рис. 42 Визуализация изображения после 2-го сверточного слоя и 2-го слоя субдискретизации сети 5 для капчи типа 2*

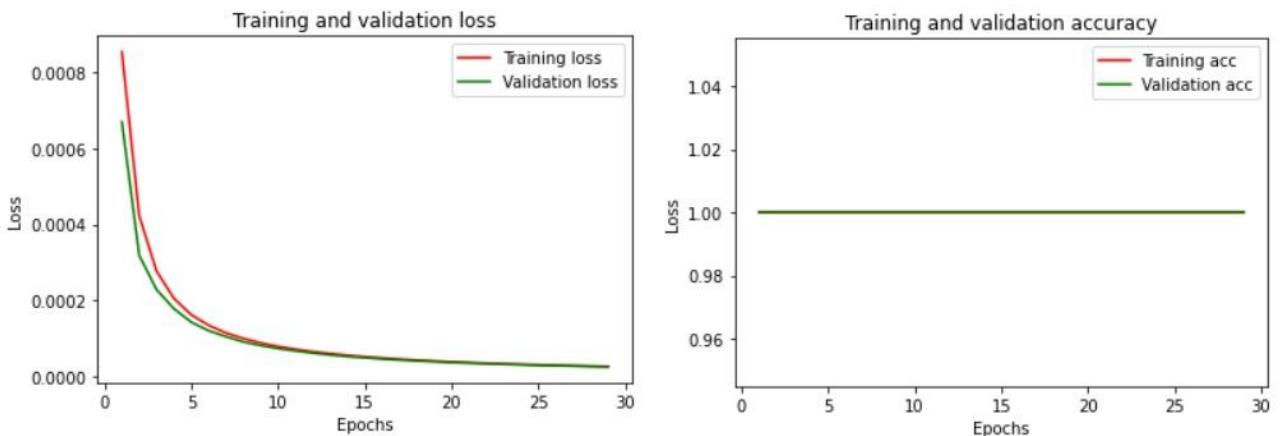
Сети с меньшим количеством слоёв дали лучшие результаты. Сеть с конфигурацией C3\*50-MP2S2-C3\*70-MP2S2-FC300-FC28 (сеть 7) дала значение точности на тестовых данных  $acc = 100\%$  уже после 1 эпохи обучения.

```

Epoch 1/30
- 83s - loss: 0.4676 - accuracy: 0.8794 - val_loss: 0.0014 - val_accuracy: 1.0000
Epoch 2/30
- 87s - loss: 8.5541e-04 - accuracy: 1.0000 - val_loss: 6.7072e-04 - val_accuracy: 1.0000
Epoch 3/30
- 85s - loss: 4.2267e-04 - accuracy: 1.0000 - val_loss: 3.1812e-04 - val_accuracy: 1.0000
Epoch 4/30
- 83s - loss: 2.7655e-04 - accuracy: 1.0000 - val_loss: 2.2786e-04 - val_accuracy: 1.0000
Epoch 5/30
- 88s - loss: 2.0468e-04 - accuracy: 1.0000 - val_loss: 1.7760e-04 - val_accuracy: 1.0000
Epoch 6/30
- 85s - loss: 1.6140e-04 - accuracy: 1.0000 - val_loss: 1.4207e-04 - val_accuracy: 1.0000
Epoch 7/30
- 81s - loss: 1.3389e-04 - accuracy: 1.0000 - val_loss: 1.1966e-04 - val_accuracy: 1.0000
Epoch 8/30
- 80s - loss: 1.1364e-04 - accuracy: 1.0000 - val_loss: 1.0410e-04 - val_accuracy: 1.0000
Epoch 9/30
- 80s - loss: 9.9117e-05 - accuracy: 1.0000 - val_loss: 9.0190e-05 - val_accuracy: 1.0000
Epoch 10/30
- 82s - loss: 8.7334e-05 - accuracy: 1.0000 - val_loss: 8.0542e-05 - val_accuracy: 1.0000
Epoch 11/30
- 81s - loss: 7.8101e-05 - accuracy: 1.0000 - val_loss: 7.2612e-05 - val_accuracy: 1.0000
Epoch 12/30
- 82s - loss: 7.0708e-05 - accuracy: 1.0000 - val_loss: 6.6285e-05 - val_accuracy: 1.0000
Epoch 13/30
- 81s - loss: 6.4553e-05 - accuracy: 1.0000 - val_loss: 6.0692e-05 - val_accuracy: 1.0000
Epoch 14/30
- 77s - loss: 5.9348e-05 - accuracy: 1.0000 - val_loss: 5.5658e-05 - val_accuracy: 1.0000
Epoch 15/30
- 75s - loss: 5.4869e-05 - accuracy: 1.0000 - val_loss: 5.1566e-05 - val_accuracy: 1.0000

```

*Рис. 43 Обучение сети 7 в течение первых 15 эпох*



*Рис. 44 Графики функции потерь и точности сети 7*

Нужно отметить, что максимальная точность была достигнута на первой эпохе, при этом функция ошибок продолжала уменьшаться в течение всего обучения. Даже после 30 эпох переобучения не происходило. На 30 эпохе достигнуто минимальное значение функции потерь  $val\_loss = 2.45e^{-5}$ .

### 4.3 Экспериментальное исследование влияния объёма обучающей выборки на точность распознавания

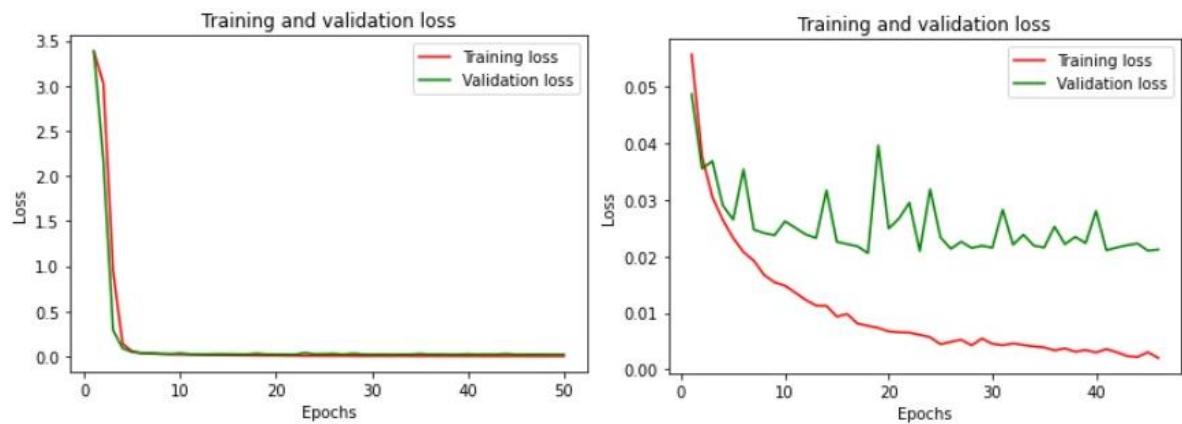
Рассмотрим влияние размера обучающей выборки на точность распознавания сети. Для исследования обучим сеть 5 для капчи типа 1 на выборках меньшего и большего объема.

#### 4.3.1 Меньшая выборка (сеть 8)

Выборка меньшего объема будет состоять из 18 480 символов (по 660 на класс). При этом, 13 860 (495) символов будут составлять обучающую выборку, а 4 620 (165) символов будут составлять тестовую выборку.

```
Epoch 1/50
- 18s - loss: 3.3749 - accuracy: 0.0371 - val_loss: 3.3850 - val_accuracy: 0.0335
Epoch 2/50
- 17s - loss: 3.0240 - accuracy: 0.1185 - val_loss: 2.1607 - val_accuracy: 0.2879
Epoch 3/50
- 17s - loss: 0.9499 - accuracy: 0.7693 - val_loss: 0.2924 - val_accuracy: 0.9558
Epoch 4/50
- 17s - loss: 0.1391 - accuracy: 0.9823 - val_loss: 0.0889 - val_accuracy: 0.9803
Epoch 5/50
- 17s - loss: 0.0558 - accuracy: 0.9891 - val_loss: 0.0487 - val_accuracy: 0.9905
Epoch 6/50
- 17s - loss: 0.0378 - accuracy: 0.9916 - val_loss: 0.0355 - val_accuracy: 0.9937
Epoch 7/50
- 17s - loss: 0.0305 - accuracy: 0.9929 - val_loss: 0.0369 - val_accuracy: 0.9922
Epoch 8/50
- 17s - loss: 0.0265 - accuracy: 0.9936 - val_loss: 0.0291 - val_accuracy: 0.9959
Epoch 9/50
- 17s - loss: 0.0233 - accuracy: 0.9935 - val_loss: 0.0265 - val_accuracy: 0.9963
Epoch 10/50
- 17s - loss: 0.0208 - accuracy: 0.9952 - val_loss: 0.0354 - val_accuracy: 0.9909
Epoch 11/50
- 17s - loss: 0.0193 - accuracy: 0.9947 - val_loss: 0.0247 - val_accuracy: 0.9959
Epoch 12/50
- 17s - loss: 0.0167 - accuracy: 0.9959 - val_loss: 0.0242 - val_accuracy: 0.9959
Epoch 13/50
- 17s - loss: 0.0154 - accuracy: 0.9962 - val_loss: 0.0238 - val_accuracy: 0.9961
Epoch 14/50
- 18s - loss: 0.0148 - accuracy: 0.9957 - val_loss: 0.0262 - val_accuracy: 0.9948
Epoch 15/50
- 17s - loss: 0.0136 - accuracy: 0.9965 - val_loss: 0.0251 - val_accuracy: 0.9957
```

Рис. 45 Обучение сети 8 в течение первых 15 эпох



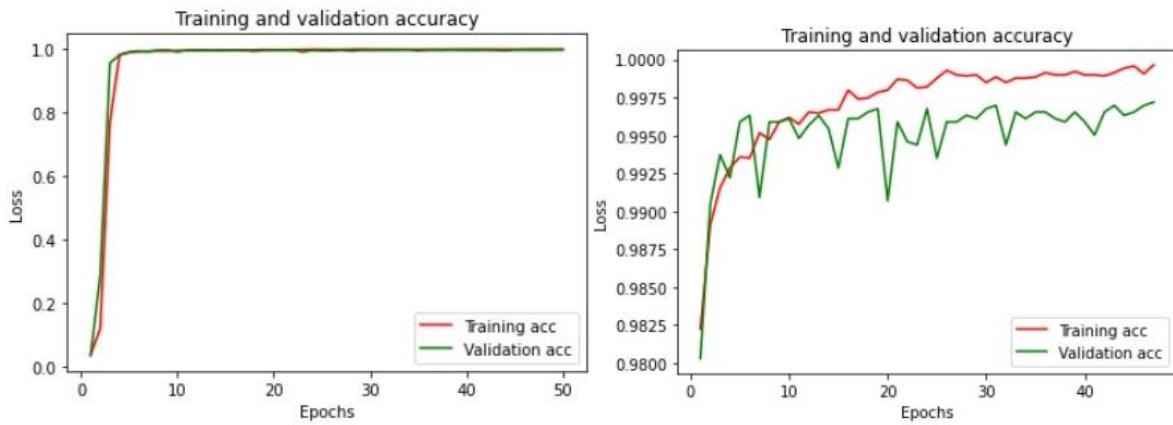


Рис. 46 Графики функции потерь и точности сети 8

После 9 эпохи точность перестает увеличиваться. Функция потерь перестает уменьшаться после 9 эпохи. Наибольшая достигнутая точность = 99,61%. Переобучение заметно после 9 эпохи.

#### 4.3.2 Большая выборка (сеть 9)

Выборка большего объема состоит из 3000 символов на каждый класс (всего  $3000 * 28 = 84000$  символов). Аналогично, три четверти её объема ( $63000 / 2250$ ) составляет обучающую выборку, четверть ( $21000 / 750$ ) – тестовую.

```

Epoch 1/25
- 76s - loss: 1.4505 - accuracy: 0.5967 - val_loss: 0.0534 - val_accuracy: 0.9901
Epoch 2/25
- 76s - loss: 0.0332 - accuracy: 0.9924 - val_loss: 0.0246 - val_accuracy: 0.9956
Epoch 3/25
- 78s - loss: 0.0204 - accuracy: 0.9952 - val_loss: 0.0200 - val_accuracy: 0.9963
Epoch 4/25
- 77s - loss: 0.0166 - accuracy: 0.9960 - val_loss: 0.0189 - val_accuracy: 0.9956
Epoch 5/25
- 75s - loss: 0.0140 - accuracy: 0.9966 - val_loss: 0.0179 - val_accuracy: 0.9967
Epoch 6/25
- 78s - loss: 0.0124 - accuracy: 0.9969 - val_loss: 0.0293 - val_accuracy: 0.9919
Epoch 7/25
- 76s - loss: 0.0105 - accuracy: 0.9976 - val_loss: 0.0184 - val_accuracy: 0.9963
Epoch 8/25
- 79s - loss: 0.0098 - accuracy: 0.9975 - val_loss: 0.0149 - val_accuracy: 0.9967
Epoch 9/25
- 75s - loss: 0.0085 - accuracy: 0.9980 - val_loss: 0.0193 - val_accuracy: 0.9955
Epoch 10/25
- 76s - loss: 0.0080 - accuracy: 0.9980 - val_loss: 0.0165 - val_accuracy: 0.9969
Epoch 11/25
- 77s - loss: 0.0074 - accuracy: 0.9981 - val_loss: 0.0132 - val_accuracy: 0.9973
Epoch 12/25
- 74s - loss: 0.0069 - accuracy: 0.9983 - val_loss: 0.0127 - val_accuracy: 0.9971
Epoch 13/25
- 77s - loss: 0.0062 - accuracy: 0.9983 - val_loss: 0.0132 - val_accuracy: 0.9972
Epoch 14/25
- 78s - loss: 0.0060 - accuracy: 0.9984 - val_loss: 0.0117 - val_accuracy: 0.9971
Epoch 15/25
- 77s - loss: 0.0054 - accuracy: 0.9986 - val_loss: 0.0137 - val_accuracy: 0.9970

```

Рис.47 Обучение сети 9 в течение первых 15 эпох

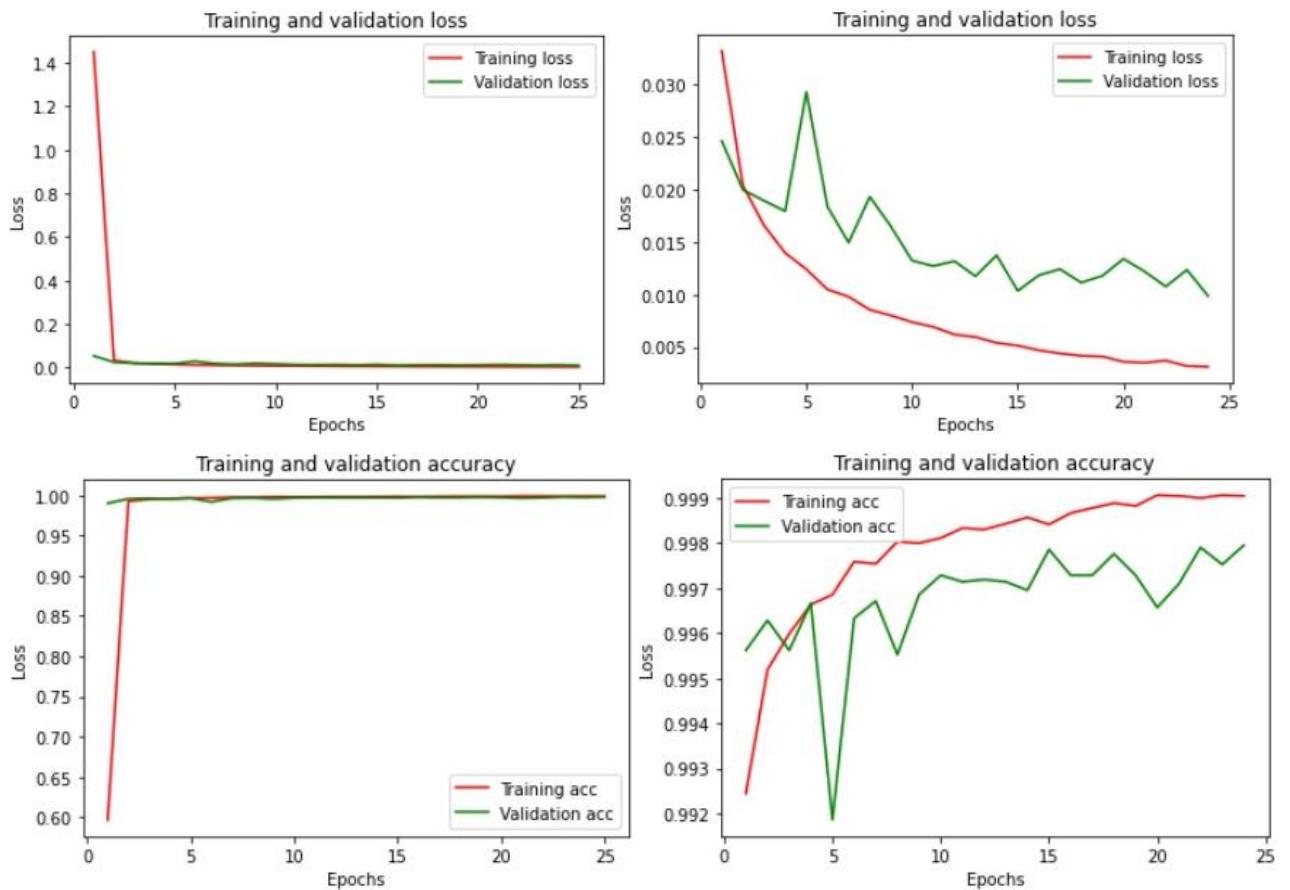


Рис. 48 Графики функции потерь и точности сети 9

После 11 эпохи точность перестает увеличиваться. Функция потерь перестает уменьшаться после 14 эпохи. Можно заметить некоторую закономерность между тремя построенными нейронными сетями, обученными на разном объеме данных.

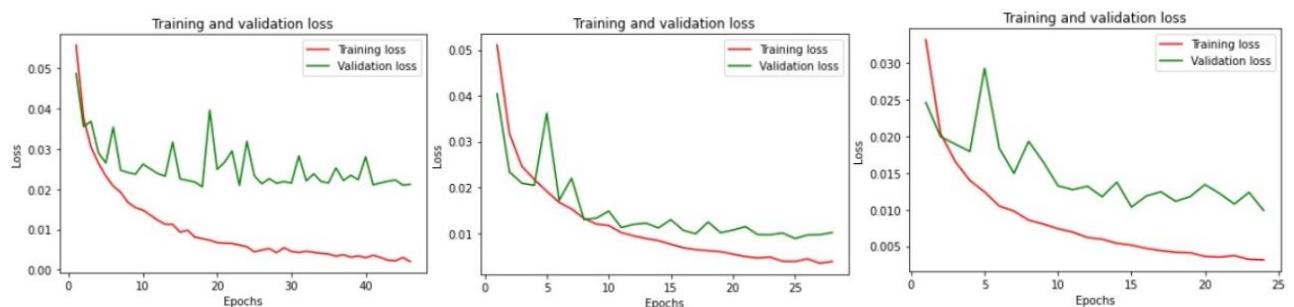
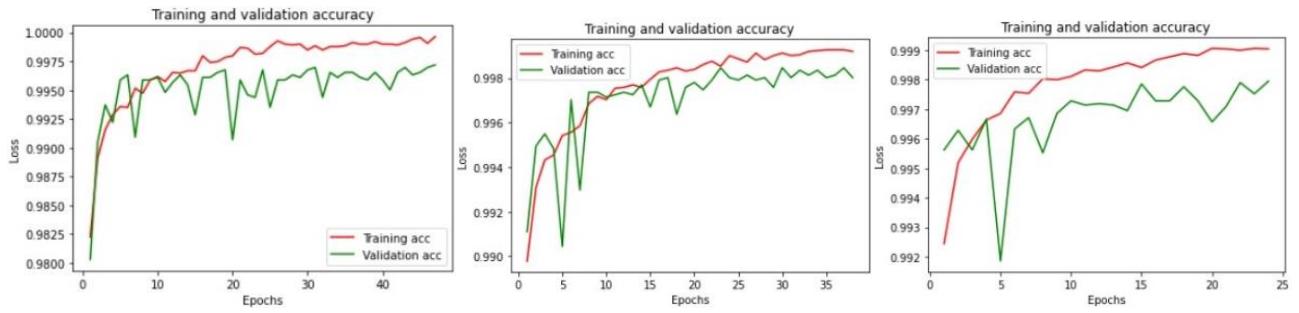


Рис.49 Сравнение функции потерь для выборок объема 500/1000/3000 (сети 8/6/9)

Чем больше обучающая выборка, тем меньше разрыв между тестовой потерей и тренировочной потерей. Для выборки 500 разрыв составляет примерно 0.016, для выборки 1000 - примерно 0.008, для выборки 3000 - примерно 0.006.



*Рис. 50 Сравнение точности для выборок объема 500/1000/3000 (сети 8/6/9)*

Аналогичная закономерность заметна для функции точности на тренировочных данных и на тестовых данных. Разрыв уменьшается, значит нейронная сеть лучше изучает закономерности, присущие символам.

#### 4.4 Выводы по результатам экспериментальных исследований

Задача распознавания 28-ми различных закрашенных символов, выбранных таким образом, чтобы они максимально отличались друг от друга (исключены похожие символы, например, буква о и символ нуля) является слишком простой для свёрточной нейронной сети, поэтому даже самые простые архитектуры, состоящие из всего 1 свёрточного слоя, дают очень хороший результат распознавания. Значительных зависимостей точности распознавания от параметров нейронной сети для капчи типа 1 обнаружено не было, все построенные сети имеют коэффициент распознавания больше 99%.

Для капчи типа 2 была обнаружена потеря информации после второго субдискретизирующего слоя. Причиной этого мог стать тонкий шрифт. Для распознавания выбрана сеть 7 с меньшим количеством слов, обученная на выборке объема 84000 (3000 на символ) изображения.

Найдена небольшая зависимость качества работы свёрточной нейронной сети от объема обучающей выборки. Обучаясь на большем объеме данных, нейронная сеть показывает лучшее качество распознавания тестовых данных.

## **Заключение**

В результате выполнения данной учебно-исследовательской работы будет достигнута главная цель - создан веб-сервис, выполняющий распознавание капчи на изображении. В ходе работы будут выполнены следующие задачи:

1. Изучены существующие подходы к распознаванию капчи
2. Изучены основные архитектуры нейронных сетей, используемые для распознавания капчи
3. Реализована архитектура нейронной сети
4. Проведены экспериментальные исследования влияния архитектур и обучающих выборок на точность распознавания.
5. Разработан веб-сервис.

Достигнутая степень успешного распознавания составляет 91,6 процента для капчи первого типа и 78,2 процента для капчи второго типа. Ошибки в ответах возникают преимущественно из-за ошибок при сегментации.

*Таблица 5. Показатели (вероятности) точности работы этапов распознавания капчи*

	Ошибка предобработки	Ошибка сегментации	Ошибка распознавания	Полное совпадение
Тип 1	0.0	0.0840	0.0	0.9160
Тип 2	0.0	0.1875	0.0374	0.7821

Исходный код проекта доступен в открытом репозитории Github [22]. В дальнейшем для улучшения работы и расширения области применения необходимо улучшать метод сегментации, скорость работы функций предобработки, увеличивать типы поддерживаемых капч. Областью применения может служить взлом существующих капч на реальных сайтах при разработке расширения браузера.

## Литература

1. Понкин Д. В поисках идеальной CAPTCHA URL:<https://habr.com/ru/post/120851/> (дата обращения: 17.02.2020).
2. Луис Педро Коэльо, Вилли Ричард Построение систем машинного обучения на языке Python – ДМК Пресс, 2016
3. S. Ben Driss, M. Soua, R. Kachouri, M. Akil A comparison study between MLP and Convolutional Neural Network models for character recognition, 2017. URL: [https://www.researchgate.net/publication/316613582\\_A\\_comparison\\_study\\_between\\_MLP\\_and\\_Convolutional\\_Neural\\_Network\\_models\\_for\\_character\\_recognition](https://www.researchgate.net/publication/316613582_A_comparison_study_between_MLP_and_Convolutional_Neural_Network_models_for_character_recognition)
4. Narges Roshanbin, James Miller A Survey and Analysis of Current CAPTCHA Approaches. 2012. URL:[https://www.riverpublishers.com/journal/journal\\_articles/RP\\_Journal\\_1540-9589\\_1211.pdf](https://www.riverpublishers.com/journal/journal_articles/RP_Journal_1540-9589_1211.pdf) (дата обращения: 07.02.2020).
5. Norhidayu binti Abdul Hamid, Nilam Nur Binti Amir Sjarif Handwritten Recognition Using SVM, KNN and Neural Network. 2017. URL: <https://arxiv.org/ftp/arxiv/papers/1702/1702.00723.pdf> (дата обращения: 01.04.2020).
6. Haichang Gao, Jeff Yan, Fang Cao, Zhengya Zhang, Lei Lei, Mengyun Tang, Ping Zhang, Xin Zhou, Xuqin Wang and Jiawei Li A Simple Generic Attack on Text Captchas. 2016. C. 9. URL: <https://www.ndss-symposium.org/wp-content/uploads/2017/09/simple-generic-attack-text-captchas.pdf> (дата обращения: 18.03.2020).
7. H Gao, W Wang, J Qi, X Wang, X Liu, and J Yan The robustness of hollow captchas. 2013. C. 1075 – 1086. URL: <https://passwordresearch.com/papers/paper759.html>
8. K. Chellapilla, P. Y. Simard Using Machine Learning to Break Visual Human Interaction Proofs (HIPs). 2004. C. 265–272. URL: <http://papers.nips.cc/paper/2571-using-machine-learning-to-break-visual-human-interaction-proofs-hips.pdf> (дата обращения: 21.03.2020).
9. J. Yan, A. S. E. Ahmad A low-cost attack on a microsoft CAPTCHA. 2008. C. 543–554. URL: <https://dl.acm.org/doi/10.1145/1455770.1455839> (дата обращения: 25.03.2020).
10. Xiyang Liu, Yang Zhang, Jing Hu, Mengyun Tang, Haichang Gao Breaking Text-based CAPTCHAs using Average Vertical Partition
11. Распознавание рукописных символов с использованием Python и scikit. 2013. URL:<https://habr.com/ru/post/171723/> (дата обращения: 10.03.2020).
12. Martin Kopp, Matej Nikl, Martin Holena Breaking CAPTCHAs with Convolutional Neural Networks. URL: <http://ceur-ws.org/Vol-1885/93.pdf>

13. Min Wang,Tianhui Zhang,Hao Song,Wenrong Jiang The Recognition of CAPTCHA, 2013.URL:[http://www.engii.org/workshop2014/januarysubmission/files/upload/soloplay\\_2013100420131892910.pdf](http://www.engii.org/workshop2014/januarysubmission/files/upload/soloplay_2013100420131892910.pdf)
14. D. Subramanian A Simple Introduction to K-Nearest Neighbors Algorithm URL: <https://towardsdatascience.com/a-simple-introduction-to-k-nearest-neighbors-algorithm-b3519ed98e> (дата обращения: 05.04.2020).
15. Ricky Han SimGAN-Captcha lib. URL: <https://github.com/0b01/SimGAN-Captcha> (дата обращения: 15.04.2020).
16. Kalen Liu Captcha-break lib. URL: <https://github.com/nladuo/captcha-break> (дата обращения: 15.04.2020).
17. SixQuant Captcha lib. URL: <https://github.com/SixQuant/captcha> (дата обращения: 15.04.2020).
18. Harmo Zhengwh Captcha-svm lib. URL: <https://github.com/zhengwh/captcha-svm> (дата обращения: 15.04.2020).
19. beyondZB SVM-Captcha lib. URL: <https://github.com/beyondZB/SVM-Captcha> (дата обращения: 15.04.2020).
20. Дмитрий Дементий Почему Django — лучший фреймворк для разработки сайтов. 2019. URL:<https://ru-hexlet-io.turbopages.org/s/ru.hexlet.io/blog/posts/pochemu-django-luchshiy-freymvork-dlya-razrabortki-saytov> (дата обращения 10.05.2020)
21. Gareth Dwyer Flask vs. Django: Why Flask Might Be Better. 2017. URL: <https://www.codementor.io/@garethdwyer/flask-vs-django-why-flask-might-be-better-4xs7mdf8v> (дата обращения 10.05.2020)
22. Репозиторий проекта URL: [https://github.com/Dora9000/captcha\\_recognition](https://github.com/Dora9000/captcha_recognition) (дата последнего обновления 02.06.2020)

## Приложения

### Приложение 1. Код скрипта классификации

```
1.  from PIL import Image, ImageDraw
2.
3.  def Classifier(pixel_map):
4.
5.      def Grey_scale(pixel):
6.          border = 25
7.          d1 = abs(pixel[0]-pixel[1])**2
8.          d2 = abs(pixel[0]-pixel[2])**2
9.          d3 = abs(pixel[1]-pixel[2])**2
10.         if d1 <= border and d2 <= border and d3 <= border:
11.             return True
12.         return False
13.
14.     w, h = len(pixel_map), len(pixel_map[0])
15.
16.     grey_colors = 0
17.     for i in range(w):
18.         for j in range(h):
19.             if Grey_scale(pixel_map[i][j]):
20.                 grey_colors += 1
21.
22.     if grey_colors > 5000:
23.         return 1
24.     else:
25.         return 2
```

## Приложение 2. Код скрипта предобработки и сегментации первого типа капчи

```
1. import numpy as np
2. from math import fabs
3. from matplotlib import pyplot as plt
4. import copy
5. import cv2 ##download opencv-python
6. import os
7. from PIL import Image, ImageFilter, ImageDraw
8. from collections import deque
9. from time import time
10. from random import randrange as rnd
11.
12.
13. def get_path_to_save(PATH):
14.     PATH = PATH[:-8]
15.     return PATH
16.
17.
18. def open_color(path):
19.     #img = cv2.imread(GetPathToImage(number, "jpeg"))
20.     img = cv2.imread(path)
21.     #plt.imshow(img);
22.     return img
23.
24. def image_show(img, nrows=1, ncols=1, cmap='gray'):
25.     fig, ax = plt.subplots(nrows=nrows, ncols=ncols, figsize=(10, 10))
26.     ax.imshow(img, cmap='gray')
27.     ax.axis('off')
28.     return fig, ax
29.
30. def noize_line(img):
31.     th = cv2.adaptiveThreshold(img, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
32. cv2.THRESH_BINARY, 17, 2)
33.     kernel = np.ones((2,2), np.uint8)
34.     dilation = cv2.dilate(th, kernel, iterations=1)
35.     erosion = cv2.erode(dilation, kernel, iterations=1)
36.     return erosion
37.
38. def cut_luz(img):
39.     img[(img > 215) & (img < 255)] = 255
40.     return img
41.
42. def no_background(img):
43.     for i in range(img.shape[0]-1,-1,-1):
44.         for j in range(img.shape[1]-1,-1,-1):
45.             img.itemset((i,j,0),max(0, min(254,-img.item(i,j,0) +
46. img.item(0,0,0))))
47.             img.itemset((i,j,1),max(0, min(254,-img.item(i,j,1) +
48. img.item(0,0,1))))
49.             img.itemset((i,j,2),max(0, min(254,-img.item(i,j,2) +
50. img.item(0,0,2))))
```

```

51.     average = img.mean(axis=0).mean(axis=0)
52.     pixels = np.float32(img.reshape(-1, 3))
53.     n_colors = 7
54.     criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 200,
55.     .1)
56.     flags = cv2.KMEANS_RANDOM_CENTERS
57.     _, labels, palette = cv2.kmeans(pixels, n_colors, None, criteria, 10,
58.     flags)
59.     _, counts = np.unique(labels, return_counts=True)
60.     dominant = palette[np.argmax(counts)]
61.     return dominant
62.
63.
64. def preproseed(img):
65.     img2 = copy.copy(img)
66.     average = dominant_color(img2)
67.     for i in range(img.shape[0]-1,-1,-1):
68.         for j in range(img.shape[1]-1,-1,-1):
69.             img.itemset((i,j,0),max(0, min(254,-img.item(i,j,0) +
70.             average[0])))
71.             img.itemset((i,j,1),max(0, min(254,-img.item(i,j,1) +
72.             average[1])))
73.             img.itemset((i,j,2),max(0, min(254,-img.item(i,j,2) +
74.             average[2])))
75.     img = cv2.bitwise_not(img)
76.     img2 = no_background(img2)
77.     average = dominant_color(img2)
78.
79.     for i in range(img.shape[0]):
80.         for j in range(img.shape[1]):
81.             if (img2.item(i,j,0) + img2.item(i,j,1) + img2.item(i,j,2) <
82.             average[0] + average[1] + average[2] - 150):
83.                 img2.itemset((i,j,0),0)
84.                 img2.itemset((i,j,1),0)
85.                 img2.itemset((i,j,2),0)
86.             else:
87.                 if (img2.item(i,j,0) + img2.item(i,j,1) + img2.item(i,j,2) <
88.                 average[0] + average[1] + average[2] - 50):
89.                     img2.itemset((i,j,0),170)
90.                     img2.itemset((i,j,1),170)
91.                     img2.itemset((i,j,2),170)
92.
93.     img = cv2.addWeighted(img,0.4,img2,0.6,0)
94.     img = cut_luz(img)
95.     img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
96.     img = noize_line(img)
97.     return img
98.
99.
100. #DSU part
101.
102. class DSU_2D:
103.     d = []
104.     p = []
105.     def __init__(self, n, m):

```

```

99.         self.d = [[1 for _ in range(m)] for __ in range(n)]
100.        self.p = [[(i, j) for j in range(m)] for i in range(n)]
101.
102.    def get(self, x, y):
103.        if self.p[x][y] == (x, y):
104.            return (x, y)
105.        self.p[x][y] = self.get(self.p[x][y][0], self.p[x][y][1])
106.        return self.p[x][y]
107.
108.    def uni(self, a, b):
109.        a = self.get(a[0], a[1])
110.        b = self.get(b[0], b[1])
111.        if a == b:
112.            return False
113.        if self.d[a[0]][a[1]] < self.d[b[0]][b[1]]:
114.            a, b = b, a
115.        self.p[b[0]][b[1]] = a
116.        self.d[a[0]][a[1]] += self.d[b[0]][b[1]]
117.        return True
118.
119.
120.
121.    def generate_steps(dlt, flag):
122.        if flag:
123.            return [(i, j) for i in range(-dlt, dlt + 1) for j in range(-dlt, dlt + 1)]
124.        return [(i, j) for i in range(-dlt, dlt + 1) for j in range(-dlt, dlt + 1) if not (i == 0 and j == 0)]
125.
126.    def distance(dim, a, b, coefs):
127.        ans = 0
128.        if dim == 1:
129.            return abs(a - b) * coefs;
130.        for i in range(dim):
131.            ans += abs(a[i] - b[i]) * coefs[i];
132.        return ans
133.
134.    def connect(pixel_map, steps, border, coefs):
135.        w = len(pixel_map)
136.        h = len(pixel_map[0])
137.        dsu = DSU_2D(w, h)
138.        for x in range(w):
139.            for y in range(h):
140.                for (dx, dy) in steps:
141.                    nx = x + dx
142.                    ny = y + dy
143.                    if nx < 0 or ny < 0 or nx >= w or ny >= h:
144.                        continue
145.                    if distance(3, pixel_map[x][y], pixel_map[nx][ny], coefs) <= border:
146.                        dsu.uni((x, y), (nx, ny))
147.        return dsu
148.
149.
150.
```

```

151. def MainFilter_captchal(path):
152.     img = open_color(path)
153.     image2 = copy.copy(img)
154.     image2 = preproseed(image2)
155.     white = 255
156.     black = 0
157.     h,w = image2.shape[0], image2.shape[1]
158.     pixel_map = [[(image2.item(y,x),image2.item(y,x),image2.item(y,x)) for
y in range(h)] for x in range(w)]
159.
160.
161.     steps = generate_steps(1, True)
162.     dsu = connect(pixel_map, steps, 0, [1, 1, 1])
163.     my_mp = {}
164.     for i in range(w):
165.         for j in range(h):
166.             if dsu.get(i,j) not in my_mp:
167.                 my_mp[dsu.get(i,j)] = []
168.                 my_mp[dsu.get(i,j)].append((i,j))
169.     for key in my_mp:
170.         if len(my_mp[key]) <= 15:
171.             for val in my_mp[key]:
172.                 pixel_map[val[0]][val[1]] = (255,255,255)
173.
174.     steps = generate_steps(1, True)
175.     dsu = connect(pixel_map, steps, 0, [1, 1, 1])
176.     my_mp = {}
177.     for i in range (w):
178.         for j in range(h):
179.             if dsu.get(i,j) not in my_mp:
180.                 my_mp[dsu.get(i,j)] = []
181.                 my_mp[dsu.get(i,j)].append((i,j))
182.     for key in my_mp:
183.         if len(my_mp[key]) <= 5:
184.             for val in my_mp[key]:
185.                 pixel_map[val[0]][val[1]] = (0,0,0)
186.
187.
188.     oy = []
189.     for x in range(w):
190.         stolb = 0
191.         for y in range(h):
192.             if pixel_map[x][y] == (0,0,0):
193.                 stolb += 1
194.             if stolb <= 0 or x < 2 or x > w - 5:
195.                 stolb = 0
196.             oy.append(stolb)
197.
198.
199.     #fig, ax = plt.subplots()
200.     ox = [i for i in range(w)]
201.     #ax.plot(ox, oy)
202.
203.     cuts = []
204.     i = 0

```

```

205.     while i < w:
206.         check = False
207.         while i < w and oy[i] <= 0:
208.             i += 1
209.         if i < w:
210.             check = True
211.             start = i
212.         while i < w and oy[i] > 0:
213.             i += 1
214.         if check:
215.             cuts.append((start, i))
216.             start = i
217.
218.         if len(cuts) < 3 or len(cuts) > 4:
219.             cuts1 = []
220.
221.         for q in range(len(cuts)):
222.             i,j = cuts[q]
223.             if i == 0 and j == 0:
224.                 continue
225.             if j - i < 15:
226.                 if q == 0:#to right
227.                     a,b = cuts[q+1]
228.                     cuts1.append((i, b))
229.                     cuts[q+1] = (0,0)
230.
231.             elif q == len(cuts) - 1:#to left
232.                 a,b = cuts1[len(cuts1) - 1]
233.                 b = j
234.                 cuts1[len(cuts1) - 1] = (a,b)
235.
236.             else:
237.                 il, jl = cuts[q-1]
238.                 ir, jr = cuts[q+1]
239.                 left = jl - il
240.                 right = jr - ir
241.                 if left <= right:#to left
242.                     a,b = cuts1[len(cuts1) - 1]
243.                     b = j
244.                     cuts1[len(cuts1) - 1] = (a,b)
245.                 else:#to right
246.                     a,b = cuts[q+1]
247.                     cuts1.append((i, b))
248.                     cuts[q+1] = (0,0)
249.             elif j - i > 35:
250.                 border = 220
251.                 d = []
252.                 for x in range(j - i):
253.                     delta = 0
254.                     cnt = 0
255.                     for y in range(h):
256.                         if pixel_map[x+i,y] == (255,255,255):
257.                             continue
258.                         if pixel_map[x+i+1,y] == (255,255,255):
259.                             continue

```

```

260.                     cnt += 1
261.                     sm = 0
262.                     for k in range(3):
263.                         sm += (pixel_map[x+i][y][k] -
pixel_map[x+i+1][y][k]) **2
264.                         if sm >= border:
265.                             delta += 1
266.                             if x > 15 and j - i - x > 15:
267.                                 d.append((delta/max(cnt,1), x))
268.                         d.sort()
269.                         if len(d) == 0:
270.                             x = (j - i) // 2
271.                         else:
272.                             delta, x = d[len(d) - 1]
273.                             cuts1.append((i, i + x))
274.                             cuts1.append((i + x, j))
275.                         else:
276.                             cuts1.append((i, j))
277.                         cuts = cuts1
278.
279.                         cuts.sort()
280.
281.                         for x in range(w):
282.                             for y in range(h):
283.                                 if pixel_map[x][y] == (255,255,255):
284.                                     continue
285.                                 else:
286.                                     pixel_map[x][y] = (0,0,0)
287.
288.
289.                         cuts2 = []
290.                         for (i,j) in cuts:
291.                             mn = h
292.                             mx = 0
293.                             for u in range(j - i):
294.                                 for k in range(h):
295.                                     if pixel_map[u+i][k] == (255,255,255):
296.                                         continue
297.                                     else:
298.                                         mn = min(mn, k)
299.                                         mx = max(mx, k)
300.                         cuts2.append((mn,mx))
301.
302.                         #draw cuts _ vertical
303.                         def draw_cut_vert():
304.                             set_cuts = set()
305.                             for i in range(len(cuts)):
306.                                 set_cuts.add(cuts[i][0])
307.                                 set_cuts.add(cuts[i][1])
308.                             for i in range(w):
309.                                 if i in set_cuts:
310.                                     for j in range(h):
311.                                         pixel_map[i][j] = (180,180,180)
312.
313.                         #draw cuts _ horizontal

```

```

314.     def draw_cut_hor():
315.         itt = 0
316.         for (i,j) in cuts:
317.             (h1,h2) = cuts2[itt]
318.             for a in range(j - i):
319.                 pixel_map[a+i][h1] = (180,180,180)
320.                 pixel_map[a+i][h2] = (180,180,180)
321.             itt += 1
322.
323.         for x in range(w):
324.             for y in range(h):
325.                 image2.itemset((y,x),pixel_map[x][y][0])
326.
327. #image_show(image2)
328. itt = 0
329. for (i,j) in cuts:
330.     (h1,h2) = cuts2[itt]
331.     crop_img = image2[h1:h2, i:j]
332.     size = (28, 28)
333.     crop_img = cv2.resize(crop_img, size)
334. #image_show(crop_img)
335. cv2.imwrite(get_path_to_save(path) + str(itt) + ".jpeg", crop_img)
336. itt+=1

```

### Приложение 3. Код скрипта предобработки и сегментации второго типа капчи

```
1.  from PIL import Image
2.  from collections import deque
3.  from random import randrange as rnd
4.  import os
5.
6.  white = (255, 255, 255)
7.  black = (0, 0, 0)
8.
9.
10. def get_path_to_save(PATH):
11.     PATH = PATH[:-7]
12.     return PATH
13.
14. #def get_path_to_image(index, s):
15. #    return 'C:\\\\Users\\\\User\\\\Desktop\\\\no_names\\\\ (%d).' % index + s
16.
17. def generate_steps(dlt, flag):
18.     if flag:
19.         return [(i, j) for i in range(-dlt, dlt + 1) for j in range(-dlt, dlt + 1)]
20.     return [(i, j) for i in range(-dlt, dlt + 1) for j in range(-dlt, dlt + 1) if not (i == 0 and j == 0)]
21.
22.
23. def connect(mp, steps):
24.     w = len(mp)
25.     h = len(mp[0])
26.     colors = [[-1 for _ in range(h)] for _ in range(w)]
27.
28.     def connect_bfs(start_x, start_y, color):
29.         q = deque()
30.         q.append((start_x, start_y))
31.         colors[start_x][start_y] = color
32.         while len(q):
33.             cx, cy = q.popleft()
34.             for dx, dy in steps:
35.                 nx = cx + dx
36.                 ny = cy + dy
37.                 if min(nx, ny) < 0 or nx == w or ny == h:
38.                     continue
39.                 if colors[nx][ny] != -1 or mp[nx][ny] != mp[cx][cy]:
40.                     continue
41.                 colors[nx][ny] = color
42.                 q.append((nx, ny))
43.
44.     answer_size = 0
45.     for x in range(w):
46.         for y in range(h):
47.             if colors[x][y] == -1:
48.                 connect_bfs(x, y, answer_size)
49.                 answer_size += 1
50.
51.     ans = [[] for _ in range(answer_size)]
52.
```

```

53.     for x in range(w):
54.         for y in range(h):
55.             ans[colors[x][y]].append((x, y))
56.
57.     return ans
58.
59. def clear_corners(mp):
60.     w = len(mp)
61.     h = len(mp[0])
62.     colors = [[-1 for _ in range(h)] for _ in range(w)]
63.     steps = generate_steps(1, False)
64.     delta = 3
65.
66.     def clear_bfs(start_x, start_y, color):
67.         q = deque()
68.         q.append((start_x, start_y))
69.         colors[start_x][start_y] = color
70.         while len(q):
71.             cx, cy = q.popleft()
72.             for dx, dy in steps:
73.                 nx = cx + dx
74.                 ny = cy + dy
75.                 if min(nx, ny) < 0 or nx == w or ny == h:
76.                     continue
77.                 if colors[nx][ny] != -1 or mp[nx][ny] != mp[cx][cy]:
78.                     continue
79.                 colors[nx][ny] = color
80.                 q.append((nx, ny))
81.
82.         for x in range(w):
83.             for y in range(h):
84.                 if min(x, w - 1 - x, y, h - 1 - y) >= delta:
85.                     continue
86.                 if mp[x][y] != white:
87.                     clear_bfs(x, y, 0)
88.
89.         for x in range(w):
90.             for y in range(h):
91.                 if colors[x][y] != -1:
92.                     mp[x][y] = white
93.
94.     return mp
95.
96. def k_means(list_of_pixels, k):
97.     arr = [[rnd(256), rnd(256), rnd(256)] for _ in range(k)]
98.     n = len(list_of_pixels)
99.
100.    def get_class(color):
101.        dist = 10 ** 10
102.        current_class = -1
103.        for current_index in range(k):
104.            cur_dist = (color[0] - arr[current_index][0]) ** 2 \
105.                      + (color[1] - arr[current_index][1]) ** 2 \
106.                      + (color[2] - arr[current_index][2]) ** 2
107.            if cur_dist < dist:

```

```

108.             dist = cur_dist
109.             current_class = current_index
110.         return current_class
111.
112.     classes = [[] for _ in range(k)]
113.
114.     for i in range(n):
115.         classes[get_class(list_of_pixels[i][0])].append(i)
116.
117.     error = 1
118.     while error:
119.         error = 0
120.         dr = [0 for _ in range(k)]
121.         dg = [0 for _ in range(k)]
122.         db = [0 for _ in range(k)]
123.         count = [1 for _ in range(k)]
124.         for i in range(k):
125.             class_r, class_g, class_b = arr[i]
126.             for val in classes[i]:
127.                 count[i] += list_of_pixels[val][1]
128.                 r, g, b = list_of_pixels[val][0]
129.                 dr[i] += (r - class_r) * list_of_pixels[val][1]
130.                 dg[i] += (g - class_g) * list_of_pixels[val][1]
131.                 db[i] += (b - class_b) * list_of_pixels[val][1]
132.
133.             classes[i].clear()
134.             for i in range(k):
135.                 error += abs(dr[i]) // count[i] + abs(dg[i]) // count[i] +
136.                 abs(db[i]) // count[i]
137.                 arr[i][0] += dr[i] // count[i]
138.                 arr[i][1] += dg[i] // count[i]
139.                 arr[i][2] += db[i] // count[i]
140.
141.             for i in range(n):
142.                 classes[get_class(list_of_pixels[i][0])].append(i)
143.
144.             answer = dict()
145.             for j in range(k):
146.                 for index in classes[j]:
147.                     answer[list_of_pixels[index][0]] = (arr[j][0], arr[j][1],
148.                         arr[j][2])
149.
150.     def filter_image(path):
151.         #name = get_path_to_image(number_of_image, "png")
152.
153.         try:
154.             image = Image.open(path).convert('RGB')
155.         except FileNotFoundError:
156.             assert False
157.
158.         w, h = image.size
159.         image = image.load()
160.
```

```

161.     pixels = [[image[x, y] for y in range(h)] for x in range(w)]
162.
163.     mp = {}
164.     for i in range(w):
165.         for j in range(h):
166.             cur = pixels[i][j]
167.             if cur not in mp:
168.                 mp[cur] = 0
169.             mp[cur] += 1
170.
171.     k = 10 # 5
172.     color_mp = k_means(list(mp.items()), k)
173.     for i in range(w):
174.         for j in range(h):
175.             pixels[i][j] = color_mp[pixels[i][j]]
176.
177.     mp = {}
178.     for i in range(w):
179.         for j in range(h):
180.             cur = pixels[i][j]
181.             if cur not in mp:
182.                 mp[cur] = 0
183.             mp[cur] += 1
184.
185.     presents = 15 # если больше presents процентов на картинке данного
цвета, то будем считать, что это фон
186.     total_size = w * h
187.     background_colors = set()
188.     background_colors.add(white) # чтобы сразу убрать белые точки
189.
190.     for color in mp:
191.         if mp[color] / total_size * 100 >= presents:
192.             background_colors.add(color)
193.
194.     for x in range(w):
195.         for y in range(h):
196.             if pixels[x][y] in background_colors:
197.                 pixels[x][y] = white
198.
199.     return pixels
200.
201.
202. def main_filter_captcha_2(path): #number_of_image
203.     while True:
204.         #pixels = clear_corners(filter_image(number_of_image))
205.         pixels = clear_corners(filter_image(path))
206.         w = len(pixels)
207.         h = len(pixels[0])
208.         black_count = 0
209.         for x in range(w):
210.             for y in range(h):
211.                 if pixels[x][y] != white:
212.                     pixels[x][y] = black
213.                     black_count += 1
214.

```

```

215.         if 0.05 * w * h < black_count < 0.95 * w * h:
216.             break
217.
218.     steps = generate_steps(1, True)
219.     classes = connect(pixels, steps)
220.     low_border = 10 # размер связкой области
221.
222.     for current_class in classes:
223.         if len(current_class) <= low_border:
224.             for point in current_class:
225.                 cx, cy = point
226.                 pixels[cx][cy] = white
227.
228.     # cuts
229.     oy = []
230.     for x in range(w):
231.         column = 0
232.         for y in range(h):
233.             if pixels[x][y] == black:
234.                 column += 1
235.             oy.append(column)
236.
237.     cuts = []
238.     low_border = 1
239.     i = 0
240.     start = 0
241.     while i < w:
242.         check = False
243.         while i < w and oy[i] < low_border:
244.             i += 1
245.         if i < w:
246.             check = True
247.             start = i
248.         while i < w and oy[i] >= low_border:
249.             i += 1
250.         if check:
251.             cuts.append((start, i))
252.             start = i
253.
254.     if len(cuts) < 5:
255.         cuts1 = []
256.         difference = 5 - len(cuts)
257.         if difference == 5:
258.             i = 0
259.             j = w - 1
260.             for k in range(6):
261.                 cuts1.append((i + (j - i) * k // 6, i + (j - i) * (k + 1) // 6))
262.
263.         if difference == 4:
264.             i = 0
265.             j = w - 1
266.             for k in range(5):
267.                 cuts1.append((i + (j - i) * k // 5, i + (j - i) * (k + 1) // 5))

```

```

268.         if difference < 4:
269.             c = []
270.             for i, j in cuts:
271.                 if j - i <= 20:
272.                     cuts1.append((i, j))
273.                     continue
274.                 else:
275.                     c.append((i, j))
276.             for i, j in c:
277.                 if difference == 0:
278.                     cuts1.append((i, j))
279.                     continue
280.                 if j - i > 60:
281.                     coefficient = min(difference + 1, 4)
282.                 elif j - i > 40:
283.                     coefficient = min(difference + 1, 3)
284.                 else:
285.                     coefficient = min(difference + 1, 2)
286.                 for k in range(coefficient):
287.                     cuts1.append((int(i + (j - i) * k / coefficient),
288.                                   int(i + (j - i) * (k + 1) / coefficient)))
289.                     difference = max(0, difference - coefficient)
290.
291.
292.             cuts.sort()
293.             cuts2 = []
294.             for (i, j) in cuts:
295.                 mn = h
296.                 mx = 0
297.                 for u in range(j - i):
298.                     for k in range(h):
299.                         if pixels[u + i][k] == white:
300.                             continue
301.                         else:
302.                             mn = min(mn, k)
303.                             mx = max(mx, k)
304.             cuts2.append((mn, mx))
305.
306.             #print(cuts2)
307.
308.             img2 = Image.new('RGB', (w, h))
309.             mp = img2.load()
310.             for x in range(w):
311.                 for y in range(h):
312.                     mp[x, y] = pixels[x][y]
313.             # img2.show()
314.             itt = 0
315.             for (mnx, mxx) in cuts:
316.                 (mnh, mxh) = cuts2[itt]
317.                 img3 = img2.crop((mnx, mn, mxx, mxh))
318.                 size = (28, 28)
319.                 img3 = img3.resize(size)
320.                 # img3.show()
321.                 itt += 1

```

```
322.         img3.save(get_path_to_save(path) + str(itt) + ".jpeg", "JPEG")
323.
324.     return pixels
```

#### Приложение 4. Код скрипта распознавания

```
1. import keras
2. import pickle
3. import os
4. import cv2
5.
6. def recognition_1():
7.     CUR_PATH = os.getcwd()
8.     CUR_PATH = CUR_PATH[:-4]
9.     PATH = CUR_PATH + "/pic/"
10.    CUR_PATH += "/cgi-bin/my_libs/"
11.    model = keras.models.load_model(CUR_PATH + '/network/model_25.h5')
12.    lb = pickle.loads(open(CUR_PATH + "/network/bin_class.txt",
13.                           "rb").read()) #загружаем сохраненный бинаризатор меток
14.    os.chdir(PATH)
15.    listdir = os.listdir()
16.    answer = ""
17.    for letter in listdir:
18.        if 'img' in letter:
19.            continue
20.        if 'jpeg' not in letter and 'png' not in letter :
21.            continue
22.        if '.' in letter:
23.            #print(PATH + letter)
24.            test_image = cv2.imread(PATH + letter)
25.            test_image = cv2.resize(test_image,(28,28))
26.            test_image = test_image.astype("float") / 255.0
27.            test_image = test_image.reshape((1,test_image.shape[0],
28.                                             test_image.shape[1],test_image.shape[2]))
29.            preds = model.predict(test_image)
30.            i = preds.argmax(axis=1)[0]
31.            test_label = lb.classes_[i]
32.            #print("Результат распознавания: " + test_label)
33.            answer += test_label
34.    return answer
35.
36. def recognition_2():
37.     CUR_PATH = os.getcwd()
38.     CUR_PATH = CUR_PATH[:-4]
39.     PATH = CUR_PATH + "/pic/"
40.     CUR_PATH += "/cgi-bin/my_libs/"
41.     model = keras.models.load_model(CUR_PATH + '/network/2/model_2_26.h5')
42.     lb = pickle.loads(open(CUR_PATH + "/network/2/bin_class.txt",
43.                           "rb").read()) #загружаем сохраненный бинаризатор меток
44.     os.chdir(PATH)
45.     listdir = os.listdir()
46.     answer = ""
47.     for letter in listdir:
48.         if 'img' in letter:
49.             continue
50.         if 'jpeg' not in letter and 'png' not in letter :
51.             continue
52.         if '.' in letter:
```

```
52.         #print(PATH + letter)
53.         test_image = cv2.imread(PATH + letter)
54.         test_image = cv2.resize(test_image,(28,28))
55.         test_image = test_image.astype("float") / 255.0
56.         test_image = test_image.reshape((1,test_image.shape[0],
57.                                         test_image.shape[1],test_image.shape[2]))
58.         preds = model.predict(test_image)
59.         i = preds.argmax(axis=1)[0]
60.         test_label = lb.classes_[i]
61.         #print("Результат распознавания: " + test_label)
62.         answer += test_label
63.     return answer
```