

Министерство науки и высшего образования Российской Федерации

ФГБОУ ВО АЛТАЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Институт цифровых технологий, электроники и физики

Кафедра вычислительной техники и электроники (ВТиЭ)

УДК 004.94

Работа защищена

«\_\_» \_\_\_\_\_ 2025 г.

Оценка \_\_\_\_\_

Председатель ГЭК, д.т.н., проф.

\_\_\_\_\_ С. П. Пронин

Допустить к защите

«\_\_» \_\_\_\_\_ 2025 г.

Заведующий кафедрой ВТиЭ,

к.ф.-м.н., доцент

\_\_\_\_\_ В. В. Пашнев

СЕРВИС АВТОМАТИЗИРОВАННОГО РЕШЕНИЯ САРТСНА

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К ВЫПУСКНОЙ

КВАЛИФИКАЦИОННОЙ РАБОТЕ

БР 09.03.01.5.306М.1337 ПЗ

Студент группы \_\_\_\_\_ 5.306М \_\_\_\_\_ А. В. Лаптев

Руководитель работы \_\_\_\_\_ к.ф.-м.н., доцент \_\_\_\_\_ В. В. Электроник

Консультанты:

Нормоконтролер \_\_\_\_\_ к.ф.-м.н., доцент \_\_\_\_\_ А. В. Калачёв

Барнаул 2025

## РЕФЕРАТ

Объем работы листов	22
Количество рисунков	6
Количество используемых источников	5
Количество таблиц	6

КОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ, СИСТЕМА УПРАВЛЕНИЯ  
ВЕРСИЯМИ.

Объём текста не менее 1000 символов! Пока счётчики выставляются в  
ручную, при необходимости правьте cls-файл.

## ABSTRACT

The amount of work sheets	22
Number of images	6
Number of sources used	5
Number of tables	6

COMPUTER SIMULATION, DISTRIBUTED VERSION CONTROL.

Большой текст на английском!

## СОДЕРЖАНИЕ

1. Обзор технологии САРТСНА . . . . .	4
1.1. Краткая история и назначение САРТСНА . . . . .	4
1.2. Классификация САРТСНА по формату взаимодействия с пользо- вателем . . . . .	5
1.3. Критерии надежности и уязвимости различных систем САРТСНА	8
2. Методология решения задач САРТСНА . . . . .	11
2.1. Общие подходы к автоматизированному решению САРТСНА . .	11
2.2. Архитектуры нейронных сетей для различных форматов САРТСНА	16
2.3. Подготовка и аннотация датасетов . . . . .	28
3. Экспериментальная часть и исследовательский вклад . . . . .	37
3.1. Сравнительный анализ моделей нейронных сетей для автомати- зированного решения САРТСНА в текстовом формате . . . . .	37
3.2. Эксперименты по аудио- и графическим САРТСНА . . . . .	40
3.3. Исследовательский вклад . . . . .	48
4. Обсуждение, выводы и перспективы развития . . . . .	56
4.1. Анализ надежности и уязвимостей решений . . . . .	56
4.2. Ограничения проведенных экспериментов . . . . .	56
4.3. Перспективные направления . . . . .	56
СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ . . . . .	57
ПРИЛОЖЕНИЕ . . . . .	57

## 1. ОБЗОР ТЕХНОЛОГИИ CAPTCHA

### 1.1. Краткая история и назначение CAPTCHA

Проверочный код CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) – это метод защиты, основанный на принципе аутентификации «вызов-ответ». Он предназначен для предотвращения автоматических действий, таких как спам или попытки взлома учетных записей, путем выполнения пользователем простого теста, подтверждающего, что он человек, а не программа [1]. Термин был придуман в 2003 году ([https://link.springer.com/chapter/10.1007/3-540-39200-9\\_18](https://link.springer.com/chapter/10.1007/3-540-39200-9_18)).

Исторически распространенный тип CAPTCHA был впервые изобретен в 1997 году двумя группами, работающими параллельно. Эта форма CAPTCHA требует ввода последовательности букв или цифр из искаженного изображения. Поскольку тест проводится компьютером, в отличие от стандартного теста Тьюринга, который проводится человеком, CAPTCHA иногда описываются как обратные тесты Тьюринга (<https://isyou.info/jowua/papers/jowua-v4n3-3.pdf>).

Набравшая популярность технология reCAPTCHA, была приобретена Google в 2009 году (<https://googleblog.blogspot.com/2009/09/teaching-computers-to-read-google.html>). В дополнение к предотвращению мошенничества с ботами для пользователей, Google использовал технологию reCAPTCHA для оцифровки архивов The New York Times и книг из Google Books в 2011 году (<https://www.nytimes.com/2011/03/29/science/29recaptcha.html>).

На сегодняшний день CAPTCHA является важной мерой безопасности, так как предотвращает автоматические атаки, например, массовую регистрацию ботов, и защищает данные пользователя. Современные системы CAPTCHA используют не только текст, но и изображения, аудио, поведенческие анализы и другие инновационные подходы, чтобы сделать тесты удобными для людей, но сложными для программ. Среднестатистическому человеку требуется около 10 секунд, чтобы решить типичный CAPTCHA.

## 1.2. Классификация CAPTCHA по формату взаимодействия с пользователем

На сегодняшний день наиболее распространенные виды CAPTCHA включают:

1. reCAPTCHA – разработанная Google система, которая предлагает тесты на основе распознавания объектов, анализа поведения или текстовых символов.
2. hCAPTCHA – альтернатива reCAPTCHA, фокусирующаяся на защите конфиденциальности пользователей.
3. Capu – система CAPTCHA, предлагающая пользователю головоломки, например, сборку изображения или взаимодействие с элементами интерфейса [2].

reCAPTCHA – система защиты от автоматизированных действий, разработанная Google, которая помогает различать человека и бота. Она объединяет несколько подходов, делая проверку удобной для пользователей, но сложной для автоматических систем [3].

reCAPTCHA включает в себя следующие версии (<https://en.wikipedia.org/wiki/>)

1. reCAPTCHA v1 (устарела в 2018 году):
  - 1.1. пользователи вводили текст, состоящий из искаженных слов, отображаемых на изображении;
  - 1.2. использовала слова из книг и документов, которые не могли быть распознаны OCR.
2. reCAPTCHA v2:
  - 2.1. клик по флажку: пользователи подтверждают, что они не роботы, нажимая на флажок «Я не робот»;
  - 2.2. выбор объектов на изображениях: пользователи идентифицируют заданные объекты на сетке из картинок;
  - 2.3. аудио CAPTCHA: для пользователей с ограничениями зрения, предлагается прослушать запись и ввести услышанные символы.

### 3. reCAPTCHA v3:

- 3.1. полностью работает в фоновом режиме, анализируя поведение пользователя на странице;
- 3.2. не требует явных действий, если пользователь считается низко-рискованным [4].

hCAPTCHA – это альтернативная система CAPTCHA, разработанная для защиты сайтов от ботов и спама, при этом уделяющая особое внимание конфиденциальности пользователей. Она стала популярной благодаря своей гибкости и ориентации на защиту данных [5].

#### Основные особенности hCAPTCHA:

1. конфиденциальность:
  - 1.1. в отличие от reCAPTCHA, hCAPTCHA не собирает данные о пользователях для рекламных целей, что делает ее привлекательной с точки зрения соблюдения конфиденциальности.
2. простота интеграции:
  - 2.1. легко интегрируется с web-сайтами через API;
  - 2.2. совместима с большинством популярных платформ, таких как WordPress, и может быть настроена для разных типов взаимодействия.
3. модели монетизации:
  - 3.1. владельцы сайтов могут зарабатывать, разрешая hCAPTCHA использовать проверочные задачи, связанные с машинным обучением, например, разметку данных.

#### Виды взаимодействия с пользователями:

1. графическая CAPTCHA: выбор изображений, соответствующих запросу;
2. текстовая CAPTCHA: ввод символов (редко используется);
3. аудио CAPTCHA: для пользователей с ограниченными возможностями, предлагается прослушать и ввести услышанные символы;

4. клик CAPTCHA: нажатие на флажок «Я не робот» (для низкорискованных пользователей).

Сару CAPTCHA – это инновационная система CAPTCHA, разработанная с акцентом на удобство для пользователей и адаптацию к современным web-средам. Она предлагает интерактивные методы проверки, направленные на минимизацию раздражения пользователей при сохранении высокого уровня защиты от ботов [6].

Основные особенности Сару CAPTCHA:

1. интерактивность:
  - 1.1. Сару использует методы проверки, которые требуют не просто ввода текста или выбора картинок, а выполнения задач, таких как перемещение объектов;
  - 1.2. простые задачи делают процесс проверки менее раздражающим и более интуитивным;
2. гибкость настройки:
  - 2.1. система может быть адаптирована под конкретные нужды сайта, включая выбор сложности задач и дизайн интерфейса.
3. доступность:
  - 3.1. подходит для пользователей с различными потребностями, включая мобильные устройства.

Виды взаимодействия с пользователями:

1. головоломки (Puzzle CAPTCHA): сборка пазла с перемещением недостающих элементов в нужное место;
2. тесты на логику и распознавание: выбор нужного объекта или логического варианта из предложенных;
3. текстовая CAPTCHA (редко используется).

Сару CAPTCHA используется на сайтах, где важны как защита от ботов, так и положительный пользовательский опыт. Особенно популярна в проектах с высоким акцентом на дизайн и пользовательское взаимодействие.

### 1.3. Критерии надежности и уязвимости различных систем CAPTCHA

Эффективность CAPTCHA-систем определяется совокупностью признаков. К основным критериям надежности относятся ([https://en.wikipedia.org/wiki/C](https://en.wikipedia.org/wiki/CAPTCHA)

1. устойчивость к машинному распознаванию, в том числе с использованием современных алгоритмов искусственного интеллекта;
2. наличие разнообразных и уникальных тестов, исключающих возможность формирования обучающих или атакующих датасетов;
3. доступность и понятность графического пользовательского интерфейса для широкой аудитории.

Несмотря на свою популярность, CAPTCHA-системы обладают рядом уязвимостей, снижающих их надежность и ухудшающих пользовательский опыт (<https://eitca.org/cybersecurity/eitc-is-wasf-web-applications-security-fundamentals/authentication-eitc-is-wasf-web-applications-security-fundamentals/webauthn/examination-review-webauthn/what-are-the-main-vulnerabilities-and-limitations-associated-with-traditional-text-based-captchas/>, <https://www.imperva.com/learn/application-security/what-is-captcha/#what-is-captcha>):

1. высокая когнитивная нагрузка, связанная со сложностью задач для человека;
2. недоступность или трудности прохождения для отдельных групп пользователей, включая людей с нарушениями зрения или слуха;
3. низкая эффективность против целевых атак или сервисов, управляемых человеком;
4. несовместимость с некоторыми web-браузерами и мобильными устройствами;
5. ограниченная поддержка вспомогательных технологий, используемых людьми с ограниченными возможностями.

Для CAPTCHA в текстовом формате выделяют следующие критерии надежности:



1. преднамеренное искажение символов (геометрическая деформация, перекрытие, наклоны);
2. использование нестандартных шрифтов, графических помех и шумов;
3. отсутствие четкой сегментации между символами, затрудняющей их раздельное распознавание;
4. рандомизация длины строк и набора используемых символов.

Несмотря на совокупность методов для усложнения автоматизированного распознавания, текстовые CAPTCHA подвержены следующим уязвимостям:

1. современные OCR-системы и seq2seq-модели, в том числе архитектуры на основе CNN и RNN, успешно справляются с распознаванием даже при наличии искажений;
2. упрощенные CAPTCHA без шумов и дополнительных помех могут быть распознаны с высокой точностью даже базовыми алгоритмами;
3. использование ограниченного и фиксированного алфавита позволяет обучать модели, показывающие высокую точность при распознавании.

Для CAPTCHA в аудиоформате основными характеристиками надежности являются:

1. введение фонового шума и аудиоискажений, затрудняющих автоматическую обработку;
2. использование слов, сходных по звучанию, нестандартных акцентов и синтезированной речи;
3. наложение голосов, изменение темпа и интонации произношения;
4. высокая вариативность аудиофайлов.

В то же время современным реализациям аудио CAPTCHA присущи следующие уязвимости:

1. преобразование аудио в спектрограммы с последующим анализом с помощью CNN и методов СТС позволяет достигать высокой точности распознавания;
2. современные модели автоматического распознавания речи успешно решают даже зашумленные аудиозадания;
3. применение генеративных моделей и других методов предварительной обработки аудио позволяет эффективно устранять шумы, повышая точность распознавания.

К ключевым характеристикам надежности САРТСНА с изображениями относятся:

1. использование изображений из реального мира с вариативными фонами и сценами;
2. намеренное смещение объектов по положению, углу поворота и масштабу;
3. включение визуально схожих ложных объектов, усложняющих выбор правильных;
4. разнообразие типов изображений.

Среди основных уязвимостей графических САРТСНА можно выделить:

1. высокая эффективность современных моделей детектирования объектов (например, YOLOv8, Faster R-CNN) при наличии специализированного обучающего датасета;
2. возможность автоматизации взаимодействия с САРТСНА (например, выбор изображений) с использованием скриптов и эмуляторов браузеров;
3. ограниченность числа классов, используемых в задаче, позволяет быстро обучить модель для решения конкретной САРТСНА.

## 2. МЕТОДОЛОГИЯ РЕШЕНИЯ ЗАДАЧ САРТСНА

### 2.1. Общие подходы к автоматизированному решению САРТСНА

Для автоматизации решения САРТСНА могут применяться различные методы и подходы, которые зависят от конкретной реализации САРТСНА. В рамках данной работы можно выделить следующие методы:

1. реализация САРТСНА в аудиоформате:
  - 1.1. шумоподавление и фильтрация;
  - 1.2. распознавание речи;
  - 1.3. преобразование речи в текст.
2. реализация САРТСНА в текстовом формате:
  - 2.1. бинаризация изображения и фильтрация фона;
  - 2.2. сегментация символов;
  - 2.3. классификация символов;
  - 2.4. распознавание последовательности символов;
  - 2.5. аугментация данных;
  - 2.6. генерация синтетических датасетов.
3. реализация САРТСНА в графическом формате:
  - 3.1. нормализация и ручная разметка изображений;
  - 3.2. детекция объектов;
  - 3.3. сегментация объектов;
  - 3.4. классификация фрагментов изображения;
  - 3.5. поиск и сопоставление с шаблоном.
4. общие подходы для большинства реализаций:
  - 4.1. анализ структуры HTML-документа на стороне клиента;
  - 4.2. автоматизация кликов и действий на web-странице.

Для каждого формата одним из основополагающих этапов является этап предобработки исходного файла. Для аудиозаписей, которые используются в Audio САРТСНА требуется применить шумоподавление и фильтрацию.

Шумоподавление (шумопонижение) – процесс устранения или уменьшения нежелательных шумов из аудиосигнала с целью повышения его качества или уменьшения уровня ошибок при передаче и хранении данных. Методы шумоподавления могут быть реализованы как аппаратно, так и программно, и направлены на увеличение отношения сигнал/шум. (<https://fastercapital.com/ru/content/>)

Фильтрация – процесс удаления или ослабления определенных частотных составляющих аудиосигнала при сохранении других. Фильтрацию можно использовать для удаления шума, частотный спектр которого отличается от желаемого сигнала.

Фильтрация играет важную роль в улучшении качества звучания аудиофайлов, эффективном подавлении шумов и помех, а также в создании специальных звуковых эффектов.

Для САРТСНА в текстовом формате в качестве предобработки используется бинаризация и фильтрация фоновых шумов для упрощения следующих этапов обработки.

Бинаризация – процесс преобразования цветного или полутонного изображения в двухцветное (черно-белое), где каждый пиксель принимает одно из двух возможных значений: 0 (черный) или 1 (белый). Основным параметром бинаризации является пороговое значение, с которым сравнивается яркость каждого пикселя. Существуют различные методы бинаризации, включая глобальные и локальные подходы. (<https://cyberleninka.ru/article/n/issledovanie-metodov-binarizatsii-izobrazheniy/viewer>)

Бинаризация широко используется в задачах распознавания текста, где упрощение изображения до двух цветов облегчает выделение символов и их последующее распознавание.

Фильтрация фонового шума – процесс удаления или уменьшения нежелательных шумов, которые могут мешать анализу или распознаванию содержимого изображения. Шумы могут возникать из-за различных факторов,

таких как условия съемки, качество оборудования или передача данных. В частности, для САРТСНА фоновые шумы генерируются намеренно.

Существует большое количество методов фильтрации шума, включая:

1. Гауссов фильтр: используется для размытия изображения с целью удаления высокочастотного шума;
2. медианный фильтр: заменяет значение каждого пикселя на медиану значений в его окрестности, эффективно устраняя импульсные шумы;(МЕТОДИКА УСТРАНЕНИЯ ШУМА НА ОКТИЗОБРАЖЕНИЯХ)
3. частотная фильтрация: применяется в частотной области для удаления определенных частотных компонентов, связанных с шумом.(<https://www.xn--8sbempcldwd3bmt.xn--plai/article/16686>)

Эффективная фильтрация фонового шума особенно важна при обработке изображений с текстом, так как наличие шума может затруднить или сделать невозможным корректное распознавание символов.

Для облегчения работы модели в дальнейшем, для графических САРТСНА в качестве предобработки используется нормализация и ручная разметка изображений датасета.

Нормализация – процесс приведения изображений к единому стандарту по определенным характеристикам, таким как размер, яркость, контрастность или геометрические параметры. Целью нормализации является обеспечение однородности входных данных для последующей обработки или анализа.(Gonzalez, R. C., & Woods, R. E. (2018). Digital Image Processing (4th Edition). Pearson)

Разметка изображений – процесс аннотирования изображений с целью идентификации и классификации объектов или областей интереса на изображении. Разметка является ключевым этапом в подготовке данных для обучения моделей компьютерного зрения и машинного обучения.

Разметка изображений обеспечивает модели данными, необходимыми для обучения точному распознаванию и интерпретации визуальной

информации. Размеченные фотографии используются в различных задачах, от обнаружения объектов до автоматической обработки изображений. ([https://annotate.ru/blog/semanticheskaya\\_razmetka\\_dlya\\_mashinnogo\\_obucheniya](https://annotate.ru/blog/semanticheskaya_razmetka_dlya_mashinnogo_obucheniya))

Дальнейшие подходы и методы связаны непосредственно с поиском решения задания САРТСНА и также специфичны для каждой реализации САРТСНА.

Распознавание речи – технология преобразования устной речи в текстовую форму. Современные системы ASR (Automatic Speech Recognition) используют глубокие нейронные сети для обработки аудиосигналов и преобразования их в текст. (<https://www.assemblyai.com/blog/what-is-asr>)

Преобразование речи в текст – процесс транскрибирования устной речи в письменную форму. Этот процесс включает в себя анализ аудиосигнала, выделение речевых признаков и преобразование их в текст. (<https://h2o.ai/wiki/speech-to-text/>)

Задача сегментации представляет из себя процесс разделения изображения на дискретные группы пикселей для обнаружения важных участков изображения или объектов. В задачах компьютерного зрения, таких как распознавание текста на изображениях, сегментация помогает выделить отдельные символы или слова для последующей обработки. (<https://www.ibm.com/think/topics/image-segmentation>)

Задача классификации представляет собой процесс присвоения входным данным одной из predetermined категорий. В контексте обработки изображений, классификация может использоваться для определения наличия текста на изображении, для распознавания символов или различных объектов. (<https://www.sciencedirect.com/topics/engineering/image-classification>)

Распознавание последовательности символов – это задача извлечения и интерпретации последовательности символов из входных данных, таких как изображение или аудиосигнал. В системах OCR (оптического распознавания

символов) это включает в себя определение порядка символов и их преобразование в текст. ([https://aws.amazon.com/what-is/ocr/?nc1=hl\\_s](https://aws.amazon.com/what-is/ocr/?nc1=hl_s))

Аугментация данных – метод увеличения объема обучающего набора данных путем применения различных трансформаций к существующим данным. В задачах обработки изображений это может включать повороты, масштабирование, изменение яркости и контрастности. (<https://www.ibm.com/think/topics/data-augmentation>)

Генерация синтетических датасетов – процесс создания искусственных данных для обучения моделей машинного обучения. Это особенно полезно, когда реальные данные ограничены или трудно доступны. (<https://www.nvidia.com/en-us/glossary/synthetic-data-generation/>)

Детекция объектов – задача определения и локализации объектов на изображении. В системах САПТСНА это может использоваться для идентификации определенных элементов, таких как дорожные знаки или транспортные средства. (<https://www.ibm.com/think/topics/object-detection>)

Поиск и сопоставление с шаблоном – метод обработки изображений, используемый для поиска частей входного изображения (большого изображения или целевого изображения), которые соответствуют шаблонному изображению (эталонному изображению или меньшему изображению). Сопоставление шаблонов обычно используется для задач обнаружения объектов, распознавания изображений и распознавания образов. (<https://blog.roboflow.com/template-matching/>)

Также, задача автоматизации решения САПТСНА была бы нерешаема без непосредственного взаимодействия с браузером без участия пользователя, для чего применяются описанные ниже подходы.

Анализ структуры HTML-страницы – процесс разбора и интерпретации HTML-кода web-страницы для извлечения информации или автоматизации взаимодействия с элементами страницы. (<https://www.geeksforgeeks.org/html-parsing-and-processing/>)

Автоматизация действий пользователя – это использование программных средств для имитации действий пользователя, таких как клики мышью, ввод текста или навигация по интерфейсу. Это может применяться для автоматического прохождения CAPTCHA или тестирования пользовательских интерфейсов. (<https://www.testresults.io/definitions/automated-user-interaction>)

## 2.2. Архитектуры нейронных сетей для различных форматов CAPTCHA

В данной работе рассмотрены три наиболее популярные и частовстречающиеся реализации CAPTCHA, которые применяются для защиты web-ресурсов: аудио CAPTCHA, текстовые CAPTCHA и графические CAPTCHA (CAPTCHA с изображениями). Для каждой реализации необходим свой подход к решению, разный набор инструментов и библиотек.

Далее для каждой из реализаций будут рассмотрены различные архитектуры нейронных сетей, которые могут быть использованы для автоматизации решения CAPTCHA.

### **Архитектуры нейронных сетей для текстовых CAPTCHA**

Для задачи решения текстовых CAPTCHA могут быть использованы различные модели нейронных сетей, которые поддерживают обработку последовательностей различной длины. Среди таких архитектур и инструментов можно выделить следующие:

1. оптическое распознавание символов (Tesseract OCR);
2. сверточные рекуррентные нейронные сети с функцией потерь Connectionist Temporal Classification (CRNN + CTC);
3. архитектура последовательного обучения (Sequence-to-Sequence).

Tesseract OCR – это свободный движок оптического распознавания текста, изначально разработанный в Hewlett-Packard в 1985–1995 годах и в дальнейшем выпущенный как open-source-проект Google в 2005 году. В настоящее время он поддерживается и активно развивается сообществом под эгидой Google. С версии 4.0 Tesseract использует глубокую нейросетевую архитектуру, что значительно улучшило его производительность на задачах



распознавания сложных текстов и документов.(An Overview of the Tesseract OCR Engine, Adrian Rosebrock. OCR with OpenCV, Tesseract, and Python. PyImageSearch.)

Основные этапы обработки текста в Tesseract:(Adrian Rosebrock. OCR with OpenCV, Tesseract, and Python. PyImageSearch.)

1. предобработка изображения: включает бинаризацию, удаление шумов, коррекцию наклона и выравнивание; эти шаги направлены на улучшение качества входного изображения для более точного распознавания;
2. разметка структуры документа: изображение разбивается на блоки, строки и отдельные слова; сначала происходит анализ макета документа, включая идентификацию текста, таблиц, изображений и других структур;
3. распознавание текста: начиная с версии 4.0, Tesseract использует LSTM (Long Short-Term Memory) – разновидность рекуррентных нейросетей, позволяющую учитывать контекст в последовательности символов; распознавание не требует предварительной сегментации символов – вместо этого используется подход Connectionist Temporal Classification (CTC), обеспечивающий сопоставление входной последовательности признаков с выходной строкой текста;
4. постобработка: включает коррекцию ошибок с помощью встроенных языковых моделей, фильтрацию недопустимых символов и оптимизацию финального вывода текста.

Нейросетевой движок в Tesseract 4.0 построен следующим образом:(An Overview of the Tesseract OCR Engine)

1. входное изображение пропускается через сверточный слой для извлечения признаков;
2. далее признаки передаются в двунаправленные LSTM-ячейки, способные учитывать как предыдущий, так и последующий контексты;

3. выход LSTM обрабатывается функцией CTC, которая выравнивает вероятности символов по времени и строит финальную строку текста.

Архитектура движка Tesseract на примере обработки входной последовательности приведена на рис. 2.1.

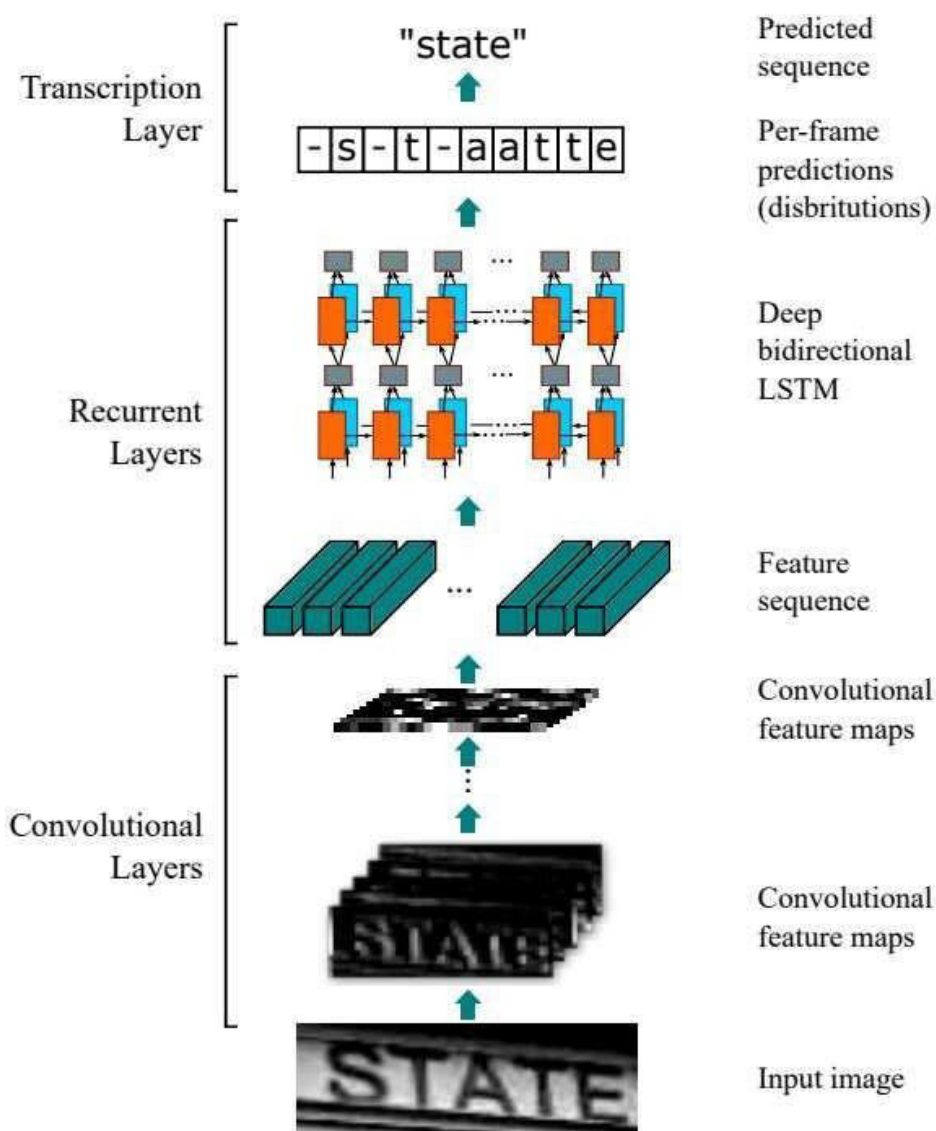


Рис. 2.1 Архитектура нейросетевого движка Tesseract OCR.

Этот подход особенно устойчив к искажениям и перемещениям текста, поскольку модель обучается «видеть» всю строку целиком, а не по символам.

Преимущества Tesseract: (Adrian Rosebrock. OCR with OpenCV, Tesseract, and Python. PyImageSearch.)

1. гибкость: поддержка множества языков, включая написание справа-налево, китайский, японский и арабский;

2. обучаемость: пользователь может дообучать модель на своих собственных данных;
3. интеграция: легко используется совместно с библиотеками обработки изображений, такими как OpenCV.

Архитектура CRNN (Convolutional Recurrent Neural Network) с CTC (Connectionist Temporal Classification) – это специализированный подход к распознаванию последовательностей на входах переменной длины. Он широко применяется в OCR, распознавании речи и жестов, где отсутствует строгое выравнивание между входами и метками. (Ian Goodfellow, Yoshua Bengio, Aaron Courville. Deep Learning)

Данный подход объединяет сверточные нейронные сети (для извлечения признаков), рекуррентные нейронные сети (для моделирования зависимости во времени/пространстве) и CTC (для сопоставления выходной последовательности с целевым результатом без предварительной сегментации).

Первая составляющая архитектуры – сверточная нейронная сеть (CNN), задача которой – извлечь дискретные признаки из исходного изображения (например, строки текста, номера или CAPTCHA).

Архитектура сверточных нейронных сетей состоит из комбинации слоев:

1. сверточные слои, извлекают локальные пространственные шаблоны (например, края, формы);
2. пуллинговые слои, которые уменьшают размерность и повышают инвариантность;
3. выходной слой, который представляет собой матрицу признаков, часто сжатую по вертикали, но сохраняющую пространственную структуру по ширине, что формирует последовательность признаков, где каждый вектор соответствует «временной метке» (например, потенциальному символу).

После получения последовательности признаков из CNN, она поступает на вход рекуррентной нейронной сети для анализа временной структуры

последовательности признаков. Наиболее часто используется двунаправленная LSTM, поскольку она обрабатывает последовательность слева направо и справа налево, позволяя учитывать как предшествующий, так и последующий контекст, что особенно важно при наличии похожих символов и искажений. (Ian Goodfellow, Yoshua Bengio, Aaron Courville. Deep Learning)

Результатом является последовательность выходных векторов, каждый из которых соответствует одному «временному шагу» – потенциальному символу.

Выходы из рекуррентной части подаются на полносвязный слой (или линейную проекцию), где каждый вектор переводится в вектор вероятностей по алфавиту (включая специальный пустой символ, обозначающий отсутствие выхода).

Затем применяется Connectionist Temporal Classification – специализированная функция потерь, предназначенная для задач, где отсутствует выравнивание между входной и выходной последовательностью. (Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks)

Connectionist Temporal Classification обладает следующими особенностями:

1. не требует сегментации или аннотированных позиций символов;
2. поддерживает множество путей к одному выходу;
3. вводит символ «blank» для обозначения промежутков или неопределенности;
4. обеспечивает обучение «end-to-end» на уровне целых строк текста.

Архитектура Sequence-to-Sequence (Seq2Seq) представляет собой класс нейронных сетей, предназначенных для преобразования одной последовательности элементов в другую, при этом длины входной и выходной последовательностей могут различаться. Первоначально эта архитектура была предложена для задач машинного перевода (<https://arxiv.org/abs/1409.3215>), однако в последующем она нашла широкое применение и в других областях, та-

ких как распознавание речи, автоматическая транслитерация, генерация текстов и, в частности, распознавание текстовых CAPTCHA.

Основу Seq2Seq составляет двухкомпонентная архитектура: энкодер и декодер. Энкодер последовательно обрабатывает входные данные (например, текст или объекты, извлеченные из изображения с помощью сверточной сети) и кодирует их в вектор фиксированной длины – так называемый вектор контекста. Декодер, в свою очередь, получает этот вектор и генерирует выходную последовательность, по одному элементу за шаг, используя скрытое состояние и ранее сгенерированные элементы.

Классическая реализация Seq2Seq использует рекуррентные нейронные сети (RNN), включая их модификации – LSTM и GRU. В энкодере последовательность входных векторов обрабатывается пошагово, и финальное скрытое состояние используется как компактное представление всей последовательности. Это состояние затем передается в декодер, который генерирует выход, начиная с начального маркера.

Одной из ключевых проблем базовой Seq2Seq-модели является невозможность эффективно работать с длинными входными последовательностями, поскольку информация вектора контекста может быть потеряна. Для преодоления этого ограничения была предложена модификация с использованием механизма внимания, позволяющего декодеру на каждом шаге фокусироваться на различных частях входной последовательности. (<https://arxiv.org/abs/1508.04025>).

### **Архитектуры нейронных сетей для графических CAPTCHA**

При решении графических CAPTCHA важными являются возможности модели по детекции и сегментации объектов, поскольку данные CAPTCHA могут требовать как обычного поиска объекта, так и выбора клеток, в которых содержится объект. Для решения данных задач могут применяться следующие инструменты и архитектуры нейронных сетей:

1. одноэтапная модель для детекции объектов (YOLO, You Only Look Once);

2. модель детекции, основанная на архитектуре трансформера (DETR, DEtection TRansformer);
3. модель сверточной нейронной сети для двухэтапного обнаружения объектов (Faster R-CNN).

YOLO (You Only Look Once) – семейство нейросетевых моделей, разработанных для реализации задачи детекции объектов в изображениях в реальном времени. В отличие от традиционных методов, которые разделяют задачу на два этапа (предсказание регионов и их классификация), YOLO выполняет эти задачи одновременно в рамках единой сверточной нейросети. Первая версия YOLO была представлена Джозефом Редмоном и его коллегами в 2016 году, с тех пор были разработаны многочисленные улучшенные версии (последняя версия на данный момент YOLOv11), каждая из которых повышала точность и скорость работы.

YOLO разделяет входное изображение на сетку  $S \times S$ , для каждой ячейки сеть предсказывает  $B$  ограничивающих рамок, а также вероятности принадлежности к каждому из классов. Каждое предсказание содержит следующие параметры:

1. координаты ограничивающей рамки (центр  $x$ ,  $y$ , ширина  $w$ , высота  $h$ );
2. уверенность модели в том, что в рамке находится объект;
3. вероятности классов (условные вероятности при наличии объекта).

Выход модели представляет собой тензор с размерностью  $S \times S \times (B * 5 + C)$ , где  $C$  – количество классов.

YOLO использует «end-to-end» обучение, что позволяет быстро и эффективно предсказывать объекты и их классы за один проход изображения через сеть. Это особенно важно при применении в задачах САПТСНА, где требуется обрабатывать большое количество мелких изображений в режиме реального времени.

YOLOv11 представляет собой значительный шаг вперед в развитии моделей детекции объектов реального времени. Она сочетает в себе высокую

точность, эффективность и универсальность, что делает ее применимой в различных задачах компьютерного зрения, включая распознавание объектов на изображениях CAPTCHA.(<https://arxiv.org/abs/1506.02640>)

Основные компоненты архитектуры YOLOv11:(<https://arxiv.org/abs/2410.17725>)

1. основная сеть: YOLOv11 использует улучшенный каркас для извлечения признаков, основанный на блоках C3K2 (Cross Stage Partial) с ядром  $2 \times 2$ , эти блоки обеспечивают эффективное извлечение признаков при меньших вычислительных затратах, сохраняя при этом способность модели захватывать важные особенности изображения;
2. связующий модуль: модель внедряет модуль SPPF (Spatial Pyramid Pooling – Fast), который позволяет эффективно обрабатывать объекты различных масштабов, улучшая способность модели к детекции как крупных, так и мелких объектов;
3. механизмы внимания: YOLOv11 интегрирует блок C2PSA (Convolutional block with Parallel Spatial Attention), который усиливает способность модели фокусироваться на релевантных частях изображения, улучшая точность детекции, особенно в сложных сценах;
4. выходной слой: выходной слой модели оптимизирован для быстрого и точного предсказания ограничивающих рамок и классов объектов, обеспечивая высокую производительность в реальном времени.

Модель YOLOv11 обладает следующими преимуществами:

1. высокая точность: благодаря улучшенному извлечению признаков и механизмам внимания YOLOv11 демонстрирует высокие показатели точности в задачах детекции объектов;
2. эффективность: оптимизированная архитектура обеспечивает быстрое выполнение модели, что особенно важно для приложений, требующих обработки в реальном времени;

3. универсальность: модель поддерживает различные задачи компьютерного зрения, включая детекцию объектов, сегментацию, классификацию и определение позы, что делает ее применимой в широком спектре приложений.

Также, примером современной архитектуры для задачи обнаружения объектов является DETR (DEtection TRansformer) – модель, предложенная исследователями из Facebook AI Research в 2020 году (<https://arxiv.org/abs/2005.12872>). В отличие от традиционных сверточных решений, DETR применяет архитектуру трансформера, ранее широко использовавшуюся в области обработки естественного языка, для анализа визуальных данных. Основной особенностью DETR является переосмысление задачи детекции как задачи сопоставления предсказаний с реальными объектами на изображении в постановке «многие ко многим».

Архитектура DETR состоит из трех основных компонентов: извлекающей признаковую информацию сверточной нейронной сети, архитектуры трансформера (включающей энкодер и декодер), а также выходной регрессионно-классификационной части. (<https://arxiv.org/abs/2005.12872>)

На начальном этапе используется сверточная нейронная сеть, такая как ResNet-50 или ResNet-101, предназначенная для извлечения признаков изображения. На выходе формируется тензор размерности  $C \times H \times W$ , где  $C$  – число каналов,  $H$  и  $W$  – пространственные размеры. Этот тензор преобразуется в последовательность векторов, которая поступает на вход трансформера. К каждому вектору добавляется позиционная информация, необходимая для сохранения пространственной структуры изображения.

Следующим этапом является применение трансформера, включающего энкодер и декодер:

1. энкодер принимает последовательность признаков и обрабатывает их с помощью многоголового механизма самовнимания (multi-head self-attention), формируя глобальное представление изображения;



2. декодер работает с фиксированным набором обучаемых векторов, называемых объектными запросами, каждый из которых предназначен для предсказания одного потенциального объекта, декодер на каждом шаге взаимодействует с выходами энкодера, формируя выходные векторы, соответствующие найденным объектам.

Каждый из выходных векторов декодера передается в многослойный перцептрон, предсказывающий класс объекта и координаты ограничивающего прямоугольника. Сопоставление между предсказанными и реальными объектами выполняется с использованием венгерского алгоритма, что позволяет обучать модель в режиме «end-to-end» без применения процедур постобработки, таких как подавление немаксимумов.

Модель DETR обладает рядом преимуществ по сравнению с традиционными подходами:

1. архитектура позволяет отказаться от использования якорей и эвристических алгоритмов;
2. глобальное внимание трансформера способствует лучшему захвату контекста;
3. прямое сопоставление предсказаний с объектами упрощает процедуру обучения;
4. обучение происходит в парадигме «end-to-end».

При этом, оригинальная версия DETR характеризуется сравнительно медленной сходимостью, особенно при обучении на малых объемах данных, поэтому с момента появления оригинальной версии модели было предложено несколько улучшений, направленных на повышение скорости сходимости и качества предсказаний:

1. Deformable DETR – введение деформируемого внимания, ограничивающего область восприятия и позволяющего ускорить обучение;(<https://arxiv.org/abs/2010.04159>)
2. DN-DETR – использование стратегии шумоустойчивого обучения;(<https://arxiv.org/abs/2203.01305>)

3. DINO – улучшенная инициализация запросов и усиленное взаимодействие между предсказаниями. (<https://arxiv.org/abs/2203.03605>)

Faster R-CNN (Region-based Convolutional Neural Network) – одна из широко используемых архитектур для задачи обнаружения объектов на изображениях. Модель была представлена исследователями из Microsoft Research в 2015 году и стала логическим развитием предыдущих архитектур – R-CNN и Fast R-CNN. (<https://arxiv.org/abs/1506.01497>)

Главным отличием Faster R-CNN является интеграция модуля генерации предложений (регионов интереса) непосредственно в архитектуру нейронной сети, что позволило устранить узкие места в скорости и повысить эффективность обнаружения.

Архитектура Faster R-CNN состоит из трех ключевых компонентов:

1. сверточная нейронная сеть для извлечения признаков;
2. сеть генерации предложений;
3. модуль классификации и регрессии объектов.

На первом этапе изображение обрабатывается глубокой сверточной нейронной сетью, например, ResNet-50, VGG-16 или Inception, для извлечения карты признаков. Эта карта представляет собой компактное абстрактное представление содержимого изображения, на основе которого осуществляется последующий анализ и обнаружение объектов.

После чего, специализированный сверточный модуль, встроенный в архитектуру, который скользит по карте признаков и формирует якоря – прямоугольные области различного размера и пропорций. Для каждого якоря сеть оценивает:

1. вероятность наличия объекта;
2. координаты смещения ограничивающего прямоугольника.

Из множества сгенерированных предложений отбираются наиболее вероятные при помощи процедуры подавления немаксимальных значений, устраняющей избыточные перекрывающиеся прямоугольники.

Отобранные регионы интереса приводятся к фиксированному размеру, после чего проходят через серию полносвязных слоев. Далее параллельно производятся следующие действия:

1. классификация объекта по заранее определенным классам;
2. уточнение координат ограничивающего прямоугольника для каждого объекта.

Среди особенностей Faster R-CNN можно выделить:

1. двухэтапную архитектуру, которая обеспечивает более высокую точность по сравнению с одноэтапными моделями (например, YOLO), особенно при работе с мелкими объектами;
2. совместное обучение генератора предложений и детектора объектов улучшает интеграцию и повышает сходимость модели;
3. гибкость архитектуры позволяет использовать различные сверточные сети в качестве основы, адаптируя модель под конкретные задачи.

В то же время данная архитектура не лишена недостатков, среди которых можно выделить следующие:

1. относительно низкая скорость работы по сравнению с одноэтапными моделями делает ее менее предпочтительной для приложений в реальном времени;
2. чувствительность к гиперпараметрам якорей неоптимальный выбор которых может существенно снизить итоговую точность модели.

### **Архитектуры нейронных сетей для аудио CAPTCHA**

Для решения задач распознавания и анализа аудио CAPTCHA могут применяться следующие архитектуры и инструменты:

1. облачные API речевого распознавания, основанных на языковых моделях (Google Speech-to-Text);
2. модели на основе сверточных и рекуррентных нейронных сетей, обученных на спектрограммах аудиосигналов (CRNN);

3. модели автоматического распознавания речи, основанные на архитектуре трансформеров (Wav2Vec 2.0).

Поскольку архитектуры на основе трансформеров и сверточно-рекуррентных нейронных сетей уже были подробно рассмотрены ранее, а принципы их построения сохраняются при применении к различным типам задач, включая работу с аудиосигналами, дополнительные пояснения в данном разделе не требуются.

Еще одним из широко используемых инструментов для автоматического распознавания речи является облачный сервис Google Speech-to-Text. Этот API предоставляет доступ к масштабируемым и постоянно обновляемым языковым моделям, разработанным компанией Google для преобразования аудиосигнала в текст. Сервис поддерживает более 125 языков и диалектов, а также различные режимы работы, включая потоковое и пакетное распознавание. (<https://cloud.google.com/speech-to-text>)

Архитектура Google Speech-to-Text основана на современных нейросетевых подходах, включая глубокие нейронные сети и архитектуры с механизмами внимания, а также модели-трансформеры. Google применяет «end-to-end» модели, обученные на больших наборах аудиоданных, с использованием Connectionist Temporal Classification и Sequence-to-Sequence с механизмами внимания в качестве основных стратегий декодирования аудиопотока. (<https://arxiv.org/abs/1610.03022>)

Google Speech-to-Text обладает следующими преимуществами:

1. высокая точность распознавания даже в условиях шума;
2. масштабируемость и возможность интеграции в облачные приложения;
3. поддержка множества языков и специализированных терминологий;
4. возможность использования в реальном времени;
5. доступ к регулярно обновляемым языковым моделям Google.

### 2.3. Подготовка и аннотация датасетов

#### **Подготовка датасета для текстовых CAPTCHA**

Современные текстовые CAPTCHA обычно состоят из букв и цифр. Зачастую, используются символы латинского алфавита (как прописные, так и строчные) и цифры от 0 до 9. Но обычно реализации исключают символы, которые могут быть легко перепутаны, например, буква «O» и цифру «0», буквы «I» и «l» и другие. Рекомендуемый набор символов в генераторах на некоторых CRM платформах выглядит следующим образом: ABCDEFGHJKLMNPQRSTWXYZ23456789 [Bitrix].

Длина последовательности символов в CAPTCHA обычно составляет от 4 до 8 символов, что обеспечивает баланс между удобством для пользователя и безопасностью, однако конкретная длина может варьироваться в зависимости от требований системы безопасности.

Для усложнения автоматического распознавания текстовые CAPTCHA подвергаются различным искажениям:

1. геометрические искажения: символы могут быть искажены, повернуты или наклонены, что затрудняет их распознавание автоматическими системами [BrightData];
2. перекрытие символов: символы могут быть расположены близко друг к другу или даже перекрываться, что усложняет их сегментацию и последующее распознавание [Proglib];
3. добавление шума: на изображение могут быть добавлены различные шумы, такие как линии, точки или круги, чтобы затруднить распознавание символов;
4. сложные фоны: использование фонов с различными цветами или узорами, что делает выделение символов более сложным [NVJournal];
5. нелинейные искажения: применение нелинейных трансформаций к символам, что делает их форму менее предсказуемой для автоматических систем распознавания [Simai].

Эти методы направлены на повышение сложности автоматического распознавания CAPTCHA, сохраняя при этом относительную легкость распознавания для человека.

Текстовые CAPTCHA на сегодняшний день уже не являются настолько же широкоиспользуемыми как CAPTCHA с изображениями, в связи с чем получение достаточного количества изображений для формирования датасета является трудоемкой задачей.

Качество используемого датасета оказывает существенное влияние на итоговую точность работы модели. Для эффективного обучения необходимо, чтобы набор данных соответствовал следующим требованиям:

1. достаточное количество изображений для каждого символа, что обеспечивает статистическую устойчивость модели;
2. разнообразие данных, включающее:
  - 2.1. различные углы наклона символов;
  - 2.2. вариативность написания символов и их искажения;
  - 2.3. наличие побочных визуальных элементов, создающих препятствия для распознавания;
  - 2.4. использование различных шрифтов.
3. переменная длина последовательностей символов, что позволяет модели адаптироваться к разным формам CAPTCHA.

Включение указанных факторов способствует обучению модели на более широком спектре признаков, что, в свою очередь, повышает ее способность к обобщению на ранее невидимых данных.

Поскольку в открытом доступе отсутствует достаточное количество данных для формирования сбалансированного датасета, необходимо использовать другие способы для получения разнообразных примеров. Среди таких способов наиболее удобным и подходящим для данной задачи является генерация синтетических изображений с использованием специализированных библиотек. В качестве основного инструмента для решения данной задачи, зачастую, используется библиотека `captcha` на языке Python, обладающая

необходимым функционалом для создания изображений CAPTCHA с заданными параметрами. Данная библиотека поддерживает генерацию изображений с пользовательскими шрифтами и различными эффектами искажений, что исключает необходимость привлечения дополнительных инструментов.

В рамках данной работы был разработан собственный генератор синтетических текстовых CAPTCHA с использованием данной библиотеки. Исходный код генератора синтетических CAPTCHA представлен в приложении (листинг 3).

После создания изображений все они прошли этапы предобработки, направленные на улучшение качества данных и повышение эффективности обучения модели. Предобработка включала следующие этапы:

1. преобразование изображений в градации серого для уменьшения количества каналов и снижения вычислительной нагрузки;
2. бинаризация изображений с целью получения контрастного представления символов (белый текст на черном фоне);
3. удаление шумов и фона с использованием морфологических операций, в частности, дилатации.

Исходный код обработчика изображений представлен в приложении (листинг 4).

Примеры сгенерированных и предобработанных CAPTCHA приведены на рисунке ниже:



Рис. 2.2 Изображения CAPTCHA: а) – сгенерированное изображение, б) – результат обработки.

### **Подготовка датасета для CAPTCHA с изображениями**

Большинство предобученных моделей компьютерного зрения, таких как YOLOv8, обучены на датасете COCO [COCO], содержащем изображения

высокого качества с четкими контурами и однозначной аннотацией объектов. Однако CAPTCHA с изображениями имеют принципиально иные характеристики: они могут включать в себя размытие, наложенные артефакты, искажения, шумы, повторяющиеся элементы и искусственно пониженное разрешение. Все это снижает эффективность использования стандартных датасетов и моделей, не адаптированных под такие условия.

Для обеспечения высокой точности в задаче автоматического решения CAPTCHA необходимо подготовить собственный набор данных, приближенный к реальным условиям использования. Наиболее эффективным методом является автоматизированный парсинг изображений CAPTCHA, представленных на web-сайтах, использующих визуальные CAPTCHA-решения, такие как Google reCAPTCHA v2.

Использование реальных CAPTCHA, собранных в автоматическом режиме, имеет ряд преимуществ по сравнению с синтетической генерацией данных:

1. изображения содержат разнообразные сцены, освещение, углы обзора и уровни шума, что положительно влияет на способность модели к обобщению;
2. присутствует большое количество уникальных объектов на фоне, в том числе в частично перекрытых и смазанных вариантах;
3. отсутствует необходимость в ручной генерации изображений и создании дополнительных искажений для повышения реалистичности;
4. возможно извлекать текстовые инструкции к CAPTCHA, что позволяет соотносить каждое изображение с требуемым классом.

Для парсинга CAPTCHA необходим автоматизированный сценарий взаимодействия с браузером с использованием библиотеки Selenium [Selenium]. Данный подход позволяет воспроизвести действия пользователя при работе с CAPTCHA, обходя при этом ручной ввод. Для



обеспечения стабильной работы и масштабируемости процесса применяется браузерная автоматизация через WebDriver (в частности, ChromeDriver).

Функциональность парсера включает следующие ключевые этапы:

1. поиск `iframe`-элемента, содержащего чекбокс «Я не робот», и эмуляция клика по нему для инициирования визуальной CAPTCHA;
2. ожидание загрузки CAPTCHA и извлечение изображения с заданием (включая его URL или пиксельный снимок);
3. извлечение информации о структуре сетки (количество строк и столбцов), на которую разбито изображение CAPTCHA;
4. получение текста задания, содержащего имя объекта (например, «выберите все изображения с мотоциклами»), для последующего использования в аннотации данных.

Типичная CAPTCHA представляет собой изображение, разделенное на сетку из  $3 \times 3$  или  $4 \times 4$  ячеек, каждая из которых может содержать фрагмент сцены. При этом пользователю предлагается выбрать ячейки, в которых присутствует объект заданного класса. Процесс парсинга может быть представлена блок-схемой на рис. 2.3.

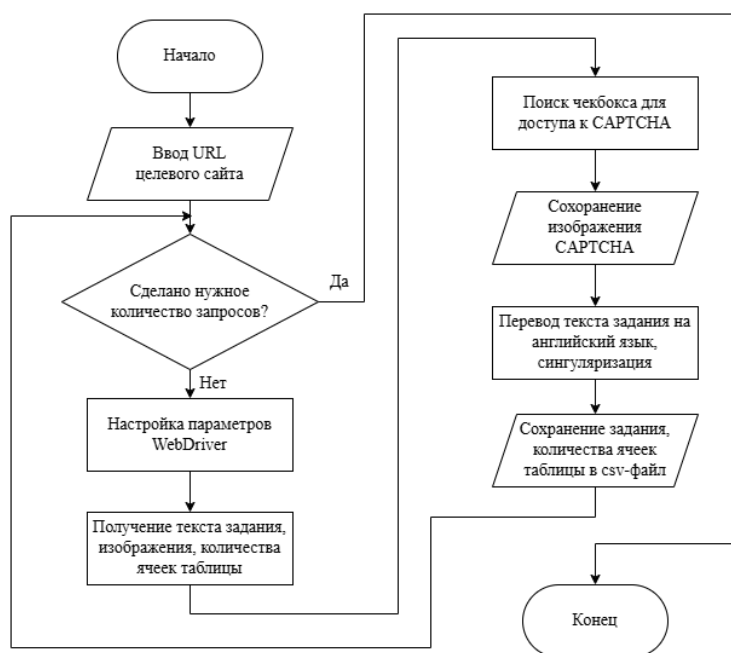


Рис. 2.3 Блок-схема процесса парсинга CAPTCHA.

Полученные изображения и метаданные (включая текст задания и параметры сетки) используются для формирования обучающего датасета, пригодного для дообучения модели YOLOv8 в задачах классификации и сегментации объектов.

После получения достаточного количества изображений для составления датасета необходимо провести их предварительную обработку и разметку. Это один из самых важных этапов работы, поскольку от качества разметки напрямую зависит точность и эффективность последующей работы модели.

Для создания меток используется инструмент CVAT (Computer Vision Annotation Tool) – многофункциональное веб-приложение с поддержкой аннотации объектов с помощью полигонов, прямоугольников и других форм. CVAT позволяет экспортировать разметку напрямую в формат, совместимый с YOLO [CVAT].

Поскольку САРТСНА-изображения часто содержат объекты с нечеткими контурами, наложением и визуальными искажениями, особенно важно использовать ручную точную разметку, а не ограничиваться автоматическими методами. Выделение объектов должно проводиться как можно точнее, с учетом геометрии контуров. На рисунке ниже представлен пример изображения с размеченными объектами:



Рис. 2.4 Пример разметки изображения с тестовой САРТСНА.

Кроме того, разметка позволяет учесть сразу несколько объектов разных классов на одном изображении, что особенно характерно для САРТСНА, где в одной сетке могут одновременно находиться, например, автомобили и автобусы. Такой подход положительно влияет на обобщающую способность модели.

В случае, если количество данных по отдельным классам окажется недостаточным, можно дополнительно использовать методы аугментации: вращение, масштабирование, искажение цвета и контраста. Однако при хорошо организованном парсинге и разметке зачастую удастся обойтись без аугментации.

Кроме того, для корректного обучения модели YOLO требуется создать иерархическую структуру папок, в которой изображения и соответствующие метки будут разделены на тренировочную и валидационную выборки. Стандартная структура включает следующие директории:

1. директория `train` – содержит тренировочную выборку:
  - 1.1. директория `images` – изображения;
  - 1.2. директория `labels` – метки к изображениям.
2. директория `val` – содержит валидационную выборку:
  - 2.1. директория `images` – изображения;
  - 2.2. директория `labels` – метки к изображениям.

Набор классов, пути к выборкам и параметры конфигурации задаются в YAML-файле, который передается при обучении модели. Содержимое такого файла для данной модели:

Листинг 2.1

#### Параметры конфигурации для обучения модели

---

```
1 path: ../datasets/image_dataset
2 train: images/train
3 val: images/val
4
5 nc: 9 # Количество классов
6 names: ['pedestrian transition', 'stair', 'motorcycle', 'bus', 'traffic light', 'car', 'bicycle', 'fire hydrant', 'tractor']
```

---

### 3. ЭКСПЕРИМЕНТАЛЬНАЯ ЧАСТЬ И ИССЛЕДОВАТЕЛЬСКИЙ ВКЛАД

#### 3.1. Сравнительный анализ моделей нейронных сетей для автоматизированного решения САРТСНА в текстовом формате

Для распознавания текста с переменной длиной последовательности в задачах САРТСНА наиболее часто применяются следующие из описанных ранее инструменты и архитектуры нейронных сетей:

1. оптическое распознавание символов (Tesseract OCR);
2. сверточные рекуррентные нейронные сети с функцией потерь Connectionist Temporal Classification (CRNN + CTC);
3. архитектура последовательного обучения (Sequence-to-Sequence).

С целью выбора наиболее эффективной модели были реализованы и протестированы все указанные подходы, после чего была выбрана архитектура, обеспечивающая наилучшую точность предсказаний.

Для обучения моделей был сформирован датасет из 100 000 изображений САРТСНА, содержащих случайные последовательности символов длиной от 4 до 7. Такой объем данных позволяет добиться высокой обобщающей способности модели и снизить вероятность переобучения.

#### **Оптическое распознавание символов (OCR Tesseract)**

Изначально была реализована модель с использованием OCR, поскольку такие системы изначально разрабатывались для задач оптического распознавания текста. В качестве конкретной модели был выбран Tesseract.

Для решения поставленной задачи использовалась предобученная модель Tesseract, которая была дообучена на специализированном датасете, содержащем изображения САРТСНА с характерными искажениями. Однако, в ходе экспериментов было установлено, что точность распознавания последовательностей символов целиком составляла 0%, а точность для отдельных символов оказалась крайне низкой. Это связано с тем, что архитектура Tesseract недостаточно устойчива к искажениям, характерным для

САРТСНА, таким как деформация символов, наложение шумов и изменение углов наклона [TrainTesseract].

Таким образом, было принято решение отказаться от использования Tesseract в пользу более адаптированных к данной задаче моделей, таких как сверточные рекуррентные нейронные сети (CRNN) или модели последовательного обучения (Sequence-to-Sequence), обладающие высокой устойчивостью к вариативности и искажениям, характерным для САРТСНА.

### **Рекуррентные сверточные нейронные сети (CRNN)**

Разработанная модель CRNN для распознавания САРТСНА включает в себя три ключевых блока:

1. сверточный блок (CNN): предназначен для выделения признаков из изображений САРТСНА. Включает в себя три последовательных сверточных слоя, а также методы нормализации и уменьшения размерности признакового пространства;
2. рекуррентный блок (RNN): использует двунаправленные слои GRU, позволяющие модели учитывать зависимость между последовательными символами в САРТСНА;
3. выходной слой: полносвязный, который выполняет классификацию каждого символа в последовательности.

В приложении 6 представлена реализация CRNN-модели на языке Python с использованием библиотеки TensorFlow/Keras:

В данной архитектуре применяются слои Dropout для регуляризации, также используется l2-регуляризация, BatchNormalization для ускорения обучения и повышения устойчивости модели, а также функция softmax для предсказания классов символов.

После обучения данной модели результаты оказались превосходящими показатели OCR, однако все же не достигли удовлетворительного уровня. В частности, точность распознавания всей последовательности символов не превышала 10%, тогда как точность классификации отдельных символов составляла около 70%.

## Архитектура последовательного обучения (Seq-to-Seq)

Одним из ключевых элементов реализованной Seq2Seq является механизм внимания, который позволяет декодеру динамически фокусироваться на различных частях входной последовательности при генерации выходных символов [Seq2SeqBook]. Этот подход особенно полезен для распознавания САРТСНА, так как символы в изображениях могут иметь разную ориентацию и степень искажения.

Кодировщик, в данной модели принимает входное изображение САРТСНА и преобразует его в компактное представление. Архитектура кодировщика включает:

1. четыре сверточных блока, слои пакетной нормализации и слои подвыборки для понижения размерности входных данных;
2. глобальный усредненный слой для получения векторного представления изображения;
3. полносвязный слой для финального представления скрытого состояния;
4. рекуррентный слой для кодирования последовательности, возвращающий последнее скрытое состояние кодировщика.

Декодировщик выполняет пошаговую генерацию выходной последовательности, используя скрытое состояние кодировщика. В архитектуру декодировщика входят:

1. входной слой для последовательности токенов;
2. слой вложения, который преобразует входные токены в векторные представления;
3. рекуррентный слой, обрабатывающий последовательность с учетом скрытого состояния кодировщика;
4. механизм внимания, который позволяет декодеру учитывать релевантные части входного изображения;
5. полносвязный слой с функцией активации softmax для предсказания вероятностей символов.

Полная архитектура модели реализована в TensorFlow/Keras и реализация модели приведена в приложении 7.

На начальных этапах экспериментов предложенная Seq2Seq-модель показала наилучшие результаты среди рассмотренных вариантов. В отличие от OCR- и CRNN-моделей, данная архитектура смогла достичь более высокой точности распознавания последовательностей символов, что обусловлено применением механизма внимания. Дальнейшая работа с моделью была сосредоточена на ее оптимизации и улучшении параметров обучения.

### 3.2. Эксперименты по аудио- и графическим CAPTCHA

#### **CAPTCHA в аудио формате**

##### **Выбор языка программирования и инструментов для разработки**

Для разработки сервиса по автоматизации распознавания CAPTCHA был выбран язык программирования Python и библиотека для автоматизации тестирования web-приложений Selenium.

Python обладает рядом преимуществ для решения подобных задач:

1. простота и читаемость кода: Python легко использовать благодаря лаконичному синтаксису, что ускоряет разработку и упрощает поддержку проекта;
2. широкая экосистема библиотек: Для работы с CAPTCHA можно использовать специализированные библиотеки, а также сторонние инструменты для машинного обучения;
3. поддержка скриптового и объектно-ориентированного подхода: Это делает Python гибким для создания как небольших сценариев автоматизации, так и сложных систем.

Selenium выделяется следующими преимуществами среди других инструментов автоматизации:

1. кросс-браузерная поддержка: Selenium поддерживает тестирование во всех популярных браузерах, таких как Chrome, Firefox, Edge и Safari;



2. возможности для взаимодействия с динамическими элементами: Selenium может эмулировать действия пользователя, включая ввод текста, клики и работу с выпадающими меню, что полезно для работы с CAPTCHA;
3. поддержка различных языков программирования: Хотя Python удобен для автоматизации, Selenium можно использовать с Java, C#, Ruby и другими языками;
4. интеграция с другими библиотеками и инструментами: Selenium легко интегрируется с фреймворками для тестирования или системами для распознавания изображений.

Audio CAPTCHA представляет собой элемент, встроенный в веб-страницу, который содержит в себе ссылку на отрезок звуковой дорожки, которая содержит шум и запись голоса.

Подобная запись хорошо поддается распознаванию с использованием современных библиотек для распознавания речи, одна из которых была использована для решения Audio CAPTCHA в данной работе.

В Python существует множество библиотек для распознавания человеческой речи, таких как Google Speech Recognition, Pocketsphinx, DeepSpeech и других [7]. Среди них была выбрана библиотека SpeechRecognition по следующим причинам:

1. удобство использования: Простота в освоении и интеграции благодаря интуитивно понятному API;
2. поддержка нескольких API: Библиотека предоставляет интерфейсы для работы с несколькими сервисами, включая Google Web Speech API, IBM Watson, Microsoft Azure и другие [8].
3. кроссплатформенность: SpeechRecognition работает на Windows, macOS и Linux, что обеспечивает гибкость разработки;
4. встроенные функции обработки звука: Возможность работы с различными форматами аудио, включая wav, и встроенные методы для улучшения качества записи перед отправкой на обработку;

5. локальная и облачная обработка: Поддержка локальных движков, таких как PocketSphinx, и облачных сервисов, таких как Google Speech API, что делает библиотеку универсальной для различных задач;
6. открытый исходный код: Это бесплатная библиотека с открытым кодом, что позволяет исследователям и разработчикам адаптировать ее под свои нужды.

### **Описание алгоритма работы программы**

Процесс получения аудиофайла, содержащего CAPTCHA, с web-сайта для последующего распознавания и отправку готового решения с использованием Selenium можно представить как следующую последовательность шагов:

1. Инициализация настроек браузера;
2. Открытие web-страницы, содержащей CAPTCHA;
3. Переключение на фрейм с чекбоксом CAPTCHA на web-странице и нажатие на чекбокс;
4. Переключение на фрейм с CAPTCHA в виде картинки или набора картинок и нажатие на кнопку доступа к аудиофайлу;
5. Переключение на фрейм с аудиозаписью и поиск элемента, который содержит ссылку на аудиозапись;
6. Создание запроса на получение файла по ссылке;
7. Передача файла в решатель для последующей обработки;
8. Вставка результата распознавания в текстовое поле и подтверждение ввода для успешного решения CAPTCHA.

Для создания запроса на получение файла используется встроенный модуль Python – requests.

Блок-схема, иллюстрирующая приведенный алгоритм представлена на рис. 3.1.

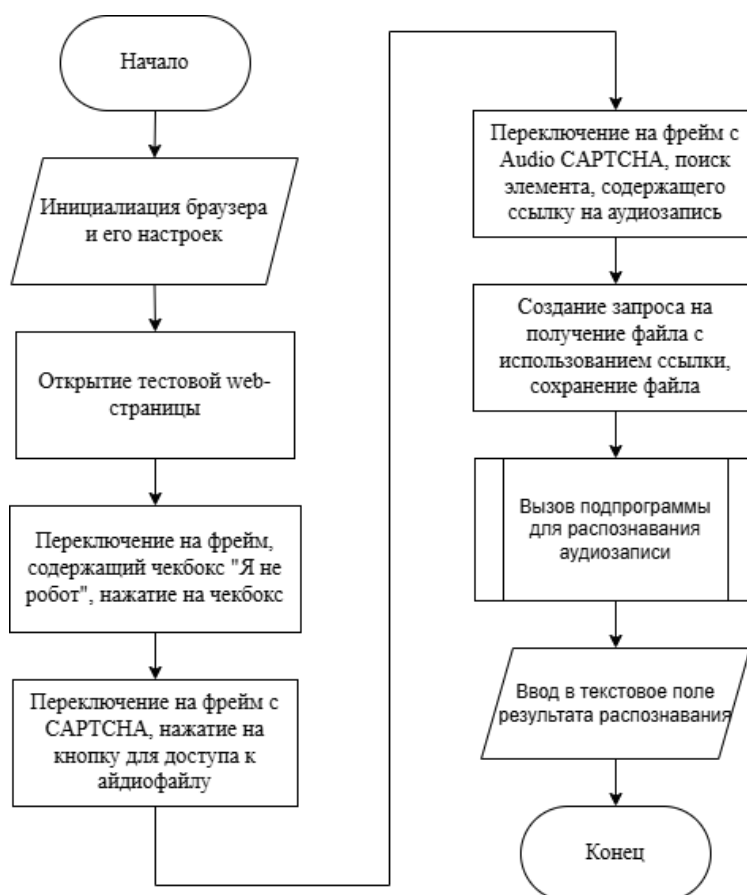


Рис. 3.1 Блок-схема процесса прохождения Audio CAPTCHA.

Процесс обработки аудиофайла, полученного с web-страницы можно разделить на следующие этапы:

1. преобразование полученного аудиофайла в другой формат, который является подходящим для распознавания;
2. распознавание речи в перекодированном файле;
3. сохранение полученного результата распознавания для последующей вставки в текстовое поле.

На первом этапе исходный файл всегда имеет формат mp3, которое не пригодно для распознавания с использованием SpeechRecognition, поскольку данный формат использует сжатие с потерями, поэтому исходный файл необходимо перекодировать в формат wav. Для перекодирования файла используется библиотека с открытым исходным кодом – ffmpeg, которая предоставляет обширные возможности для работы с любыми мультимедиа файлами [9].

На следующем этапе создается объект для распознавания, который содержит перекодированный файл. Распознавание происходит с использованием Google Web Speech API, поскольку данный API обеспечивает более высокую точность распознавания среди прочих [10].

Результатом распознавания является текстовое сообщение, которое сохраняется для последующих действий в браузере.

Описанный алгоритм можно представить в виде следующей блок-схемы (рис. 3.2):

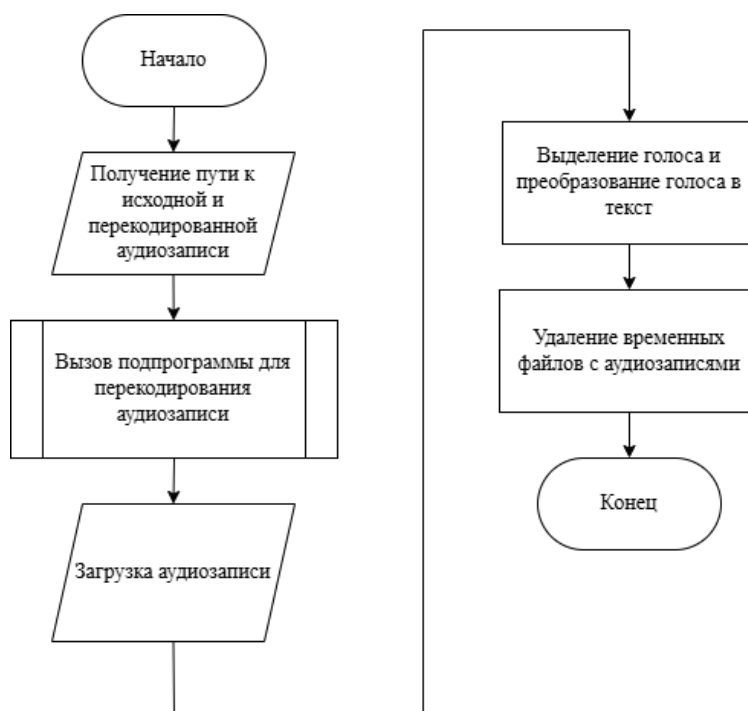


Рис. 3.2 Блок-схема процесса распознавания Audio CAPTCHA.

### САРТСНА в графическом формате

САРТСНА в формате изображений, на сегодняшний день, широко используется для защиты ресурсов от автоматизированных ботов и может быть реализована несколькими способами. Как правило, такие САРТСНА направлены на проверку способности пользователя распознавать и интерпретировать объекты на изображении. Наиболее распространены два варианта реализации (оба варианта реализации проиллюстрированы на рис. 3.3):

1. цельное изображение, содержащее несколько объектов, частично размытых или искаженных, при этом изображение разбито на сетку

3×3 или 4×4. Пользователю предлагается выбрать ячейки, содержащие объекты определенного класса (например, автобусы или светофоры);

2. составное изображение, сформированное из 9 или 12 отдельных фрагментов (изображений), каждый из которых представляет собой независимое изображение – зачастую низкого качества, с наложением артефактов или шумов. Задача пользователя – выбрать те изображения, где присутствует нужный объект.



Рис. 3.3 Изображения САРТСНА с размером сетки 3×3: а) – цельное, б) – составное.

Такие САРТСНА требуют от системы автоматического анализа способности как к глобальному восприятию изображения, так и к локальной интерпретации его фрагментов. Соответственно, модель, предназначенная для решения данной задачи, должна поддерживать:

1. классификацию объектов на уровне отдельных изображений (для САРТСНА, основанных на отдельных картинках в сетке);
2. локализацию и сегментацию объектов с высокой точностью, чтобы корректно определить границы объектов в пределах ячеек, особенно

в случаях, когда объект может частично заходить за границу между ячейками.

Для решения этих задач были рассмотрены следующие современные архитектуры нейронных сетей, которые были рассмотрены ранее:

1. YOLO (You Only Look Once) – однопроводная модель, объединяющая классификацию и регрессию ограничивающих рамок в одной сверточной архитектуре; отличается высокой скоростью и хорошей точностью [**redmon2016yolov2**; **UltralyticsYOLOv8**];
2. Faster R-CNN – двухступенчатая модель, в которой сначала генерируются области предложений, а затем выполняется классификация и уточнение рамок; обладает высокой точностью, но уступает в скорости [**ren2015fasterrcnn**];
3. DETR (DEtection TRansformer) – основана на архитектуре трансформеров, что позволяет эффективно моделировать глобальные взаимосвязи между объектами. Подходит для задач с большим количеством контекстных зависимостей, но требует больше ресурсов для обучения [**carion2020detr**].

Среди этих архитектур было принято решение использовать YOLOv8 по следующим причинам:

1. высокая производительность: YOLOv8 показывает высокую скорость обработки изображений без значительного ущерба для точности, что критично в условиях, когда необходимо обрабатывать CAPTCHA в реальном времени [**bochkovskiy2020yolov4**];
2. гибкость и масштабируемость: модель предоставляет множество предобученных вариантов с различной глубиной и числом параметров (версии n, s, m, l, x), что позволяет использовать как на слабых, так и на производительных устройствах;
3. широкая поддержка и документация: YOLOv8 имеет активное сообщество, подробную документацию и регулярно обновляется, что

значительно упрощает интеграцию и адаптацию модели под пользовательские задачи;

4. поддержка сегментации: в отличие от более ранних версий, YOLOv8 поддерживает не только детекцию, но и сегментацию объектов, что особенно важно для задач, где необходимо точно определить область объекта внутри изображения;
5. дообучение на пользовательских данных: YOLOv8 позволяет эффективно дообучать модель на собственных датасетах, что особенно важно при работе с CAPTCHA-изображениями, содержащими специфические классы объектов и нестандартные искажения.

Кроме того, модель YOLOv8 была успешно протестирована в задачах, близких по структуре к CAPTCHA: детекции дорожных знаков, транспортных средств, пешеходов и других объектов в сложных условиях съемки, что подтверждает ее универсальность и применимость к рассматриваемой задаче.

Таким образом, YOLOv8 является наиболее сбалансированным выбором, обеспечивающим как точную классификацию, так и локализацию объектов в условиях ограниченных ресурсов и с возможностью адаптации под специфику визуальных CAPTCHA.

В качестве основной архитектуры была выбрана модель YOLOv8m-seg, поддерживающая сегментацию объектов. Она представляет собой сбалансированное решение между качеством распознавания, производительностью и требованиями к аппаратному обеспечению. Благодаря своей универсальности, модель подходит как для задач классификации, так и для задач детектирования и сегментации, что особенно важно при работе с CAPTCHA, содержащими зашумленные или плохо различимые объекты.

Преимущества YOLOv8m-seg заключаются в следующем:

1. наличие встроенной поддержки сегментации объектов, что особенно важно при необходимости выделения фрагментов изображений;
2. возможность использования предобученных весов, сокращающих время на обучение и повышающих стартовую точность;

3. высокая скорость инференса по сравнению с другими моделями сегментации (например, Mask R-CNN или DETR);
4. встроенные средства аугментации (изменения яркости, повороты, масштабирование и пр.);
5. удобный интерфейс через библиотеку `ultralitics`, позволяющий быстро запускать обучение, логировать метрики и визуализировать результаты;
6. полная совместимость с аннотациями в формате YOLO, полученными из CVAT.

Перед запуском обучения структура данных была организована в соответствии с требованиями YOLOv8: директории `train` и `val` содержали соответствующие изображения и файлы разметки, а в `.yaml` файле конфигурации были указаны пути к выборкам и список классов.

Обучение проводилось на 35 эпохах при размере изображений  $640 \times 640$  пикселей и размере батча 8. Использование предобученных весов позволило достичь стабильного снижения функции потерь с первых эпох, а встроенные механизмы аугментации способствовали улучшению обобщающей способности модели.

### 3.3. Исследовательский вклад

#### **Тестирование выбранной модели нейронной сети (text captcha)**

Как было установлено в предыдущих разделах, модель последовательного преобразования (Seq2Seq) продемонстрировала наилучшие результаты среди рассмотренных архитектур. Следующим этапом работы являлась оптимизация параметров модели, включая веса и коэффициенты регуляризации, с целью ускорения сходимости, минимизации риска переобучения и повышения точности распознавания целевых последовательностей.

Для проведения экспериментов исходный набор данных, содержащий 100 000 изображений, был случайным образом перемешан и разделен на три подмножества: обучающее, тестовое и валидационное в соотношении 80:10:10. Обучающая выборка использовалась непосредственно для обуче-



ния модели, валидационная – для контроля качества процесса обучения на каждой эпохе, а тестовая – для окончательной оценки модели на данных, с которыми она ранее не сталкивалась. В качестве основных метрик качества модели использовались функция потерь (loss) и точность (accuracy), рассчитываемая для каждого символа последовательности.

В процессе многократного обучения были экспериментально определены оптимальное количество эпох и значения гиперпараметров, обеспечивающие эффективное снижение функции потерь до приемлемых значений. График сходимости функции потерь представлен на рис. 3.4.

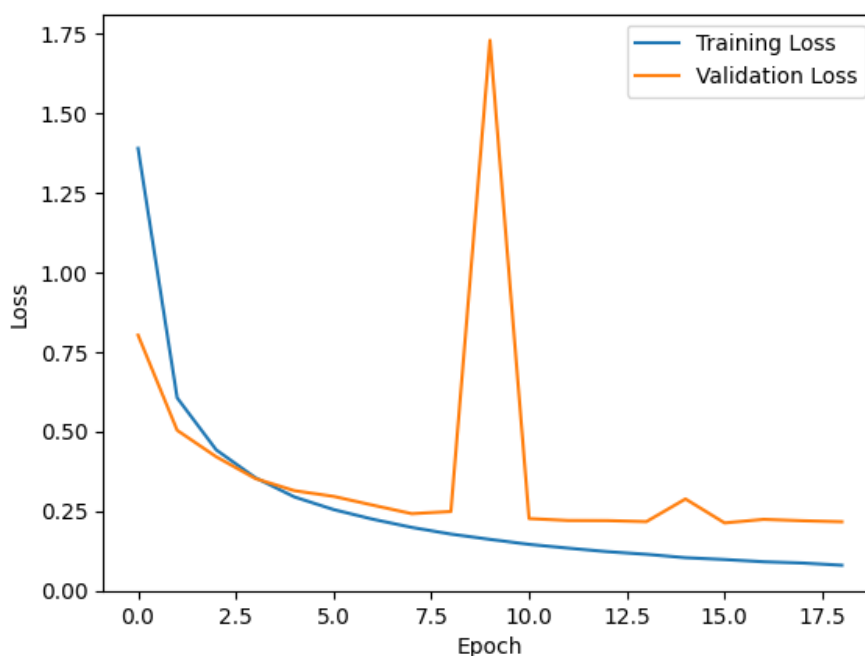


Рис. 3.4 График изменения значений функции потерь в процессе обучения.

Для предотвращения переобучения использовался механизм ранней остановки, согласно которому обучение прекращалось при отсутствии уменьшения значения функции потерь на валидационной выборке в течение трех последовательных эпох. В данном эксперименте обучение завершилось на 18-й эпохе. На графике видно, что функция потерь стабилизировалась после 10 эпохе, поэтому 10 эпоха является балансом между точностью распознавания последовательностей и скоростью обучения модели.

Анализ графика сходимости функции потерь показывает наличие резкого увеличения ее значения на 9-й эпохе, что может быть обусловлено следующими факторами:

1. перемешивание данных перед каждой эпохой могло привести к образованию несбалансированной выборки, содержащей значительное число сложных примеров.
2. динамическое изменение скорости обучения, осуществляемое с помощью механизма регулирования скорости обучения (learning rate scheduler), могло повлиять на изменение функции потерь.

Окончательная точность распознавания отдельных символов составила 0.9263.

После подбора оптимальных значений гиперпараметров модель была сохранена и протестирована на тестовой выборке. Точность распознавания последовательностей различной длины представлена в таблице 3.1.

Таблица 3.1

Точность предсказаний для последовательностей различной длины.

Длина последовательности	Точность распознавания
4 символа	0.9305
5 символов	0.7450
6 символов	0.4575
7 символов	0.1915

Также была построена матрица ошибок, позволяющая проанализировать частоту и характер ошибок модели при классификации различных классов. Данная матрица приведена на рис. 3.5.

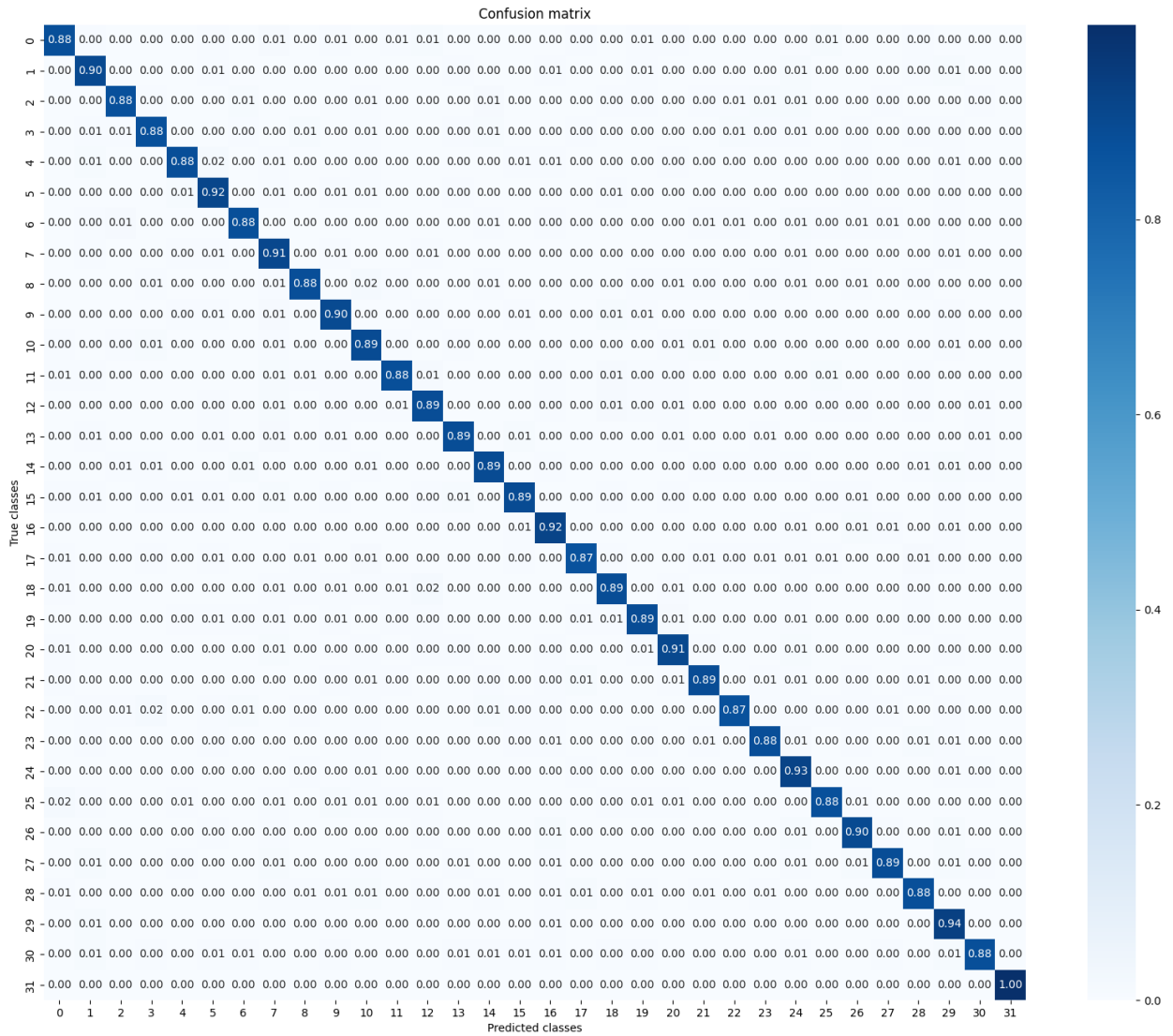


Рис. 3.5 Матрица ошибок для обученной модели.

Анализ полученных результатов показывает, что точность распознавания последовательностей значительной длины остается относительно низкой. Это можно объяснить высокой зависимостью модели Seq2Seq от объема обучающих данных: для эффективного обобщения признаков, извлекаемых из изображений, требуется значительное количество примеров. Следовательно, увеличение размера обучающего набора данных потенциально может способствовать повышению точности модели, однако это также накладывает дополнительные требования к вычислительным ресурсам, необходимым для ее обучения.

### Тестирование модели (image captcha)

Результаты обучения модели на основе YOLO отслеживались по ключевым метрикам (IoU, Precision, Recall, Loss), которые визуализировались автоматически. Примеры графиков с результатами обучения приведены ниже:

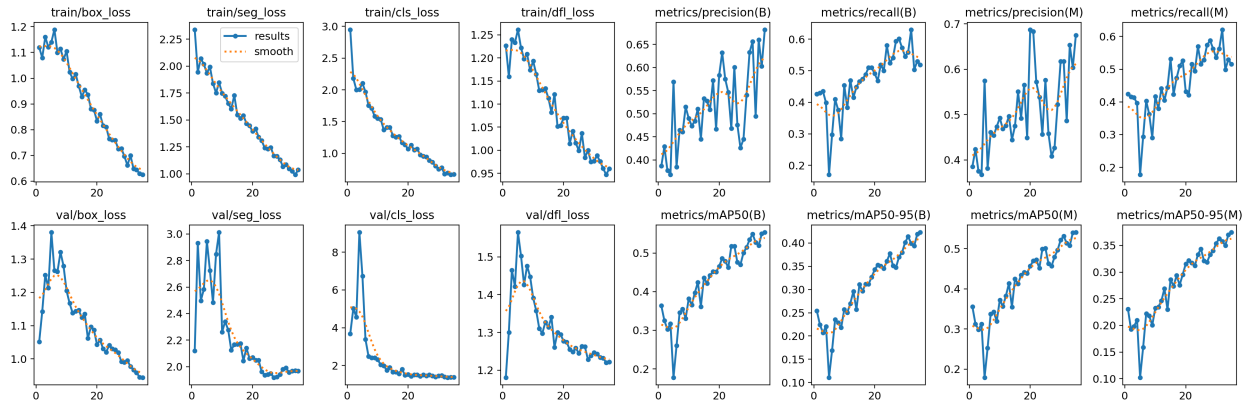


Рис. 3.6 Изменение ключевых метрик в процессе обучения.

Также, была построена нормализованная матрица ошибок для определения точности предсказания необходимых классов на валидационной выборке, которая представлена на рис. 3.7.

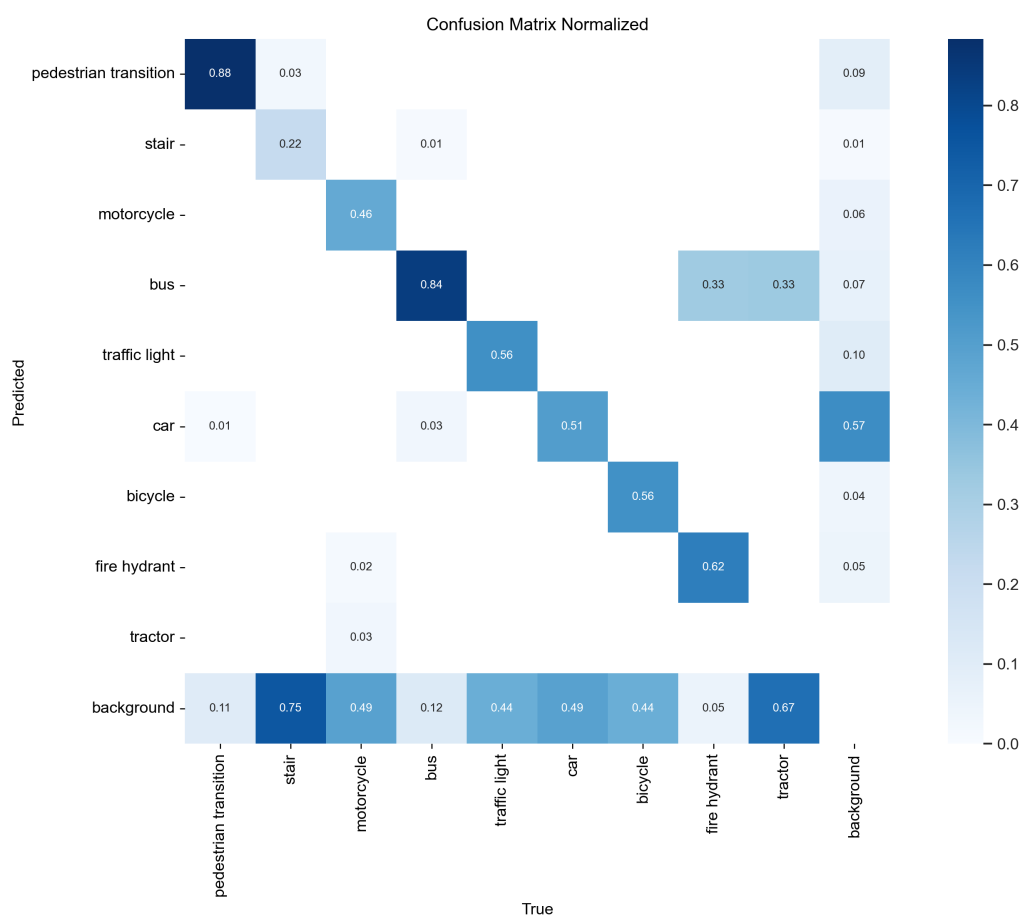


Рис. 3.7 Матрица ошибок для изображений валидационной выборки.

После завершения обучения модель на основе YOLO была протестирована на реальных CAPTCHA, собранных с помощью автоматического парсера, реализованного на базе библиотеки Selenium. Тестирование проводилось в автоматическом режиме, имитируя реальные действия пользователя в браузере, что позволило оценить работоспособность системы в условиях, приближенных к реальной эксплуатации.

Сценарий тестирования предусматривал выполнение следующих шагов:

1. автоматический переход к странице с CAPTCHA и активация чек-бокса «Я не робот»;
2. извлечение изображения CAPTCHA (включая структуру сетки и текст задания);
3. определение целевого объекта из текста задания (например, «выберите все изображения с автобусами»);

4. разбиение изображения САРТСНА на ячейки (в зависимости от размера сетки –  $3 \times 3$  или  $4 \times 4$ );
5. применение обученной модели для сегментации и классификации каждого изображения или фрагмента;
6. определение ячеек, содержащих нужный класс, и программная симуляция кликов по ним;
7. повторная попытка прохождения САРТСНА в случае, если результат оказался некорректным (что также фиксировалось в логах).

Тестирование было организовано в виде цикла, позволяющего автоматически проходить САРТСНА до тех пор, пока не будет достигнут положительный результат. Это позволило зафиксировать частоту ошибок модели и определить случаи, в которых требуются дообучение или оптимизация.

Рабочий процесс тестирования и взаимодействия модели с САРТСНА представлен на блок-схеме ниже.

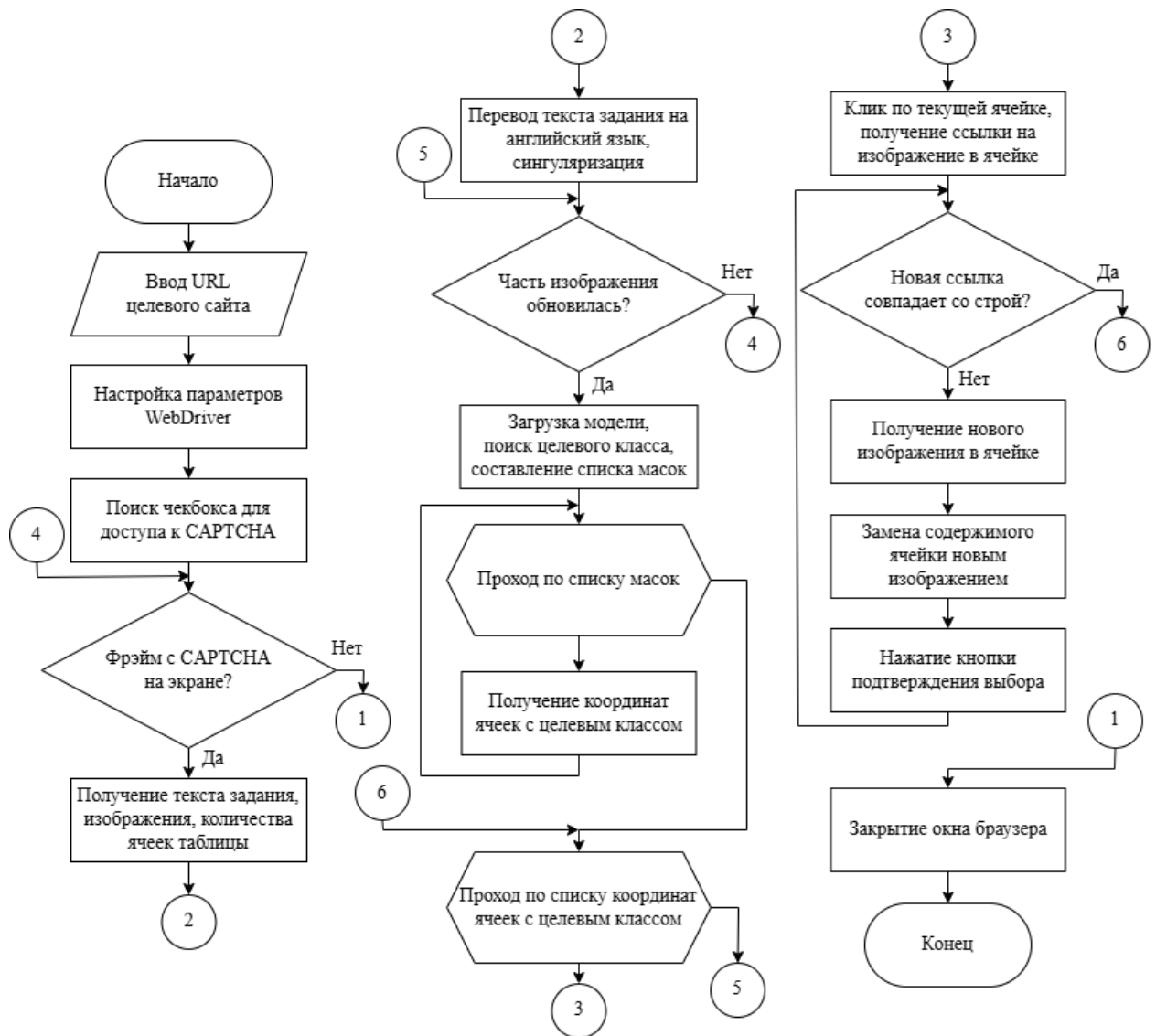


Рис. 3.8 Блок-схема процесса прохождения CAPTCHA.

Полученные данные используются для последующего анализа качества модели и корректировки процесса обучения. Основное внимание при анализе было уделено типам ошибок, сложности распознаваемых объектов и влиянию качества исходного изображения на точность сегментации.

#### 4. ОБСУЖДЕНИЕ, ВЫВОДЫ И ПЕРСПЕКТИВЫ РАЗВИТИЯ

4.1. Анализ надежности и уязвимостей решений

4.2. Ограничения проведенных экспериментов

4.3. Перспективные направления



# ПРИЛОЖЕНИЕ

## Текст программы

### Листинг 1

#### Исходный код расшифровки Audio CAPTCHA

---

```

1  """Файл с классом для решения audiocaptcha"""
2  import speech_recognition as sr
3  import subprocess
4  import logger
5  import os
6
7
8  logger = logger.ConfigLogger(__name__)
9
10 class AudioCaptchaSolver():
11     """Класс решателя audio captcha"""
12
13     def __init__(self):
14         """Конструктор класса"""
15         # Создаем объект распознавателя речи
16         self.recognizer = sr.Recognizer()
17         # Распознанное текстовое сообщение
18         self.text_message = None
19
20
21     def recognition_audio(self, path_to_audio: str) -> str:
22         """
23         Метод распознавания аудиофайла
24         Файлы сохраняются в формате mp3 (обычно содержат шум, кроме мест, где слышен голос)
25         """
26
27         # Преобразование mp3 файла в формат, который подходит для распознавания
28         mp3_file = path_to_audio
29         wav_file = './audio/audiocaptcha.wav'
30
31         if os.name == 'nt':
32             subprocess.run(['C:/ffmpeg/bin/ffmpeg.exe', '-i', mp3_file, wav_file])
33         else:
34             subprocess.run(['ffmpeg', '-i', mp3_file, wav_file])
35
36         try:
37             # Загружаем аудио файл
38             audio_captcha = sr.AudioFile(wav_file)
39
40             # Распознаем речь из аудио файла
41             with audio_captcha as voice:
42                 audio_data = self.recognizer.record(voice)
43                 text_message = self.recognizer.recognize_google(audio_data, language='en-US')
44                 logger.log_info('Распознавание речи завершено успешно!')
45         except Exception as e:

```

```

46     logger.log_warning(f'Распознавание завершилось с ошибкой: {e}')
47
48     if text_message:
49         self.text_message = text_message
50         os.remove(mp3_file)
51         os.remove(wav_file)
52
53     return self.text_message

```

## Листинг 2

### Исходный код автоматизированного решения Audio CAPTCHA

```

1  """Это основной файл проекта, в котором будут вызываться классы и методы для решения всех популярных видов captcha"""
2  from selenium import webdriver
3  from selenium.webdriver.remote.webdriver import WebDriver
4  from selenium.webdriver.common.by import By
5
6  from random import randint
7  import time
8  import requests
9  import os
10
11 from audiocaptcha import AudioCaptchaSolver
12
13
14 class MainWorker():
15     """
16     Основной класс проекта, который управляет вызовом дочерних классов для решения определенных видов captcha
17     На начальном этапе здесь также будет все, что касается получения captcha с веб-страницы
18     """
19
20     def __init__(self, browser: WebDriver):
21         """Конструктор класса"""
22         super().__init__()
23         self.browser = browser
24
25
26     def get_captcha(self, link: str) -> str:
27         """Метод получения captcha со страницы"""
28         # Проходим по ссылке
29         self.browser.get(link)
30         time.sleep(randint(3, 5))
31
32         # Переключаемся на фрейм с чекбоксом captcha
33         self.browser.switch_to.frame(self.browser.find_element(By.XPATH, '//*[@id="g-recaptcha"]/div/div/iframe'))
34         # Кликаем по чекбоксу "Я не робот"
35         self.browser.find_element(By.XPATH, '/html/body/div[2]/div[3]/div[1]/div/div/span').click()
36         time.sleep(randint(3, 5))
37
38         # Переключаемся на обычную веб-страницу
39         self.browser.switch_to.default_content()
40         # Переключаемся на фрейм с картинкой captcha
41         self.browser.switch_to.frame(self.browser.find_element(By.XPATH, '/html/body/div[2]/div[4]/iframe'))
42         # Кликаем на кнопку для перехода к audiocaptcha

```

```

43 self.browser.find_element(By.XPATH, '//*[@id="recaptcha-audio-button"]').click()
44 time.sleep(randint(3, 5))
45
46 # Переключаемся на обычную web-страницу
47 self.browser.switch_to.default_content()
48 # Переключаемся на фрейм с аудиозаписью
49 self.browser.switch_to.frame(self.browser.find_element(By.XPATH, '/html/body/div[2]/div[4]/iframe'))
50 # Находим элемент, содержащий ссылку на аудиозапись
51 audio = self.browser.find_element(By.XPATH, '//*[@id="audio-source"]').get_attribute('src')
52 # Делаем запрос для получения файла
53 response = requests.get(audio)
54 response.raise_for_status()
55
56 # Создаем папку для хранения временных файлов
57 if not os.path.isdir('./audio'):
58     os.mkdir('./audio')
59 path_to_file = './audio/audiocaptcha.mp3'
60 # Сохраняем файл
61 with open(f'{path_to_file}', 'wb') as audioCaptcha:
62     audioCaptcha.write(response.content)
63
64 return path_to_file
65
66
67 def paste_response(self, response_message):
68     '''Метод для вставки результата распознавания'''
69     browser.find_element(By.XPATH, '//*[@id="audio-response"]').send_keys(f'{response_message}')
70     time.sleep(randint(3, 5))
71     browser.find_element(By.XPATH, '//*[@id="recaptcha-verify-button"]').click()
72
73
74 if __name__ == '__main__':
75     '''Запуск программы'''
76     list_of_links = [
77         'https://rucaptcha.com/demo/recaptcha-v2',
78         'https://lessons.zennolab.com/captchas/recaptcha/v2_simple.php?level=low',
79         'https://lessons.zennolab.com/captchas/recaptcha/v2_simple.php?level=middle',
80         'https://lessons.zennolab.com/captchas/recaptcha/v2_simple.php?level=high',
81         'https://lessons.zennolab.com/captchas/recaptcha/v2_nosubmit.php?level=low',
82         'https://lessons.zennolab.com/captchas/recaptcha/v2_nosubmit.php?level=middle',
83         'https://lessons.zennolab.com/captchas/recaptcha/v2_nosubmit.php?level=high',
84         'https://lessons.zennolab.com/ru/advanced'
85     ]
86
87     for link in list_of_links:
88         # Настройка user agent
89         USER_AGENT = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.0.0
90             ↪ Safari/537.36"
91
92         select_browser = randint(1, 10)
93
94         # Выбор браузера и опций характерных для него
95         if select_browser < 5:
96             options = webdriver.ChromeOptions()
97         else:

```

```

97     options = webdriver.EdgeOptions()
98
99     options.add_experimental_option("excludeSwitches", ["enable-automation"])
100     options.add_experimental_option('useAutomationExtension', False)
101     options.add_argument(f'user-agent={USER_AGENT}')
102     options.add_argument("--disable-blink-features=AutomationControlled")
103
104     # Передача параметров
105     if select_browser < 5:
106         browser = webdriver.Chrome(options=options)
107     else:
108         browser = webdriver.Edge(options=options)
109     browser.implicitly_wait(30)
110
111     # Создаем аудиофайл по указанному пути с captcha
112     solver = MainWorker(browser)
113     path_to_audio = solver.get_captcha(link)
114
115     # Запускаем распознавание
116     captcha_solver = AudioCaptchaSolver()
117     response = captcha_solver.recognition_audio(path_to_audio)
118
119     # Вставляем результат распознавания в поле ввода
120     solver.paste_response(response)
121     time.sleep(randint(10, 15))

```

### Листинг 3

## Исходный код генератора синтетических CAPTCHA

```

1  from captcha.image import ImageCaptcha
2
3  from random import randint, shuffle
4  import numpy as np
5  import os
6
7  from textcaptcha.preprocessing_image import preprocessing_image
8
9
10 def generate_image(path_to_file: str, alphabet: list, number_of_start: int, number_of_captcha: int, size_of_image: tuple) -> list:
11     # Генерация текстовых captcha
12     text = ImageCaptcha(size_of_image[0], size_of_image[1], ['./fonts/arial.ttf', './fonts/comic.ttf', './fonts/cour.ttf', './fonts/georgia.ttf'])
13     # Структура возвращаемого списка: [filename, label, (width, height)]
14     filenames = []
15     for _ in range(number_of_start, number_of_captcha):
16         captcha_text = [alphabet[randint(0, len(alphabet) - 1)] for _ in range(randint(4, 7))]
17         shuffle(captcha_text)
18         text.write("".join(captcha_text), f'{path_to_file}/{"".join(captcha_text)}.png')
19         filenames.append(
20             [f'{path_to_file}/{"".join(captcha_text)}.png',
21              "".join(captcha_text)]
22         )
23
24     return filenames
25

```

```

26
27 if __name__ == '__main__':
28     # Алфавит допустимых символов
29     alphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ023456789'
30     # Создание директории для хранения полноценных синтетических текстовых captcha
31     path_to_dataset = './datasets/captcha'
32     if not os.path.isdir(path_to_dataset):
33         os.mkdir(path_to_dataset)
34     # Создаем датасет из нужного полноценных синтетических captcha длиной от 4 до 7 символов размером 250x60
35     filenames = generate_image(path_to_dataset, list(alphabet), 0, 100000, (250, 60))
36
37     # Предобработка изображений
38     preprocessing_image(filenames)
39
40     # Для отладки без создания датасета с нуля
41     numpy_data = np.array(filenames, dtype=object)
42     np.save('data.npy', numpy_data)

```

---

## Листинг 4

## Исходный код для предобработки изображений датасета

```

1 import cv2
2 import numpy as np
3
4
5 def preprocessing_image(list_filenames: list):
6     """Функция для предобработки изображений или изображения для предсказания"""
7     # Предобработка изображений с CAPTCHA
8     for file in list_filenames:
9         # Открытие изображения в градациях серого
10         gray_image = cv2.imread(file[0], 0)
11         # Приведение всех изображений к одному размеру ширина x высота
12         resized_image = cv2.resize(gray_image, (250, 60))
13
14         # Морфологический фильтр (дилатация) для сужения символов и более четкого отделения их друг от друга
15         morph_kernel = np.ones((3, 3))
16         dilatation_image = cv2.dilate(resized_image, kernel=morph_kernel, iterations=1)
17
18         # Применяем пороговую обработку, чтобы получить только черные и белые пиксели
19         _, thresholder = cv2.threshold(
20             dilatation_image,
21             0,
22             255,
23             cv2.THRESH_BINARY + cv2.THRESH_OTSU
24         )
25
26         cv2.imwrite(file[0], thresholder)

```

---

## Листинг 5

## Исходный код для создания датасета в формате тензоров

```

1 import pandas as pd

```

```

2 import tensorflow as tf
3 from keras._tf_keras.keras.preprocessing.sequence import pad_sequences
4 from sklearn.model_selection import train_test_split
5
6
7 def parse_data(image_path: list, encoder_labels: list, decoder_labels: list) -> tuple[tf.Tensor, list]:
8     """Функция для склеивания изображений и лейблов для датасета"""
9     image = tf.io.read_file(image_path)
10    image = tf.image.decode_png(image, channels=1)
11    image = tf.cast(image, tf.float32) / 255.0
12
13    return (image, encoder_labels), decoder_labels
14
15
16 def create_dataset(images: list, encoder_labels: list, decoder_labels: list, shuffle = True, batch_size = 32) -> tf.data.Dataset:
17     """Функция для создания датасета, понятного для TensorFlow"""
18    dataset = tf.data.Dataset.from_tensor_slices((images, encoder_labels, decoder_labels))
19    dataset = dataset.map(lambda x, y, w: parse_data(x, y, w))
20    if shuffle == True:
21        dataset = dataset.shuffle(len(images)).batch(batch_size)
22    else:
23        dataset = dataset.batch(batch_size)
24
25    return dataset
26
27
28 def create_dataframe(images: list) -> pd.DataFrame:
29     """Функция для создания датафреймов на основе списков"""
30
31    # Создание файла с лейблами о содержимом изображений с CAPTCHA
32    filenames = [objects[0] for objects in images]
33    list_labels = [objects[1] for objects in images]
34
35    # Создание DataFrame для сохранения соответствия между путями, лейблами и размерами для каждого элемента датасета
36    data = {
37        'filename': filenames,
38        'label': list_labels,
39    }
40
41    return pd.DataFrame(data)
42
43
44 def preparing_dataset(dataframe: pd.DataFrame, alphabet: str, shuffle = True) -> tuple[
45     tuple[tf.data.Dataset, tf.data.Dataset, tf.data.Dataset, list],
46     tuple[tf.data.Dataset, tf.data.Dataset, tf.data.Dataset, list]
47 ]:
48     """Подготовка датасета"""
49
50    # Сохранение отдельных составляющих DataFrame
51    X_captcha, y_captcha = dataframe['filename'].tolist(), dataframe['label'].tolist()
52
53    dict_alphabet = {alphabet[i]:i for i in range(len(alphabet))}
54    start_token = len(alphabet) # Индекс токена <start>
55    end_token = len(alphabet) + 1 # Индекс токена <end>
56

```

```

57  # Кодируем лейблы с добавлением токена <start> для кодера
58  encoder_labels = [[start_token] + [dict_alphabet[char] for char in label] for label in y_captcha]
59
60  # Кодируем лейблы с добавлением токена <end> для декодера
61  decoder_labels = [[dict_alphabet[char] for char in label] + [end_token] for label in y_captcha]
62
63  # Преобразование меток в тензоры
64  encoder_tensors = pad_sequences(encoder_labels, maxlen=8, padding='post')
65  decoder_tensors = pad_sequences(decoder_labels, maxlen=8, padding='post')
66
67  # Создание датасета
68  dataset = create_dataset(X_captcha, encoder_tensors, decoder_tensors, shuffle)
69
70  return dataset
71
72
73  def create_dataset_for_captcha(filenamees: list, alphabet: str) -> tuple[
74      tuple[tf.data.Dataset, tf.data.Dataset, tf.data.Dataset, list],
75      tuple[tf.data.Dataset, tf.data.Dataset, tf.data.Dataset, list]
76  ]:
77      '''Функция для создания датасета на основе алфавита и имен файлов'''
78
79      # Создание датафрейма для удобства последующей обработки
80      captcha_dataframe = create_dataframe(filenamees)
81
82      # Разделение датасета на обучающую и тестовую выборки
83      train_captcha_df, test_captcha_df = train_test_split(captcha_dataframe, test_size=0.2, random_state=42)
84      # Разделение тестовой части датасета на валидационную и тестовую выборки
85      val_captcha_df, test_captcha_df = train_test_split(test_captcha_df, test_size=0.5, random_state=42)
86
87      train_dataset = preparing_dataset(train_captcha_df, alphabet)
88      val_dataset = preparing_dataset(val_captcha_df, alphabet)
89      test_dataset = preparing_dataset(test_captcha_df, alphabet, False)
90
91  return train_dataset, val_dataset, test_dataset

```

## Листинг 6

### Исходный код CRNN модели

```

1  import tensorflow as tf
2  import numpy as np
3  from keras_tf.keras.layers import Input, Conv2D, MaxPooling2D, Reshape, Dense, Dropout, Bidirectional, GRU,
   ↪ BatchNormalization
4  from keras_tf.keras.regularizers import l2
5  from keras_tf.keras.models import Model
6  from keras_tf.keras import backend as K
7  from keras_tf.keras.backend import ctc_decode
8  from keras_tf.keras.optimizers import Adam
9  from keras_tf.keras.callbacks import EarlyStopping, ReduceLROnPlateau
10 from keras_tf.keras.saving import register_keras_serializable
11
12
13 def decode_predictions(preds, max_length, alphabet):
14     # Используем CTC-декодирование для предсказаний

```

```

15     decoded_preds, _ = ctc_decode(preds, input_length=np.ones(preds.shape[0]) * preds.shape[1])
16     texts = []
17     for seq in decoded_preds[0]:
18         text = ''.join([alphabet[i] for i in seq.numpy() if i != -1]) # Искключаем 'blank' символы
19         texts.append(text)
20     return texts
21
22
23 def decode_batch_predictions(pred):
24     # CTC decode
25     decoded, _ = ctc_decode(pred, input_length=np.ones(pred.shape[0]) * pred.shape[1],
26                             greedy=True)
27     decoded_texts = []
28
29     # Преобразование в текст
30     for seq in decoded[0]:
31         text = ''.join([chr(x) for x in seq if x != -1]) # Пропускаем -1 (пустые символы CTC)
32         decoded_texts.append(text)
33     return decoded_texts
34
35
36 # Функция CTC Loss
37 # Функция для декодирования предсказаний модели
38 @register_keras_serializable(package='Custom', name='ctc_loss')
39 def ctc_loss(y_true, y_pred):
40     # Формируем входные данные для CTC
41     input_length = tf.ones(shape=(tf.shape(y_pred)[0], 1)) * tf.cast(tf.shape(y_pred)[1], tf.float32)
42     label_length = tf.ones(shape=(tf.shape(y_true)[0], 1)) * 7
43     return tf.reduce_mean(K.ctc_batch_cost(y_true, y_pred, input_length, label_length))
44
45
46 # Модель
47 def build_model(num_of_classes):
48     '''Создание модели'''
49     # Входной слой
50     input_layer = Input((60, 250, 1))
51
52     # Первый сверточный блок
53     x = Conv2D(32, (3, 3), activation='relu', padding='same', kernel_regularizer=l2(0.003))(input_layer)
54     x = BatchNormalization()(x)
55     x = Conv2D(32, (3, 3), activation='relu', kernel_regularizer=l2(0.003))(x)
56     x = MaxPooling2D((1, 2))(x)
57     x = Dropout(0.25)(x) # Dropout после каждого блока
58
59     # Второй сверточный блок
60     x = Conv2D(64, (3, 3), activation='relu', padding='same', kernel_regularizer=l2(0.003))(x)
61     x = BatchNormalization()(x)
62     x = Conv2D(64, (3, 3), activation='relu', kernel_regularizer=l2(0.003))(x)
63     x = MaxPooling2D((1, 2))(x)
64     x = Dropout(0.3)(x)
65
66     # Третий сверточный блок
67     x = Conv2D(128, (3, 3), activation='relu', padding='same', kernel_regularizer=l2(0.003))(x)
68     x = BatchNormalization()(x)
69     x = Conv2D(128, (3, 3), activation='relu', kernel_regularizer=l2(0.003))(x)

```



```

70 x = MaxPooling2D((1, 2))(x)
71 x = Dropout(0.4)(x)
72
73 # Изменяем размерность тензора
74 x = Reshape((-1, x.shape[-1] * x.shape[-2]))(x)
75
76 # Первый рекуррентный блок
77 x = Bidirectional(GRU(128, return_sequences=True))(x)
78 x = BatchNormalization()(x)
79 x = Dropout(0.6)(x)
80
81 # Второй рекуррентный блок
82 x = Bidirectional(GRU(128, return_sequences=True))(x)
83 x = BatchNormalization()(x)
84 x = Dropout(0.6)(x)
85
86 # Третий рекуррентный блок
87 x = Bidirectional(GRU(128, return_sequences=True))(x)
88 x = BatchNormalization()(x)
89 x = Dropout(0.6)(x)
90
91 # Выходной слой
92 outputs = Dense(num_of_classes + 1, activation='softmax')(x)
93
94 # Создание модели
95 model = Model(inputs=input_layer, outputs=outputs)
96
97 return model
98
99
100 def fit_crnn(num_of_classes, train, val):
101     # Компиляция модели
102     model = build_model(num_of_classes)
103     optimizer = Adam(learning_rate=0.001, weight_decay=1e-6)
104     model.compile(
105         loss=ctc_loss,
106         optimizer=optimizer
107     )
108
109     # Вывод структуры модели
110     model.summary()
111
112     lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, min_lr=1e-6)
113     early_stop = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
114
115     # Обучение модели
116     history = model.fit(
117         train,
118         validation_data=val,
119         epochs=15,
120         callbacks=[early_stop, lr_scheduler]
121     )
122
123     model.save('crnn_model.keras')
124

```

125 **return** model, history

## Листинг 7

### Исходный код Seq-to-Seq модели

```

1 import tensorflow as tf
2 from keras._tf_keras.keras import layers, Model
3 from keras._tf_keras.keras.callbacks import EarlyStopping, ReduceLROnPlateau
4
5 from create_dataset import create_dataset_for_captcha
6
7
8 # Обновлённый кодировщик
9 def build_encoder():
10     encoder_inputs = layers.Input(shape=(60, 250, 1), name="encoder_inputs")
11
12     # Первый сверточный блок
13     x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(encoder_inputs)
14     x = layers.BatchNormalization()(x)
15     x = layers.Conv2D(32, (3, 3), activation='relu')(x)
16     x = layers.MaxPooling2D((2, 2))(x)
17
18     # Второй сверточный блок
19     x = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(x)
20     x = layers.BatchNormalization()(x)
21     x = layers.Conv2D(64, (3, 3), activation='relu')(x)
22     x = layers.MaxPooling2D((2, 2))(x)
23
24     # Третий сверточный блок
25     x = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(x)
26     x = layers.BatchNormalization()(x)
27     x = layers.Conv2D(128, (3, 3), activation='relu')(x)
28     x = layers.MaxPooling2D((2, 2))(x)
29
30     # Четвертый сверточный блок
31     x = layers.Conv2D(256, (3, 3), activation='relu', padding='same')(x)
32     x = layers.BatchNormalization()(x)
33     x = layers.Conv2D(256, (3, 3), activation='relu')(x)
34     x = layers.MaxPooling2D((2, 2))(x)
35
36     x = layers.GlobalAveragePooling2D()(x)
37     x = layers.Dense(256, activation="relu")(x)
38     x = layers.BatchNormalization()(x)
39     x = layers.Reshape((1, 256))(x) # Добавляем временное измерение
40
41     # RNN слой
42     encoder_output, encoder_state = layers.GRU(256, return_sequences=True, return_state=True)(x)
43
44     return Model(encoder_inputs, [encoder_output, encoder_state], name="encoder")
45
46
47 # Декодировщик с Attention
48 def build_decoder(alphabet_size):
49     decoder_inputs = layers.Input(shape=(None,), name="decoder_inputs")

```

```

50 encoder_state_input = layers.Input(shape=(256,), name="encoder_state_input")
51
52 x = layers.Embedding(alphabet_size, 128)(decoder_inputs)
53 rnn_output, decoder_state = layers.GRU(256, return_sequences=True, return_state=True)(x, initial_state=encoder_state_input)
54
55 # Attention
56 attention_output = layers.AdditiveAttention()([rnn_output, encoder_state_input])
57 x = layers.Concatenate()([rnn_output, attention_output])
58 decoder_outputs = layers.Dense(alphabet_size, activation="softmax")(x)
59
60 return Model([decoder_inputs, encoder_state_input], [decoder_outputs, decoder_state], name="decoder")
61
62
63 def fit_seq_to_seq(number_of_classes: int, train_dataset: tf.data.Dataset, val_dataset: tf.data.Dataset) -> tuple[Model, dict]:
64     # Построение полной модели
65     encoder = build_encoder()
66     decoder = build_decoder(number_of_classes + 2)
67
68     # Полная модель
69     encoder_inputs = encoder.input
70     decoder_inputs = layers.Input(shape=(None,), name="decoder_inputs")
71
72     _, encoder_state = encoder(encoder_inputs)
73     decoder_output, _ = decoder([decoder_inputs, encoder_state])
74
75     seq2seq_model = Model([encoder_inputs, decoder_inputs], decoder_output, name="seq2seq_model")
76
77     # Компиляция модели
78     seq2seq_model.compile(
79         loss="sparse_categorical_crossentropy",
80         optimizer="adam",
81         metrics=["accuracy"]
82     )
83
84     seq2seq_model.summary()
85
86     lr_sheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, min_lr=1e-6)
87     early_stop = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
88
89     # Обучение модели
90     history = seq2seq_model.fit(
91         train_dataset,
92         validation_data=val_dataset,
93         epochs=20,
94         callbacks=[early_stop, lr_sheduler]
95     )
96
97     seq2seq_model.save('seq_to_seq_model.keras')
98
99     return seq2seq_model, history

```

---

```

1 import numpy as np
2 import tensorflow as tf
3 from keras_tf.keras.models import load_model
4
5
6 if __name__ == '__main__':
7     import matplotlib.pyplot as plt
8     import seaborn as sbn
9
10    # Алфавит допустимых символов
11    alphabet = ' ABCDEFGHJKLMNPQRSTWXYZ023456789'
12
13    list_filenames = np.load('data.npy', allow_pickle=True)
14    # Создание единого датасета
15    captcha_dataset = create_dataset_for_captcha(list_filenames, alphabet)
16    if False:
17        # Обучение модели
18        model_captcha, history_captcha = fit_seq_to_seq(len(alphabet), captcha_dataset[0], captcha_dataset[1])
19        # Построение графика изменения val_loss и loss
20        plt.plot(history_captcha.history['loss'], label='Training Loss')
21        plt.plot(history_captcha.history['val_loss'], label='Validation Loss')
22        plt.xlabel('Epoch')
23        plt.ylabel('Loss')
24        plt.legend()
25        # Сохраняем график для отчета
26        plt.savefig('Model_loss.png')
27
28    # Загружаем предобученную модель и получаем предсказания для тестовой выборки
29    model = load_model('seq_to_seq_model.keras')
30    predictions = model.predict(captcha_dataset[2])
31
32    # Переводим предсказания из представления вероятностей в классы
33    pred_classes = np.argmax(predictions, axis=-1)
34    captcha_labels = [label.numpy() for _, label in captcha_dataset[2].unbatch()]
35    captcha_labels = np.array(captcha_labels)
36
37    # Убираем padding
38    def remove_padding(sequences, padding_value=0):
39        return [seq[seq != padding_value] for seq in sequences]
40
41    # Убираем padding из предсказаний и меток
42    pred_classes_no_padding = remove_padding(pred_classes, padding_value=0)
43    true_labels_no_padding = remove_padding(np.array(captcha_labels), padding_value=0)
44
45    # Проверяем размеры списков после удаления padding
46    print(f'Количество предсказаний: {len(pred_classes_no_padding)}')
47    print(f'Количество меток: {len(true_labels_no_padding)}')
48
49    # Проверяем совпадение предсказаний и истинных меток посимвольно
50    sequence_accuracy = np.mean(
51        [np.array_equal(pred, true) for pred, true in zip(pred_classes, captcha_labels)]
52    )
53    print(f'Точность последовательностей (без padding): {sequence_accuracy:.4f}')
54
55    # Расчет точности символов (character-level accuracy)

```

```

56 total_characters = np.prod(captcha_labels.shape)
57 correct_characters = np.sum(pred_classes == captcha_labels)
58 character_accuracy = correct_characters / total_characters
59 print(f"Точность символов: {character_accuracy:.4f}")
60
61 from sklearn.metrics import confusion_matrix
62 # Построение матрицы ошибок для анализа
63 true_symb, pred_symb = [], []
64
65 for true_seq, pred_seq in zip(true_labels_no_padding, pred_classes_no_padding):
66     true_symb.extend(true_seq)
67     pred_symb.extend(pred_seq)
68 cm = confusion_matrix(true_symb, pred_symb)
69
70 plt.figure(figsize=(10, 7))
71 sbn.heatmap(cm, annot=True, fmt='g', cmap='Blues')
72 plt.xlabel('Predicted classes')
73 plt.ylabel('True classes')
74 plt.title('Confusion matrix')
75 # plt.show()
76 plt.savefig('Confusion_matrix.png')
77
78 from collections import defaultdict
79
80 sequence_accuracy_by_length = defaultdict(list)
81 for pred, true in zip(pred_classes_no_padding, true_labels_no_padding):
82     seq_len = len(true)
83     is_correct = np.array_equal(pred, true)
84     sequence_accuracy_by_length[seq_len].append(is_correct)
85
86 # Считаем точность для каждой длины
87 for length, results in sequence_accuracy_by_length.items():
88     acc = np.mean(results)
89     print(f"Длина {length}: Точность {acc:.4f}")

```

## Листинг 9

### Исходный код получения CAPTCHA с целевого сайта

```

1 # Подключение библиотек для работы с браузером
2 from selenium import webdriver
3 from selenium.webdriver.remote.webdriver import WebDriver
4 from selenium.webdriver.common.by import By
5
6 # Подключение библиотек для работы с текстом задания captcha
7 from deep_translator import GoogleTranslator
8 import inflect
9
10 # Библиотека для парсинга HTML
11 from bs4 import BeautifulSoup
12
13 from random import randint
14 import time
15 import requests
16 import os

```

```

17 import csv
18
19
20 class GetCaptcha():
21     """
22     Основной класс проекта, который управляет вызовом дочерних классов для решения определенных видов captcha
23     На начальном этапе здесь также будет все, что касается получения captcha с веб-страницы
24     """
25
26     def __init__(self, browser: WebDriver):
27         """Конструктор класса"""
28         super().__init__()
29         self.browser = browser
30
31
32     def get_captcha(self, link: str, cnt: int) -> tuple[str, str, str]:
33         """Метод получения captcha со страницы"""
34         # Проходим по ссылке
35         self.browser.get(link)
36         time.sleep(randint(3, 5))
37
38         # Переключаемся на фрейм с чекбоксом captcha
39         self.browser.switch_to.frame(self.browser.find_element(
40             By.XPATH,
41             '//*[@id="g-recaptcha"]/div/div/iframe'
42         ))
43         # Кликаем по чекбоксу "Я не робот"
44         self.browser.find_element(
45             By.XPATH,
46             '/html/body/div[2]/div[3]/div[1]/div/div/span'
47         ).click()
48         time.sleep(randint(3, 5))
49
50         # Переключаемся на обычную web-страницу
51         self.browser.switch_to.default_content()
52         # Переключаемся на фрейм с картинкой captcha
53         self.browser.switch_to.frame(self.browser.find_element(
54             By.XPATH,
55             '/html/body/div[2]/div[4]/iframe'
56         ))
57         # Находим элемент, содержащий ссылку на исходное изображение
58         image = self.browser.find_element(
59             By.XPATH,
60             '//*[@id="rc-imageselect-target"]/table/tbody'+
61             '/tr[1]/td[1]/div/div[1]/img'
62         ).get_attribute('src')
63         # Делаем запрос для получения файла
64         response = requests.get(image)
65         response.raise_for_status()
66
67         # Получаем название объекта, который надо найти
68         object_name = self.browser.find_element(
69             By.XPATH,
70             '//*[@id="rc-imageselect"]/div[2]/div[1]/div[1]+'
71             '/div/strong'

```

```

72     ).text
73
74     # Получаем таблицу с кусочками изображения
75     table = self.browser.find_element(
76         By.XPATH,
77         '//*[@id="rc-imageselect-target"]/table'
78     ).get_attribute('outerHTML')
79
80     # Создаем папку для хранения временных файлов
81     if not os.path.isdir('./datasets/imagecaptcha_dataset'):
82         os.mkdir('./datasets/imagecaptcha_dataset')
83     path_to_file = f'./datasets/imagecaptcha_dataset/{cnt}.jpg'
84     # Сохраняем файл
85     with open(f'{path_to_file}', 'wb') as imageCaptcha:
86         imageCaptcha.write(response.content)
87
88     return object_name, path_to_file, table
89
90
91     def get_number_of_cells(self, table:str) -> tuple[int, int]:
92         """Метод для получения количества ячеек таблицы для последующего разбиения изображения на части"""
93         # Парсинг HTML
94         soup = BeautifulSoup(table, 'lxml')
95
96         # Получаем количество строк
97         number_of_rows = len(soup.find_all('tr'))
98
99         # Получаем количество столбцов
100        number_of_columns = len(soup.find('tr').find_all(['td', 'th']))
101
102        return number_of_rows, number_of_columns
103
104
105    if __name__ == "__main__":
106        # Целевой сайт
107        target_link = 'https://rucaptcha.com/demo/recaptcha-v2'
108        for cnt in range(463, 638):
109            # Настройку user agent
110            USER_AGENT = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.0.0
111            ↪ Safari/537.36"
112
113            options = webdriver.ChromeOptions()
114
115            options.add_experimental_option("excludeSwitches", [{"enable-automation"}])
116            options.add_experimental_option('useAutomationExtension', False)
117            options.add_argument(f'user-agent={USER_AGENT}')
118            options.add_argument(
119                "--disable-blink-features=AutomationControlled"
120            )
121
122            # Передача параметров
123            browser = webdriver.Chrome(options=options)
124            browser.implicitly_wait(30)
125
126            captcha = GetCaptcha(browser)
127            # Получение captcha и объекта для поиска

```

```

126 task_object, image, table = captcha.get_captcha(target_link, cnt)
127
128 # Перевод названия объекта на английский и сохранение его в единственном числе
129 task_object = GoogleTranslator(source='auto', target='en').translate(task_object)
130 singular = inflect.engine()
131 if len(task_object) > 3:
132     # Исключаем ошибки с множественным числом для слов, которые не могут быть во множественном числе из-за малого
133     ↔ количества символов
134     task_object = singular.singular_noun(task_object)
135     if task_object.lower() == 'hydrant':
136         task_object = 'fire hydrant'
137
138 # Получаем количество ячеек
139 rows, columns = captcha.get_number_of_cells(table)
140
141 # Запись полученных параметров в csv-файл
142 with open('images_for_captcha.csv', 'a') as datasetFile:
143     csv_rows = csv.writer(datasetFile, quoting=csv.QUOTE_NONE)
144     csv_rows.writerow([task_object, image, rows, columns])

```

## Листинг 10

### Исходный код дообучения модели на датасете

```

1 from ultralytics import YOLO
2
3 # Загрузка модели
4 model = YOLO("yolov8m-seg.pt") # Загрузка предобученной лёгкой модели
5 # Дообучение модель
6 model.train(
7     data="datasets/image_dataset/image_captcha.yaml", # Путь к файлу конфигурации
8     epochs=35,
9     imgsz=640,
10    batch=8,
11    workers=4,
12    device="cpu",
13    name="captcha_seg" # Название директории для сохранения результатов обучения
14 )

```

## Листинг 11

### Исходный код автоматизированного решения CAPTCHA

```

1 # Подключение библиотек для работы с браузером
2 from selenium import webdriver
3 from selenium.webdriver.remote.webdriver import WebDriver
4 from selenium.webdriver.common.by import By
5 from selenium.common.exceptions import ElementClickInterceptedException
6
7 # Подключение библиотек для работы с текстом задания captcha
8 from deep_translator import GoogleTranslator
9 import inflect
10
11 # Библиотека для парсинга HTML

```



```

12 from bs4 import BeautifulSoup
13
14 # Библиотека для работы с изображениями
15 from ultralytics import YOLO
16 import cv2
17 import numpy as np
18
19 from random import randint
20 import time
21 import requests
22
23
24 class SolveCaptcha():
25     """
26     Основной класс проекта, который управляет вызовом дочерних классов для решения определенных видов captcha
27     На начальном этапе здесь также будет все, что касается получения captcha с веб-страницы
28     """
29
30     def __init__(self, browser: WebDriver):
31         """Конструктор класса"""
32         super().__init__()
33         self.browser = browser
34
35
36     def find_captcha(self, link: str):
37         # Проходим по ссылке
38         self.browser.get(link)
39         time.sleep(randint(3, 5))
40
41         # Переключаемся на фрейм с чекбоксом captcha
42         self.browser.switch_to.frame(self.browser.find_element(
43             By.XPATH,
44             '//*[@id="g-recaptcha"]/div/div/iframe'
45         ))
46         # Кликаем по чекбоксу "Я не робот"
47         self.browser.find_element(By.XPATH, 'html/body/div[2]/div[3]/div[1]/div/div/span').click()
48         time.sleep(randint(3, 5))
49
50         # Переключаемся на обычную web-страницу
51         self.browser.switch_to.default_content()
52         # Переключаемся на фрейм с картинкой captcha
53         self.browser.switch_to.frame(self.browser.find_element(
54             By.XPATH,
55             'html/body/div[2]/div[4]/iframe'
56         ))
57
58
59     def get_captcha(self) -> tuple[str, str, str, np.ndarray]:
60         """Метод получения captcha со страницы"""
61         # Находим элемент, содержащий ссылку на исходное изображение
62         src_image = self.browser.find_element(
63             By.XPATH,
64             '//*[@id="rc-imageselect-target"]/table/tbody/tr[1]/td[1]/div/div[1]/img'
65         ).get_attribute('src')

```

```

67     # Делаем запрос для получения файла
68     response = requests.get(src_image)
69     response.raise_for_status()
70
71     # Получаем название объекта, который надо найти
72     object_name = self.browser.find_element(
73         By.XPATH,
74         '//*[@id="rc-imageselect"]/div[2]/div[1]/div[1]/'+
75         'div/strong'
76     ).text
77
78     # Получаем таблицу с кусочками изображения
79     table = self.browser.find_element(
80         By.XPATH,
81         '//*[@id="rc-imageselect-target"]/table'
82     ).get_attribute('outerHTML')
83
84     # Преобразование байтовой последовательности в изображение
85     image = cv2.imdecode(np.frombuffer(response.content, np.uint8), cv2.IMREAD_COLOR)
86
87     return object_name, table, src_image, image
88
89
90     def get_properties_for_recognition(self, task_object: str, table: str) -> tuple[str, int, int]:
91         """Метод для получения необходимых параметров для распознавания на картинке"""
92         # Перевод названия объекта на английский и сохранение его в единственном числе
93         task_object = GoogleTranslator(source='auto', target='en').translate(task_object)
94         singular = inflect.engine()
95         if len(task_object) > 3:
96             # Исключаем ошибки с множественным числом для слов, которые не могут быть во множественном числе из-за малого
97             ↪ количества символов
98             task_object = singular.singular_noun(task_object)
99             if task_object.lower() == 'hydrant':
100                 task_object = 'fire hydrant'
101
102         # Парсинг HTML
103         soup = BeautifulSoup(table, 'lxml')
104         # Получаем количество строк
105         number_of_rows = len(soup.find_all('tr'))
106         # Получаем количество столбцов
107         number_of_columns = len(soup.find('tr').find_all(['td', 'th']))
108
109         return task_object, number_of_rows, number_of_columns
110
111     def predict_class(self, image: np.ndarray, task_object: str) -> list:
112         """Метод для получения масок для изображения с необходимым классом"""
113         # Передаем в предобученную модель изображение для поиска нужного объекта
114         results = model(image)[0]
115         class_names = model.names
116
117         # Получаем идентификатор нужного класса
118         for id, name in class_names.items():
119             if name == task_object.lower():
120                 class_id = id

```

```

121         break
122
123     # Получаем все маски для классов
124     masks = results.masks.data.cpu().numpy()
125     classes = results.bboxes.cls.cpu().numpy()
126
127     # Получаем список масок для нужного класса
128     selected_masks = [masks[i] for i in range(len(classes)) if int(classes[i]) == class_id]
129
130     return selected_masks
131
132
133 def get_cells_with_mask(self, cells_with_object: list, coords_cells: list, mask: np.ndarray, grid_size: tuple, threshold: float) -> list:
134     '''Метод для получения ячеек таблицы, содержащих объект'''
135     # Определяем размер ячейки
136     cell_height, cell_width = int(mask.shape[0] / grid_size[0]), int(mask.shape[1] / grid_size[1])
137     idx_cell = 0
138
139     for i in range(grid_size[0]):
140         for j in range(grid_size[1]):
141             # Координаты прямоугольника, соответствующего ячейке
142             y1, y2 = i * cell_height, (i + 1) * cell_height
143             x1, x2 = j * cell_width, (j + 1) * cell_width
144
145             # Вырезаем часть маски, соответствующую ячейке
146             cell_mask = mask[y1:y2, x1:x2]
147             # Рассчитываем какую часть ячейки занимает объект
148             coverage_area = np.sum(cell_mask) / cell_mask.size
149
150             # Проверяем, есть ли объект в ячейке
151             if coverage_area >= threshold:
152                 # Сохраняем данные о ячейке
153                 cells_with_object.append(idx_cell)
154                 coords_cells.append((i, j))
155
156             idx_cell += 1
157
158     return cells_with_object, coords_cells
159
160
161 if __name__ == "__main__":
162     # Загружаем модель
163     model = YOLO('best.pt')
164
165     # Целевой сайт
166     target_link = 'https://rucaptcha.com/demo/recaptcha-v2'
167
168     # Настройки user agent
169     USER_AGENT = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.0.0  

    ↪ Safari/537.36"
170     options = webdriver.ChromeOptions()
171
172     options.add_experimental_option("excludeSwitches", ["enable-automation"])
173     options.add_experimental_option('useAutomationExtension', False)
174     options.add_argument(f'user-agent={USER_AGENT}')

```

```

175 options.add_argument(
176     "--disable-blink-features=AutomationControlled"
177 )
178
179 # Передача параметров
180 browser = webdriver.Chrome(options=options)
181 browser.implicitly_wait(30)
182
183 captcha = SolveCaptcha(browser)
184 # Находим фрейм с captcha (автоматизация клика на чекбокс)
185 captcha.find_captcha(target_link)
186
187 # Выполняем распознавание до тех пор, пока фрейм не исчезнет
188 while True:
189     try:
190         # Получение изображения captcha и объекта для поиска
191         task_object, table, src_image, image = captcha.get_captcha()
192         # Получаем необходимые параметры captcha
193         task_object, rows, columns = captcha.get_properties_for_recognition(task_object, table)
194
195         RECURSIVE_CAPTCHA = True # Флаг для captcha, в которых вместо выбранных изображений появляются новые
196         while RECURSIVE_CAPTCHA:
197             # Сбрасываем флаг рекурсии
198             RECURSIVE_CAPTCHA = False
199             # Находим нужный класс на изображении
200             selected_masks = captcha.predict_class(image, task_object)
201
202             cells_with_object, coords = [], []
203             for mask in selected_masks:
204                 # Проходим по выбранным маскам для определения клетки к которой она принадлежит
205                 resized_mask = cv2.resize(mask, (image.shape[1], image.shape[0]), interpolation=cv2.INTER_NEAREST)
206                 cells_with_object, coords = captcha.get_cells_with_mask(cells_with_object, coords, resized_mask, (rows, columns), 0.05)
207
208             # Кликаем по ячейкам с уникальными индексами
209             for cell, coord in list(set(zip(cells_with_object, coords))):
210                 captcha.browser.find_elements(By.TAG_NAME, 'td')[cell].click()
211                 time.sleep(randint(2, 3))
212             # Проверяем наличие новых изображений в данной ячейке
213             src_cell = captcha.browser.find_elements(By.TAG_NAME, 'td')[cell].find_element(By.TAG_NAME, 'img').get_attribute('src')
214             if src_cell != src_image:
215                 # Делаем запрос для получения изображения
216                 response = requests.get(src_cell)
217                 response.raise_for_status()
218                 cell_image = cv2.imdecode(
219                     np.frombuffer(response.content, np.uint8),
220                     cv2.IMREAD_COLOR
221                 )
222
223                 # Заменяем в исходном изображении старую ячейку на новую
224                 x1, x2 = coord[0] * cell_image.shape[0], (coord[0] + 1) * cell_image.shape[0]
225                 y1, y2 = coord[1] * cell_image.shape[1], (coord[1] + 1) * cell_image.shape[1]
226                 image[x1:x2, y1:y2] = cell_image
227
228                 # Устанавливаем флаг рекурсии
229                 RECURSIVE_CAPTCHA = True

```

```
230
231     # Находим кнопку подтверждения выбора и кликаем по ней
232     captcha.browser.find_element(By.XPATH, '//*[@id="recaptcha-verify-button"]').click()
233     time.sleep(randint(3, 5))
234     except ElementClickInterceptedException:
235         captcha.browser.quit()
236     break
237
```

---

## ПОСЛЕДНИЙ ЛИСТ ВКР

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

«\_\_\_» \_\_\_\_\_ 2025 г.

\_\_\_\_\_ А. В. Лаптев