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 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).

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).

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. Capy – 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/I

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 низкорискованным [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).

Capy 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 Capy CAPTCHA:

1. интерактивность:
    1.1. Capy 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).

Capy 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 надёжности и уязвимости различных систем CAPTCHA

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

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 надёжность 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 надёжности:

1. преднамеренное искажение символов (геометрическая деформация, перекрытие, наклоны);
2. использование нестандартных шрифтов, графических помех и шумов;
3. отсутствие чёткой 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. упрощённые 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 надёжности 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 CTC позволяет достигать высокой точности распознавания;
2. современные модели автоматического распознавания речи успешно решают даже зашумлённые аудиозадания;
3. применение генеративных моделей и других методов предварительной обработки аудио позволяет эффективно устранять шумы, повышая точность распознавания.

К ключевым характеристикам надёжности CAPTCHA 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 CAPTCHA можно выделить:

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

2. МЕТОДОЛОГИЯ РЕШЕНИЯ ЗАДАЧ CAPTCHA

2.1. Общие подходы к автоматизированному решению CAPTCHA

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

1. реализация CAPTCHA в аудиоформате:
   1.1. шумоподавление и фильтрация;
   1.2. распознавание речи;
   1.3. преобразование речи в текст.
2. реализация CAPTCHA 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 CAPTCHA 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 CAPTCHA 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 определённых 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 CAPTCHA в текстовом формате в качестве предобработки используется бинаризация и фильтрация фоновых шумов для упрощения следующих этапов обработки.

Бинаризация – процесс преобразования цветного или полутонового 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 съёмки, качество оборудования или передача данных. В частности, для CAPTCHA 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 ОКТ-ИЗОБРАЖЕНИЯХ)
3. частотная фильтрация: применяется в частотной области для удаления определённых частотных компонентов, связанных с шумом.(https://www.xn—-8sbempclcwd3bmt.xn–p1ai/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 CAPTCHA в качестве предобработки используется нормализация и ручная разметка изображений датасета.

Нормализация – процесс приведения изображений к единому стандарту по определённым 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 ин-

формации. Размеченные фотографии используются в различных задачах, от обнаружения объектов до автоматической обработки изображений.($https$: $//annotate.ru/blog/semanticheskaya_razmetka_dlya_mashinnogo_obucheniya$)

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

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 предопределённых 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.($https://aws.amazon.com/what-is/ocr/?nc1 = h_ls$)

Аугментация данных – метод увеличения объёма обучающего набора данных путём 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 CAPTCHA это может использоваться для идентификации определённых 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 CAPTCHA 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 нейросетей 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 будцт рассмотрены различные архитектуры нейронных сетей, которые могут быть использованы для автоматизации решения CAPTCHA.

Архитектуры нейронных сетей для аудио CAPTCHA

Для аудио CAPTCHA доступны следующие архитектуры нейронных сетей и инструменты:

1. Открытые API для работы с языковыми моделями от Google, Microsoft и других;
2. Написать еще несколько вариантов...

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. Tesseract OCR;
2. CRNN + CTC;
3. Sequence-to-Sequence.

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 данныхзадач могут применяться следующие инструменты и архитектуры нейронных сетей:

1. YOLO;
2. DETR;
3. Faster R-CNN.

2.3. Подготовка и аннотация датасетов

Подготовка датасета для тектсовых CAPTCHA

Поскольку в открытом доступе отсутствует достаточное количество данных для формирования сбалансированного датасета, было принято решение о генерации синтетических изображений с использованием специализированных библиотек. В качестве основного инструмента выбрана 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.

Исходный код генератора синтетических CAPTCHA представлен в приложении 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 4.

Примеры сгенерированных и предобработанных CAPTCHA приведены на рисунке ниже:

а) б)

Рис. 2.1 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 чёткими 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. Всё 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, приближённый к реальным условиям использования. Наиболее эффективным методом является автоматизированный парсинг изображений CAPTCHA, представленных на веб-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 был реализован 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 применялась 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, разделённое на сетку из 3×3 или 4×4 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. 2.2.

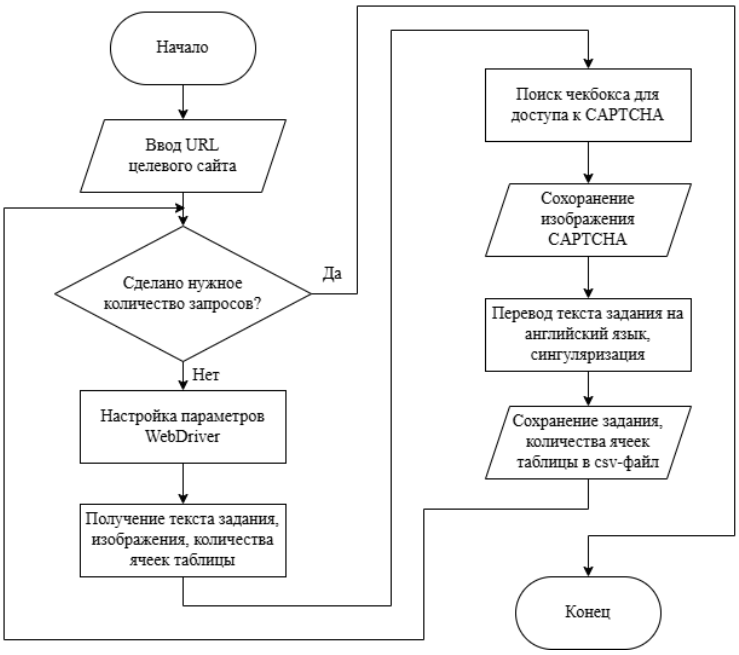

Рис. 2.2 Блок-схема процесса парсинга CAPTCHA.

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 самыхважных этапов работы, поскольку от качества разметки напрямую зависит точность и эффективность последующей работы модели.

Для корректной работы 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. images – изображения;
   1.2. labels – метки к изображениям.
2. Директория val – содержит валидационную выборку:
   2.1. images – изображения;
   2.2. 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

```yaml
path: ../datasets/image_dataset
train: images/train
val: images/val

nc: 9  # Количество классов
names: ['pedestrian transition', 'stair', 'motorcycle', 'bus', 'traffic light', 'car', 'bicycle', 'fire hydrant', 'tractor']
```

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

Рис. 2.3 Пример разметки изображения с тестовой CAPTCHA.

Кроме того, разметка позволяет учесть сразу несколько объектов разных классов на одном изображении, что особенно характерно для CAPTCHA, 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 удается обойтись без аугментации.

3. ЭКСПЕРИМЕНТАЛЬНАЯ ЧАСТЬ И ИССЛЕДОВАТЕЛЬСКИЙ ВКЛАД

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

Современная реализация 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, буквы «O» и цифру «0», буквы «I» и «l» и тому подобное. 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.

Подготовка датасета с изображениями и выбор модели нейронной сети

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 её способность к обобщению на ранее невидимых данных.

Для распознавания текста с переменной длиной последовательности в задачах CAPTCHA наиболее часто применяются следующие архитектуры нейронных сетей:

1. оптическое распознавание символов (OCR);
2. рекуррентные сверточные нейронные сети (CRNN);
3. архитектуры последовательного обучения (Seq-to-Seq).

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 CAPTCHA, 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 предполагалась реализация модели с использованием OCR, поскольку такие системы изначально разрабатывались для задач оптического распознавания текста. В качестве конкретной модели был выбран Tesseract.

Tesseract является одной из наиболее популярных систем OCR с открытым исходным кодом. Tesseract поддерживает более 100 языков, включая сложные письменности [**Klippa**]. В версии 4.0 в модель была интегрирована нейронная сеть на основе долговременной краткосрочной памяти (LSTM), что позволило существенно повысить точность распознавания, особенно при обработке сложных шрифтов и рукописного текста [**GitTesseract**].

Для решения поставленной задачи предполагалось использовать предобученную модель Tesseract и дообучить её на специализированном датасете, содержащем изображения CAPTCHA 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 CAPTCHA, 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 последова-

тельного обучения (Seq-to-Seq), обладающие высокой устойчивостью к вариативности и искажениям, характерным для CAPTCHA.

Рекуррентные сверточные нейронные сети (CRNN)

Сверточно-рекуррентные нейронные сети (CRNN) представляют собой гибридную архитектуру, сочетающую в себе возможности сверточных нейронных сетей (CNN) и рекуррентных нейронных сетей (RNN). Данный подход используется в задачах, связанных с обработкой последовательных данных, таких как распознавание текста, речь и видео [**CRNNHabr**].

Основное преимущество CRNN заключается в способности CNN-части извлекать пространственные признаки из изображений, тогда как RNN-часть позволяет учитывать временные зависимости между последовательными фрагментами данных [**CRNNBook**].

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

1. сверточный блок (CNN): предназначен для выделения признаков из изображений CAPTCHA. 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 CAPTCHA;
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)

Модели последовательного преобразования (Seq-to-Seq) широко применяются для задач, связанных с обработкой последовательностей переменной длины. Они используются в таких областях, как машинный перевод, распознавание речи и анализ текстов [**Seq2Seq**]. Данные модели основаны на архитектуре энкодера-декодера, где первый модуль кодирует входную последовательность в скрытое представление, а второй декодирует его в выходную последовательность.

Одним из ключевых элементов 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 CAPTCHA, так как символы в изображениях могут иметь разную ориентацию и степень искажения.

Кодировщик, в данной модели принимает входное изображение CAPTCHA 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 Seq-to-Seq-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 её оптимизации и улучшении параметров обучения.

3.2. Эксперименты по аудио- и графическим CAPTCHA

Выбор языка программирования и инстументов 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 web-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 её 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.

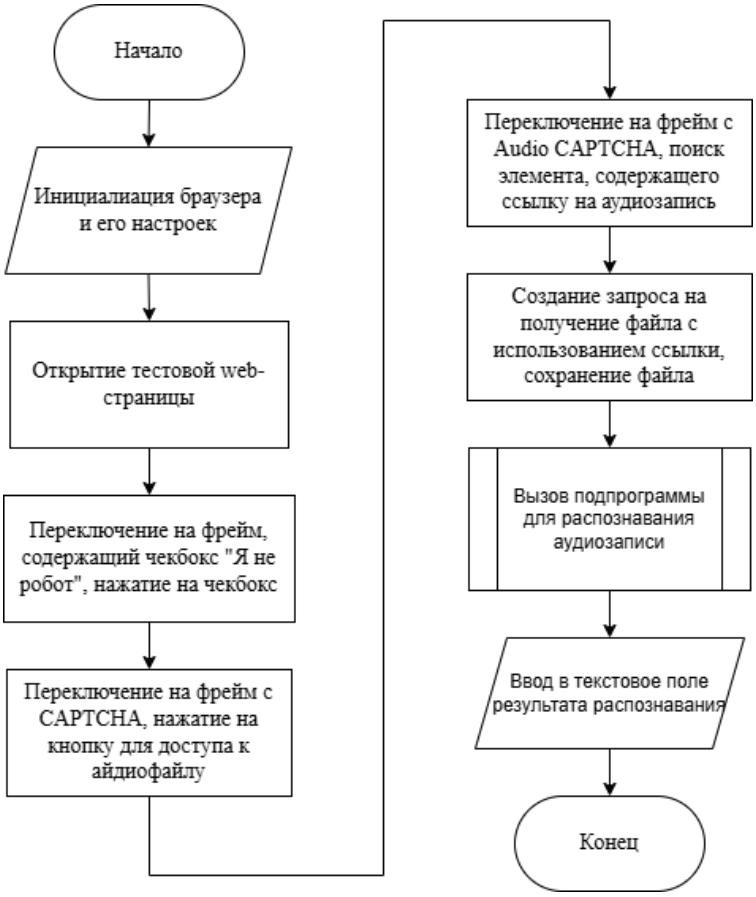

Рис. 3.1 Блок-схема процесса прохождения Audio CAPTCHA.

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 тектстовое 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):

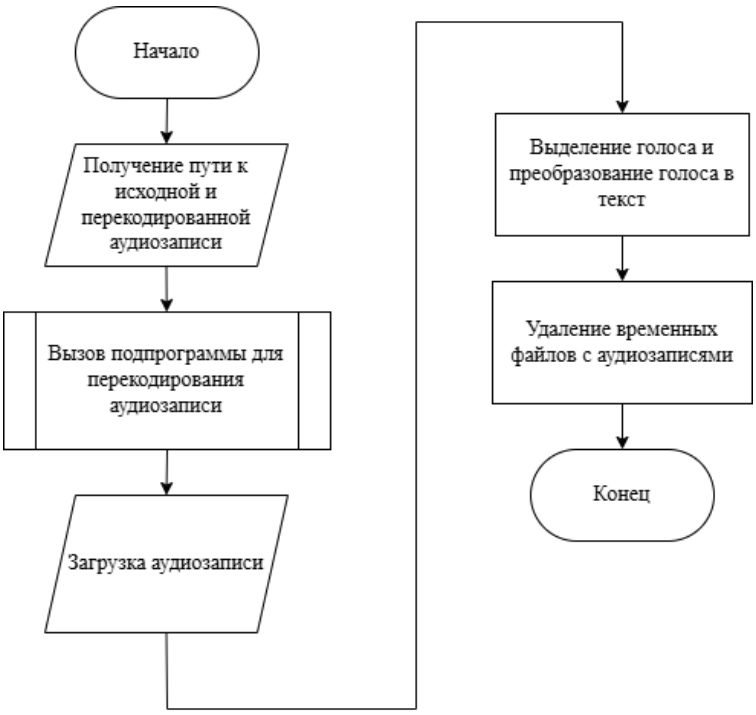

Рис. 3.2 Блок-схема процесса распознавания Audio CAPTCHA.

3.3. Автоматизация решения CAPTCHA в формате изображений

Выбор модели нейронной сети для обучения

CAPTCHA в формате изображений широко используется для защиты ресурсов от автоматизированных ботов и может быть реализована несколькими способами. Как правило, такие CAPTCHA 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 искажённых, при этом изображение разбито на сетку 3×3 или 4×4. Пользователю предлагается выбрать ячейки, содержащие объекты определённого 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.

а) б)

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

Такие CAPTCHA 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 CAPTCHA, 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:

1. YOLO (You Only Look Once) – однопроходная модель, объединяющая классификацию и регрессию ограничивающих рамок в одной

свёрточной 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 съёмки, что подтверждает её 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.

3.4. Исследовательский вклад

Тестирование выбранной модели нейронной сети (text captcha)

Как было установлено в предыдущих разделах, модель последовательного преобразования (Seq-to-Seq) 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 разделён на три подмножества: обучающее, тестовое и валидационное в соотношении 80:10:10. Обучающая выборка использовалась непосредственно для обучения 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.

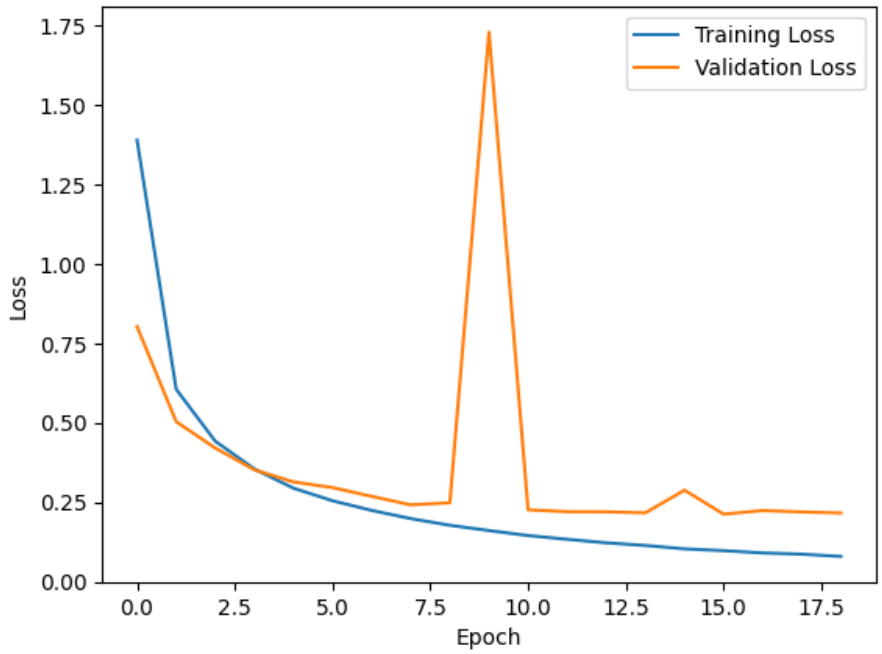

Рис. 3.4 График изменения значений функции потерь в процессе обучения.

Для предотвращения переобучения использовался механизм ранней остановки, согласно которому обучение прекращалось при отсутствии уменьшения значения функции потерь на валидационной выборке в течение трёх 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 её 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 валидационной 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.

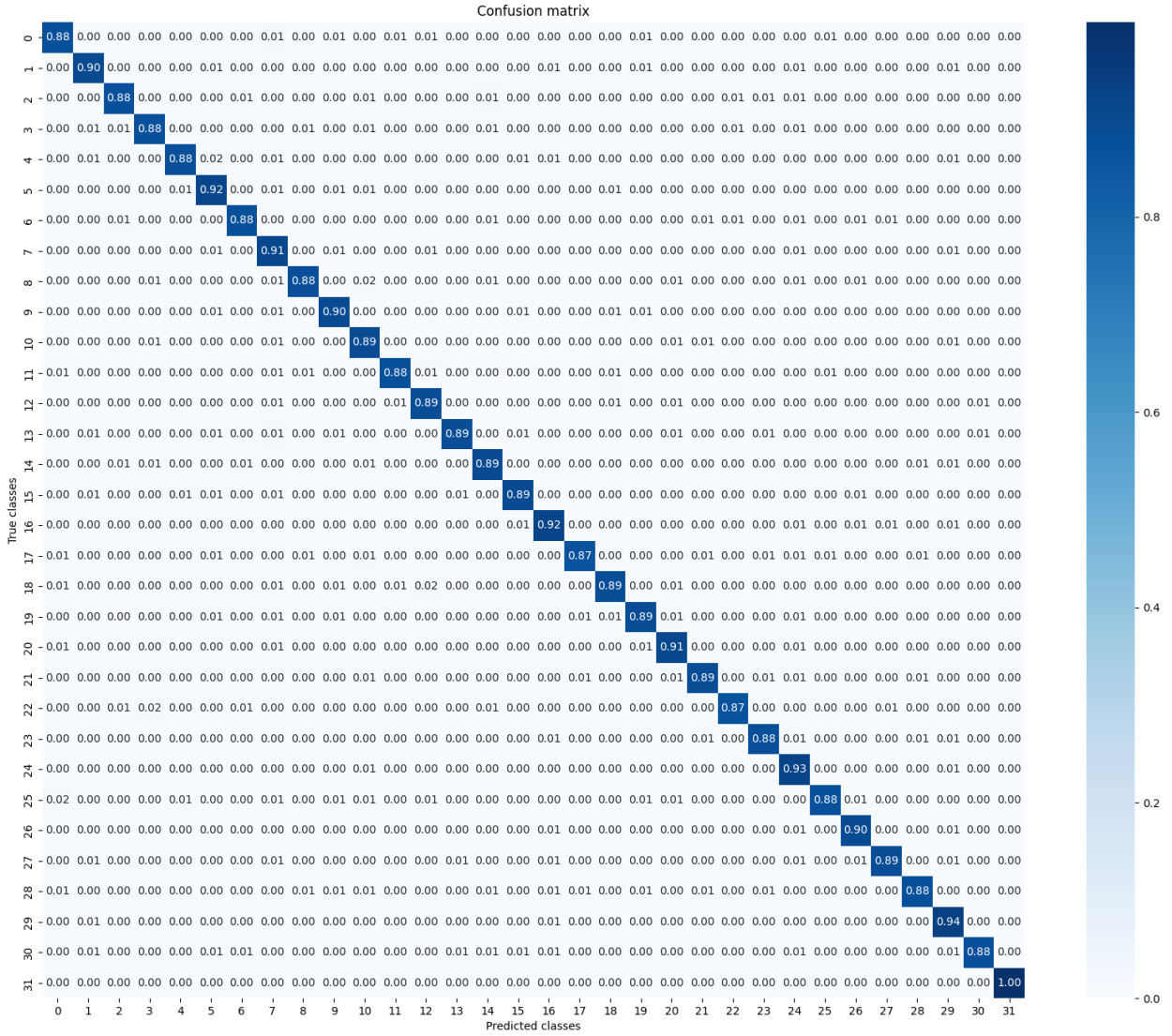

Рис. 3.5 Матрица ошибок для обученной модели.

Анализ полученных результатов показывает, что точность распознавания последовательностей значительной длины остаётся относительно низкой. Это можно объяснить высокой зависимостью модели Seq-to-Seq от объёма 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 её обучения.

Тестирование модели (image captcha)

После завершения обучения модель 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 чекбокса «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 CAPTCHA на ячейки (в зависимости от размера сетки – 3×3 или 4×4);
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 CAPTCHA в случае, если результат оказался некорректным (что также фиксировалось в логах).

Тестирование было организовано в виде цикла, позволяющего автоматически проходить CAPTCHA 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 CAPTCHA представлен на блок-схеме ниже.

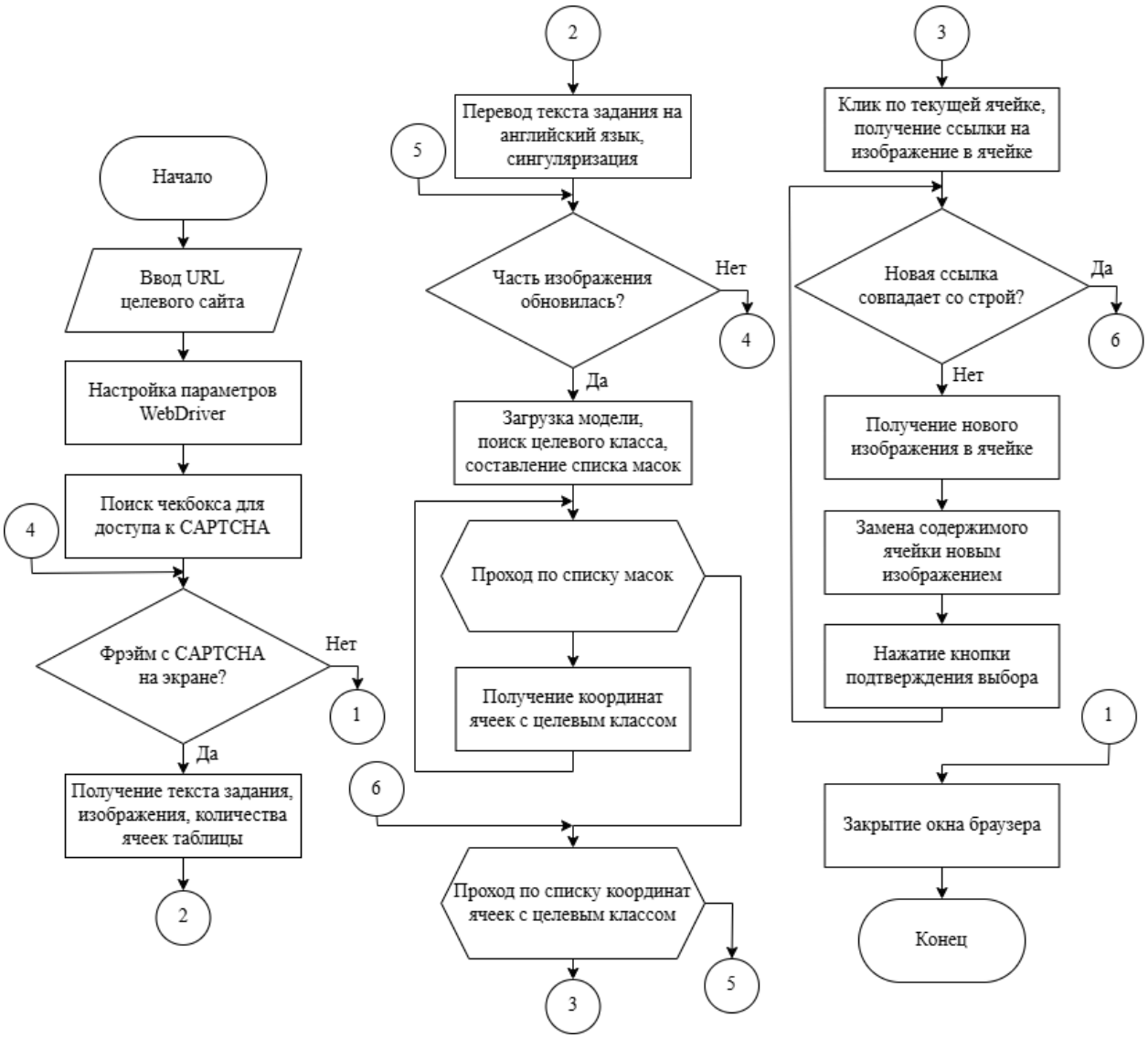

Рис. 3.6 Блок-схема процесса прохождения CAPTCHA.

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

Обучение модели

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 зашумлённые 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 ultralytics, 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 640×640 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.

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

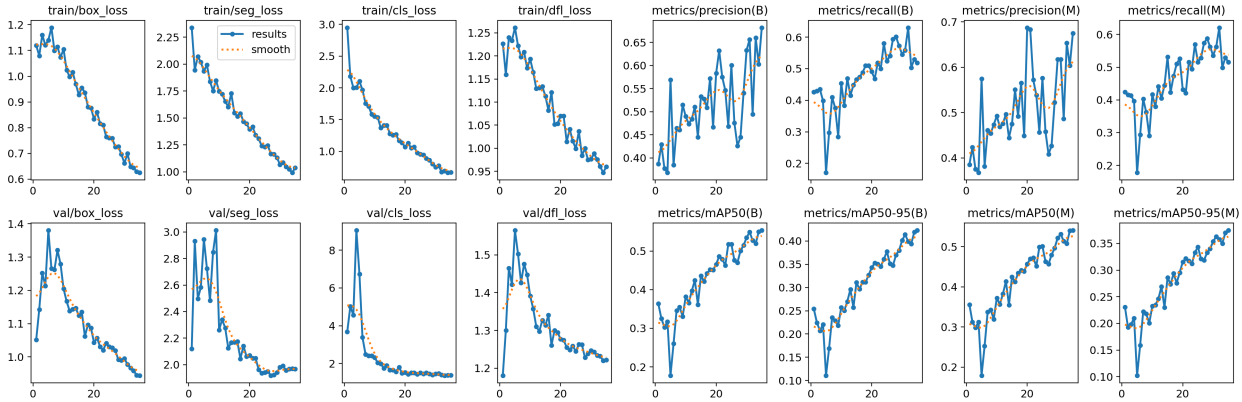

Рис. 3.7 Изменение ключевых метрик в процессе обучения.

Также, была построена нормализованная матрица ошибок для определения точности предсказания необходимых классов на валидационной выборке, которая представлена на рис. 3.8.

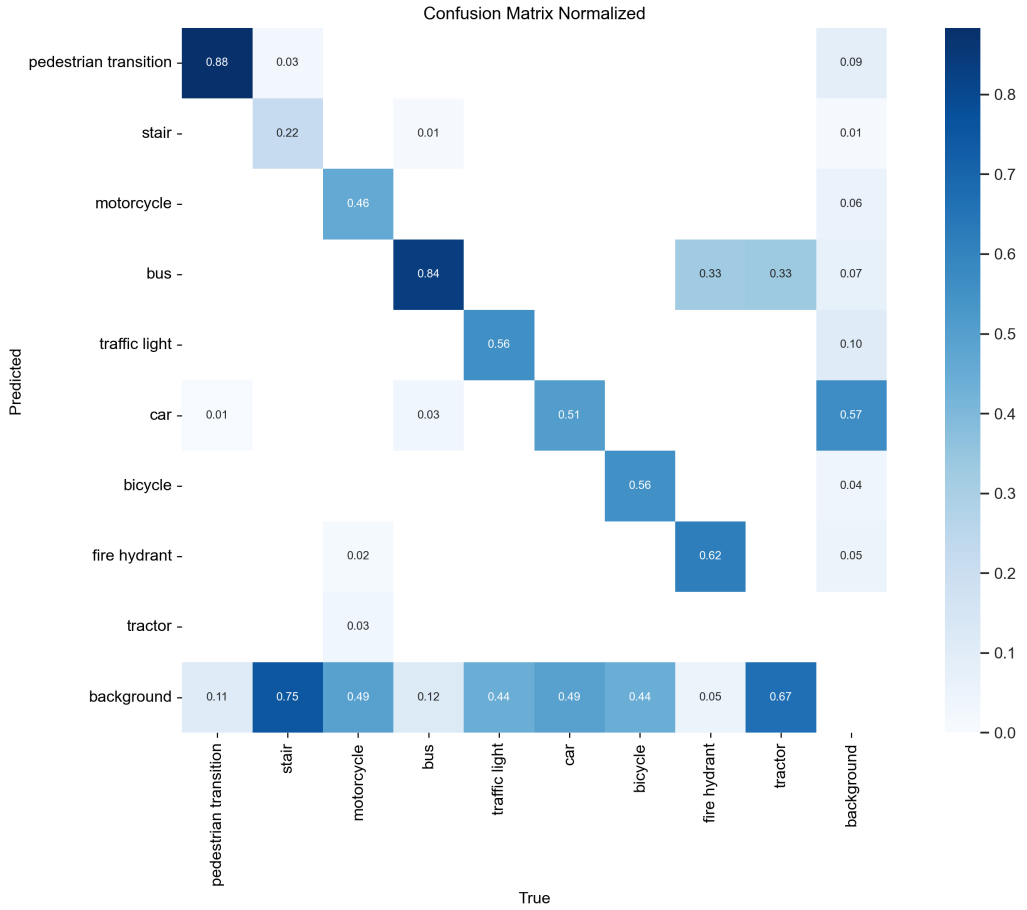

Рис. 3.8 Матрица ошибок для изображений валидационной выборки.

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

```
'''Файл с классаом для решения audiocaptcha'''
import speech_recognition as sr
import subprocess
import logger
import os


logger = logger.ConfigLogger(__name__)

class AudioCaptchaSolver():
    '''Класс решателя audio captcha'''

    def __init__(self):
        '''Конструктор класса'''
        # Создаем объект распознавателя речи
        self.recognizer = sr.Recognizer()
        # Распознанное текстовое сообщение
        self.text_message = None


    def recognition_audio(self, path_to_audio: str) -> str:
        '''
        Метод распознавания аудиофайла
        Файлы сохраняются в формате mp3 (обычно содержат шум, кроме мест, где слышен голос)
        '''

        # Преобразование mp3 файла в формат, который подходит для распознавания
        mp3_file = path_to_audio
        wav_file = './audio/audiocaptcha.wav'

        if os.name == 'nt':
            subprocess.run(['C:/ffmpeg/bin/ffmpeg.exe', '-i', mp3_file, wav_file])
        else:
            subprocess.run(['ffmpeg', '-i', mp3_file, wav_file])

        try:
            # Загружаем аудио файл
            audio_captcha = sr.AudioFile(wav_file)

            # Распознаем речь из аудио файла
            with audio_captcha as voice:
                audio_data = self.recognizer.record(voice)
                text_message = self.recognizer.recognize_google(audio_data, language='en-US')
            logger.log_info('Распознавание речи завершено успешно!')
        except Exception as e:
```

```
        logger.log_warning(f'Распознавание завершилось с ошибкой: {e}')

    if text_message:
        self.text_message = text_message
        os.remove(mp3_file)
        os.remove(wav_file)

    return self.text_message
```

Листинг 2

Исходный код автоматизированного решения Audio CAPTCHA

```
'''Это основной файл проекта, в котором будут вызываться классы и методы для решения всех популярных видов captcha'''
from selenium import webdriver
from selenium.webdriver.remote.webdriver import WebDriver
from selenium.webdriver.common.by import By

from random import randint
import time
import requests
import os

from audiocaptcha import AudioCaptchaSolver


class MainWorker():
    '''
    Основной класс проекта, который управляет вызовом дочерних классов для решения определенных видов captcha
    На начальном этапе здесь также будет все, что касается получения captcha с веб-страницы
    '''

    def __init__(self, browser: WebDriver):
        '''Конструктор класса'''
        super().__init__()
        self.browser = browser


    def get_captcha(self, link: str) -> str:
        '''Метод получения captcha со страницы'''
        # Проходим по ссылке
        self.browser.get(link)
        time.sleep(randint(3, 5))

        # Переключаемся на фрейм с чекбоксом captcha
        self.browser.switch_to.frame(self.browser.find_element(By.XPATH, '//*[@id="g-recaptcha"]/div/div/iframe'))
        # Кликаем по чекбоксу "Я не робот"
        self.browser.find_element(By.XPATH, '/html/body/div[2]/div[3]/div[1]/div/div/span').click()
        time.sleep(randint(3, 5))

        # Переключаемся на обычную web-страницу
        self.browser.switch_to.default_content()
        # Переключаемся на фрейм с картинкой captcha
        self.browser.switch_to.frame(self.browser.find_element(By.XPATH, '/html/body/div[2]/div[4]/iframe'))
        # 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 айдиозаписью
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          # Настройки 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
            ↪  Safari/537.36"
90
91          select_browser = randint(1, 10)
92
93          # Выбор браузера и опций характерных для него
94          if select_browser < 5:
95              options = webdriver.ChromeOptions()
96          else:
```

```
        options = webdriver.EdgeOptions()

    options.add_experimental_option("excludeSwitches", ["enable-automation"])
    options.add_experimental_option('useAutomationExtension', False)
    options.add_argument(f"user-agent={USER_AGENT}")
    options.add_argument("--disable-blink-features=AutomationControlled")

    # Передача параметров
    if select_browser < 5:
        browser = webdriver.Chrome(options=options)
    else:
        browser = webdriver.Edge(options=options)
    browser.implicitly_wait(30)

    # Создаем аудиофайл по указанному пути с captcha
    solver = MainWorker(browser)
    path_to_audio = solver.get_captcha(link)

    # Запускаем распознавание
    captcha_solver = AudioCaptchaSolver()
    response = captcha_solver.recognition_audio(path_to_audio)

    # Вставляем результат распознавания в поле ввода
    solver.paste_response(response)
    time.sleep(randint(10, 15))
```

Листинг 3

Исходный код генератора синтетических CAPTCHA

```
from captcha.image import ImageCaptcha

from random import randint, shuffle
import numpy as np
import os

from textcaptcha.preprocessing_image import preprocessing_image


def generate_image(path_to_file: str, alphabet: list, number_of_start:int, number_of_captcha: int, size_of_image: tuple) -> list:
    # Генерация текстовых captcha
    text = ImageCaptcha(size_of_image[0], size_of_image[1], ['./fonts/arial.ttf', './fonts/comic.ttf', './fonts/cour.ttf', './fonts/georgia.ttf'])
    # Структура возвращаемого списка: [filename, label, (width, height)]
    filenames = []
    for _ in range(number_of_start, number_of_captcha):
        captcha_text = [alphabet[randint(0, len(alphabet) - 1)] for _ in range(randint(4, 7))]
        shuffle(captcha_text)
        text.write(''.join(captcha_text), f'{path_to_file}/{""".join(captcha_text)}.png')
        filenames.append(
            [f'{path_to_file}/{""".join(captcha_text)}.png',
            ''.join(captcha_text)]
        )

    return filenames

```

```

if __name__ == '__main__':
    # Алфавит допустимых символов
    alphabet = 'ABCDEFGHJKLMNPQRSTWXYZ023456789'
    # Создание директории для хранения полноценных синтетических текстовых captcha
    path_to_dataset = '../datasets/captcha'
    if not os.path.isdir(path_to_dataset):
        os.mkdir(path_to_dataset)
    # Создаем датасет из нужного полноценных синтетических captcha длиной от 4 до 7 символов размером 250x60
    filenames = generate_image(path_to_dataset, list(alphabet), 0, 100000, (250, 60))

    # Предобработка изображений
    preprocessing_image(filenames)

    # Для отладки без создания датасета с нуля
    numpy_data = np.array(filenames, dtype=object)
    np.save('data.npy', numpy_data)
```

Листинг 4

Исходный код для предобработки изображений датасета

```
import cv2
import numpy as np


def preprocessing_image(list_filenames: list):
    '''Функция для предобработки изображений или изображения для предсказания'''
    # Предобработка изображений с CAPTCHA
    for file in list_filenames:
        # Открытие изображения в градациях серого
        gray_image = cv2.imread(file[0], 0)
        # Приведение всех изображений к одному размеру ширина х высота)
        resized_image = cv2.resize(gray_image, (250, 60))

        # Морфологический фильтр (дилатация) для сужения символов и более четкого отделения их друг от друга
        morph_kernel = np.ones((3, 3))
        dilatation_image = cv2.dilate(resized_image, kernel=morph_kernel, iterations=1)

        # Применяем пороговую обработку, чтобы получить только черные и белые пиксели
        _, thresholder = cv2.threshold(
            dilatation_image,
            0,
            255,
            cv2.THRESH_BINARY + cv2.THRESH_OTSU
        )

        cv2.imwrite(file[0], thresholder)
```

Листинг 5

Исходный код для создания датасета в формате тензоров

```
import pandas as pd
```

```
import tensorflow as tf
from keras._tf_keras.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split


def parse_data(image_path: list, encoder_labels: list, decoder_labels: list) -> tuple[tf.Tensor, list]:
    '''Функция для склеивания изображений и лейблов для датасета'''
    image = tf.io.read_file(image_path)
    image = tf.image.decode_png(image, channels=1)
    image = tf.cast(image, tf.float32) / 255.0

    return (image, encoder_labels), decoder_labels


def create_dataset(images: list, encoder_labels: list, decoder_labels: list, shuffle = True, batch_size = 32) -> tf.data.Dataset:
    '''Функция для создания датасета, понятного для TensorFlow'''
    dataset = tf.data.Dataset.from_tensor_slices((images, encoder_labels, decoder_labels))
    dataset = dataset.map(lambda x, y, w: parse_data(x, y, w))
    if shuffle == True:
        dataset = dataset.shuffle(len(images)).batch(batch_size)
    else:
        dataset = dataset.batch(batch_size)

    return dataset


def create_dataframe(images: list) -> pd.DataFrame:
    '''Функция для создания датафреймов на основе списков'''

    # Создание файла с лейблами о содержимом изображений с CAPTCHA
    filenames = [objects[0] for objects in images]
    list_labels = [objects[1] for objects in images]

    # Создание DataFrame для сохранения соотвествия между путями, лейблами и размерами для каждого элемента датасета
    data = {
        'filename': filenames,
        'label': list_labels,
    }

    return pd.DataFrame(data)


def preparing_dataset(dataframe: pd.DataFrame, alphabet: str, shuffle = True) -> tuple[
        tuple[tf.data.Dataset, tf.data.Dataset, tf.data.Dataset, list],
        tuple[tf.data.Dataset, tf.data.Dataset, tf.data.Dataset, list]
    ]:
    '''Подготовка датасета'''

    # Сохранение отдельных составляющих DataFrame
    X_captcha, y_captcha = dataframe['filename'].tolist(), dataframe['label'].tolist()

    dict_alphabet = {alphabet[i]:i for i in range(len(alphabet))}
    start_token = len(alphabet)  # Индекс токена <start>
    end_token = len(alphabet) + 1  # Индекс токена <end>

```

```
    # Кодируем лейблы с добавлением токена <start> для кодера
    encoder_labels = [[start_token] + [dict_alphabet[char] for char in label] for label in y_captcha]

    # Кодируем лейблы с добавлением токена <end> для декодера
    decoder_labels = [[dict_alphabet[char] for char in label] + [end_token] for label in y_captcha]

    # Преобразование меток в тензоры
    encoder_tensors = pad_sequences(encoder_labels, maxlen=8, padding='post')
    decoder_tensors = pad_sequences(decoder_labels, maxlen=8, padding='post')

    # Создание датасета
    dataset = create_dataset(X_captcha, encoder_tensors, decoder_tensors, shuffle)

    return dataset


def create_dataset_for_captcha(filenames: list, alphabet: str) -> tuple[
        tuple[tf.data.Dataset, tf.data.Dataset, tf.data.Dataset, list],
        tuple[tf.data.Dataset, tf.data.Dataset, tf.data.Dataset, list]
    ]:
    '''Функция для создания датасета на основе алфавита и имен файлов'''

    # Создание датафрейма для удобства последующей обработки
    captcha_dataframe = create_dataframe(filenames)

    # Разделение датасета на обучающую и тестовую выборки
    train_captcha_df, test_captcha_df = train_test_split(captcha_dataframe, test_size=0.2, random_state=42)
    # Разделение тестовой части датасета на валидационную и тестовую выборки
    val_captcha_df, test_captcha_df = train_test_split(test_captcha_df, test_size=0.5, random_state=42)

    train_dataset = preparing_dataset(train_captcha_df, alphabet)
    val_dataset = preparing_dataset(val_captcha_df, alphabet)
    test_dataset = preparing_dataset(test_captcha_df, alphabet, False)

    return train_dataset, val_dataset, test_dataset
```

Листинг 6

Исходный код CRNN модели

```
import tensorflow as tf
import numpy as np
from keras._tf_keras.keras.layers import Input, Conv2D, MaxPooling2D, Reshape, Dense, Dropout, Bidirectional, GRU,
↪   BatchNormalization
from keras._tf_keras.keras.regularizers import l2
from keras._tf_keras.keras.models import Model
from keras._tf_keras.keras import backend as K
from keras._tf_keras.keras.backend import ctc_decode
from keras._tf_keras.keras.optimizers import Adam
from keras._tf_keras.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from keras._tf_keras.keras.saving import register_keras_serializable


def decode_predictions(preds, max_length, alphabet):
    # Используем CTC-декодирование для предсказаний
```

```
    decoded_preds, _ = ctc_decode(preds, input_length=np.ones(preds.shape[0]) * preds.shape[1])
    texts = []
    for seq in decoded_preds[0]:
        text = ''.join([alphabet[i] for i in seq.numpy() if i != -1])  # Исключаем 'blank' символы
        texts.append(text)
    return texts


def decode_batch_predictions(pred):
    # CTC decode
    decoded, _ = ctc_decode(pred, input_length=np.ones(pred.shape[0]) * pred.shape[1],
                            greedy=True)
    decoded_texts = []

    # Преобразование в текст
    for seq in decoded[0]:
        text = ''.join([chr(x) for x in seq if x != -1])  # Пропускаем -1 (пустые символы CTC)
        decoded_texts.append(text)
    return decoded_texts


# Функция CTC Loss
# Функция для декодирования предсказаний модели
@register_keras_serializable(package='Custom', name='ctc_loss')
def ctc_loss(y_true, y_pred):
    # Формируем входные данные для CTC
    input_lenght = tf.ones(shape=(tf.shape(y_pred)[0], 1)) * tf.cast(tf.shape(y_pred)[1], tf.float32)
    label_length = tf.ones(shape=(tf.shape(y_true)[0], 1)) * 7
    return tf.reduce_mean(K.ctc_batch_cost(y_true, y_pred, input_lenght, label_length))


# Модель
def build_model(num_of_classes):
    '''Создание модели'''
    # Входной слой
    input_layer = Input((60, 250, 1))

    # Первый сверточный блок
    x = Conv2D(32, (3, 3), activation='relu', padding='same', kernel_regularizer=l2(0.003))(input_layer)
    x = BatchNormalization()(x)
    x = Conv2D(32, (3, 3), activation='relu', kernel_regularizer=l2(0.003))(x)
    x = MaxPooling2D((1, 2))(x)
    x = Dropout(0.25)(x)  # Dropout после каждого блока

    # Второй сверточный блок
    x = Conv2D(64, (3, 3), activation='relu', padding='same', kernel_regularizer=l2(0.003))(x)
    x = BatchNormalization()(x)
    x = Conv2D(64, (3, 3), activation='relu', kernel_regularizer=l2(0.003))(x)
    x = MaxPooling2D((1, 2))(x)
    x = Dropout(0.3)(x)

    # Третий сверточный блок
    x = Conv2D(128, (3, 3), activation='relu', padding='same', kernel_regularizer=l2(0.003))(x)
    x = BatchNormalization()(x)
    x = Conv2D(128, (3, 3), activation='relu', kernel_regularizer=l2(0.003))(x)
```

```
    x = MaxPooling2D((1, 2))(x)
    x = Dropout(0.4)(x)

    # Изменяем размерность тензора
    x = Reshape((-1, x.shape[-1] * x.shape[-2]))(x)

    # Первый рекурентный блок
    x = Bidirectional(GRU(128, return_sequences=True))(x)
    x = BatchNormalization()(x)
    x = Dropout(0.6)(x)

    # Второй рекурентный блок
    x = Bidirectional(GRU(128, return_sequences=True))(x)
    x = BatchNormalization()(x)
    x = Dropout(0.6)(x)

    # Третий рекурентный блок
    x = Bidirectional(GRU(128, return_sequences=True))(x)
    x = BatchNormalization()(x)
    x = Dropout(0.6)(x)

    # Выходной слой
    outputs = Dense(num_of_classes + 1, activation='softmax')(x)

    # Создание модели
    model = Model(inputs=input_layer, outputs=outputs)

    return model


def fit_crnn(num_of_classes, train, val):
    # Компиляция модели
    model = build_model(num_of_classes)
    optimizer = Adam(learning_rate=0.001, weight_decay=1e-6)
    model.compile(
        loss=ctc_loss,
        optimizer=optimizer
    )

    # Вывод структуры модели
    model.summary()

    lr_sheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, min_lr=1e-6)
    early_stop = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

    # Обучение модели
    history = model.fit(
        train,
        validation_data=val,
        epochs=15,
        callbacks=[early_stop, lr_sheduler]
    )

    model.save('crnn_model.keras')

```

```
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")
```

```
    encoder_state_input = layers.Input(shape=(256,), name="encoder_state_input")

    x = layers.Embedding(alphabet_size, 128)(decoder_inputs)
    rnn_output, decoder_state = layers.GRU(256, return_sequences=True, return_state=True)(x, initial_state=encoder_state_input)

    # Attention
    attention_output = layers.AdditiveAttention()([rnn_output, encoder_state_input])
    x = layers.Concatenate()([rnn_output, attention_output])
    decoder_outputs = layers.Dense(alphabet_size, activation="softmax")(x)

    return Model([decoder_inputs, encoder_state_input], [decoder_outputs, decoder_state], name="decoder")


def fit_seq_to_seq(number_of_classes: int, train_dataset: tf.data.Dataset, val_dataset: tf.data.Dataset) -> tuple[Model, dict]:
    # Построение полной модели
    encoder = build_encoder()
    decoder = build_decoder(number_of_classes + 2)

    # Полная модель
    encoder_inputs = encoder.input
    decoder_inputs = layers.Input(shape=(None,), name="decoder_inputs")

    _, encoder_state = encoder(encoder_inputs)
    decoder_output, _ = decoder([decoder_inputs, encoder_state])

    seq2seq_model = Model([encoder_inputs, decoder_inputs], decoder_output, name="seq2seq_model")

    # Компиляция модели
    seq2seq_model.compile(
        loss="sparse_categorical_crossentropy",
        optimizer="adam",
        metrics=["accuracy"]
    )

    seq2seq_model.summary()

    lr_sheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, min_lr=1e-6)
    early_stop = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

    # Обучение модели
    history = seq2seq_model.fit(
        train_dataset,
        validation_data=val_dataset,
        epochs=20,
        callbacks=[early_stop, lr_sheduler]
    )

    seq2seq_model.save('seq_to_seq_model.keras')

    return seq2seq_model, history
```

Листинг 8

Исходный код тестирования модели

```
import numpy as np
import tensorflow as tf
from keras._tf_keras.keras.models import load_model


if __name__ == '__main__':
    import matplotlib.pyplot as plt
    import seaborn as sbn

    # Алфавит допустимых символов
    alphabet = ' ABCDEFGHJKLMNPQRSTWXYZ023456789'

    list_filenames = np.load('data.npy', allow_pickle=True)
    # Создание единого датасета
    captcha_dataset = create_dataset_for_captcha(list_filenames, alphabet)
    if False:
        # Обучение модели
        model_captcha, history_captcha = fit_seq_to_seq(len(alphabet), captcha_dataset[0], captcha_dataset[1])
        # Построение графика изменения val_loss и loss
        plt.plot(history_captcha.history['loss'], label='Training Loss')
        plt.plot(history_captcha.history['val_loss'], label='Validation Loss')
        plt.xlabel('Epoch')
        plt.ylabel('Loss')
        plt.legend()
        # Сохраняем график для отчета
        plt.savefig('Model_loss.png')

    # Загружаем предобученную модель и получаем предсказания для тестовой выборки
    model = load_model('seq_to_seq_model.keras')
    predictions = model.predict(captcha_dataset[2])

    # Переводим предсказания из представления вероятностей в классы
    pred_classes = np.argmax(predictions, axis=-1)
    captcha_labels = [label.numpy() for _, label in captcha_dataset[2].unbatch()]
    captcha_labels = np.array(captcha_labels)

    # Убираем padding
    def remove_padding(sequences, padding_value=0):
        return [seq[seq != padding_value] for seq in sequences]

    # Убираем padding из предсказаний и меток
    pred_classes_no_padding = remove_padding(pred_classes, padding_value=0)
    true_labels_no_padding = remove_padding(np.array(captcha_labels), padding_value=0)

    # Проверяем размеры списков после удаления padding
    print(f"Количество предсказаний: {len(pred_classes_no_padding)}")
    print(f"Количество меток: {len(true_labels_no_padding)}")

    # Проверяем совпадение предсказаний и истинных меток посимвольно
    sequence_accuracy = np.mean(
        [np.array_equal(pred, true) for pred, true in zip(pred_classes, captcha_labels)]
    )
    print(f"Точность последовательностей (без padding): {sequence_accuracy:.4f}")

    # Расчет точности символов (character-level accuracy)
```

```
total_characters = np.prod(captcha_labels.shape)
correct_characters = np.sum(pred_classes == captcha_labels)
character_accuracy = correct_characters / total_characters
print(f"Точность символов: {character_accuracy:.4f}")

from sklearn.metrics import confusion_matrix
# Построение матрицы ошибок для анализа
true_symb, pred_symb = [], []

for true_seq, pred_seq in zip(true_labels_no_padding, pred_classes_no_padding):
    true_symb.extend(true_seq)
    pred_symb.extend(pred_seq)
cm = confusion_matrix(true_symb, pred_symb)

plt.figure(figsize=(10, 7))
sbn.heatmap(cm, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Predicted classes')
plt.ylabel('True classes')
plt.title('Confusion matrix')
# plt.show()
plt.savefig('Confusion_matrix.png')

from collections import defaultdict

sequence_accuracy_by_length = defaultdict(list)
for pred, true in zip(pred_classes_no_padding, true_labels_no_padding):
    seq_len = len(true)
    is_correct = np.array_equal(pred, true)
    sequence_accuracy_by_length[seq_len].append(is_correct)

# Считаем точность для каждой длины
for length, results in sequence_accuracy_by_length.items():
    acc = np.mean(results)
    print(f"Длина {length}: Точность {acc:.4f}")
```

Листинг 9

Исходный код получения CAPTCHA с целевого сайта

```
# Подключение библиотек для работы с браузером
from selenium import webdriver
from selenium.webdriver.remote.webdriver import WebDriver
from selenium.webdriver.common.by import By

# Подключение библиотек для работы с текстом заданиия captcha
from deep_translator import GoogleTranslator
import inflect

# Библиотека для парсинга HTML
from bs4 import BeautifulSoup

from random import randint
import time
import requests
import os
```

```
import csv


class GetCaptcha():
    '''
    Основной класс проекта, который управляет вызовом дочерних классов для решения определенных видов captcha
    На начальном этапе здесь также будет все, что касается получения captcha с веб-страницы
    '''

    def __init__(self, browser: WebDriver):
        '''Конструктор класса'''
        super().__init__()
        self.browser = browser


    def get_captcha(self, link: str, cnt: int) -> tuple[str, str, str]:
        '''Метод получения captcha со страницы'''
        # Проходим по ссылке
        self.browser.get(link)
        time.sleep(randint(3, 5))

        # Переключаемся на фрейм с чекбоксом captcha
        self.browser.switch_to.frame(self.browser.find_element(
            By.XPATH,
            '//*[@id="g-recaptcha"]/div/div/iframe'
        ))
        # Кликаем по чекбоксу "Я не робот"
        self.browser.find_element(
            By.XPATH,
            '/html/body/div[2]/div[3]/div[1]/div/div/span'
        ).click()
        time.sleep(randint(3, 5))

        # Переключаемся на обычную web-страницу
        self.browser.switch_to.default_content()
        # Переключаемся на фрейм с картинкой captcha
        self.browser.switch_to.frame(self.browser.find_element(
            By.XPATH,
            '/html/body/div[2]/div[4]/iframe'
        ))
        # Находим элемент, содержащий ссылку на исходное изображение
        image = self.browser.find_element(
            By.XPATH,
            '//*[@id="rc-imageselect-target"]/table/tbody'+
            '/tr[1]/td[1]/div/div[1]/img'
        ).get_attribute('src')
        # Делаем запрос для получения файла
        response = requests.get(image)
        response.raise_for_status()

        # Получаем название объекта, который надо найти
        object_name = self.browser.find_element(
            By.XPATH,
            '//*[@id="rc-imageselect"]/div[2]/div[1]/div[1]'+
            '/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 колличества 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         # Настройки 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
            ↪  Safari/537.36"
111         options = webdriver.ChromeOptions()
112
113         options.add_experimental_option("excludeSwitches", ["enable-automation"])
114         options.add_experimental_option('useAutomationExtension', False)
115         options.add_argument(f"user-agent={USER_AGENT}")
116         options.add_argument(
117             "--disable-blink-features=AutomationControlled"
118         )
119
120         # Передача параметров
121         browser = webdriver.Chrome(options=options)
122         browser.implicitly_wait(30)
123
124         captcha = GetCaptcha(browser)
125         # 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
                ↪  количества символов
133             task_object = singular.singular_noun(task_object)
134             if task_object.lower() == 'hydrant':
135                 task_object = 'fire hydrant'
136
137         # Получаем количество ячеек
138         rows, columns = captcha.get_number_of_cells(table)
139
140         # Запись полученных параметров в csv-файл
141         with open('images_for_captcha.csv', 'a') as datasetFile:
142             csv_rows = csv.writer(datasetFile, quoting=csv.QUOTE_NONE)
143             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
```

```python
from bs4 import BeautifulSoup

# Библиотека для работы с изображениями
from ultralytics import YOLO
import cv2
import numpy as np

from random import randint
import time
import requests


class SolveCaptcha():
    '''
    Основной класс проекта, который управляет вызовом дочерних классов для решения определенных видов captcha
    На начальном этапе здесь также будет все, что касается получения captcha с веб-страницы
    '''

    def __init__(self, browser: WebDriver):
        '''Конструктор класса'''
        super().__init__()
        self.browser = browser


    def find_captcha(self, link: str):
        # Проходим по ссылке
        self.browser.get(link)
        time.sleep(randint(3, 5))

        # Переключаемся на фрейм с чекбоксом captcha
        self.browser.switch_to.frame(self.browser.find_element(
            By.XPATH,
            '//*[@id="g-recaptcha"]/div/div/iframe'
        ))
        # Кликаем по чекбоксу "Я не робот"
        self.browser.find_element(By.XPATH, '/html/body/div[2]/div[3]/div[1]/div/div/span').click()
        time.sleep(randint(3, 5))

        # Переключаемся на обычную web-страницу
        self.browser.switch_to.default_content()
        # Переключаемся на фрейм с картинкой captcha
        self.browser.switch_to.frame(self.browser.find_element(
            By.XPATH,
            '/html/body/div[2]/div[4]/iframe'
        ))


    def get_captcha(self) -> tuple[str, str, str, np.ndarray]:
        '''Метод получения captcha со страницы'''
        # Находим элемент, содержащий ссылку на исходное изображение
        src_image = self.browser.find_element(
            By.XPATH,
            '//*[@id="rc-imageselect-target"]/table/tbody/'+
            'tr[1]/td[1]/div/div[1]/img'
        ).get_attribute('src')
```

```
        # Делаем запрос для получения файла
        response = requests.get(src_image)
        response.raise_for_status()

        # Получаем название объекта, который надо найти
        object_name = self.browser.find_element(
            By.XPATH,
            '//*[@id=”rc-imageselect”]/div[2]/div[1]/div[1]/'+
            'div/strong'
        ).text

        # Получаем таблицу с кусочками изображения
        table = self.browser.find_element(
            By.XPATH,
            '//*[@id=”rc-imageselect-target”]/table'
        ).get_attribute('outerHTML')

        # Преобразование байтовой последовательности в изображение
        image = cv2.imdecode(np.frombuffer(response.content, np.uint8), cv2.IMREAD_COLOR)

        return object_name, table, src_image, image


    def get_properties_for_recognition(self, task_object: str, table: str) -> tuple[str, int, int]:
        '''Метод для получения необходимых параметров для распознавания на картинке'''
        # Перевод названия объекта на английский и сохранение его в единственном числе
        task_object = GoogleTranslator(source='auto', target='en').translate(task_object)
        singular = inflect.engine()
        if len(task_object) > 3:
            # Исключаем ошибки с множественным числом для слов, которые не могут быть во множественном числе из-за малого
            ↪ количества символов
            task_object = singular.singular_noun(task_object)
            if task_object.lower() == 'hydrant':
                task_object = 'fire hydrant'

        # Парсинг HTML
        soup = BeautifulSoup(table, 'lxml')
        # Получаем количество строк
        number_of_rows = len(soup.find_all('tr'))
        # Получаем количество столбцов
        number_of_columns = len(soup.find('tr').find_all(['td', 'th']))

        return task_object, number_of_rows, number_of_columns


    def predict_class(self, image: np.ndarray, task_object: str) -> list:
        '''Метод для получения масок для изображения с необходимым классом'''
        # Передаем в предобученную модель изображение для поиска нужного объекта
        results = model(image)[0]
        class_names = model.names

        # Получаем идентификатор нужного класса
        for id, name in class_names.items():
            if name == task_object.lower():
                class_id = id
```

```
            break

    # Получаем все маски для классов
    masks = results.masks.data.cpu().numpy()
    classes = results.boxes.cls.cpu().numpy()

    # Получаем список масок для нужного класса
    selected_masks = [masks[i] for i in range(len(classes)) if int(classes[i]) == class_id]

    return selected_masks


def get_cells_with_mask(self, cells_with_object: list, coords_cells: list, mask: np.ndarray, grid_size: tuple, threshold: float) -> list:
    '''Метод для получения ячеек таблицы, содержащих объект'''
    # Определяем размер ячейки
    cell_height, cell_width = int(mask.shape[0] / grid_size[0]), int(mask.shape[1] / grid_size[1])
    idx_cell = 0

    for i in range(grid_size[0]):
        for j in range(grid_size[1]):
            # Координаты прямоугольника, соответствующего ячейке
            y1, y2 = i * cell_height, (i + 1) * cell_height
            x1, x2 = j * cell_width, (j + 1) * cell_width

            # Вырезаем часть маски, соответствующую ячейке
            cell_mask = mask[y1:y2, x1:x2]
            # Рассчитываем какую часть ячейки занимает объект
            coverage_area = np.sum(cell_mask) / cell_mask.size

            # Проверяем, есть ли объект в ячейке
            if coverage_area >= threshold:
                # Сохраняем данные о ячейке
                cells_with_object.append(idx_cell)
                coords_cells.append((i, j))

            idx_cell += 1

    return cells_with_object, coords_cells


if __name__ == "__main__":
    # Загружаем модель
    model = YOLO('best.pt')

    # Целевой сайт
    target_link = 'https://rucaptcha.com/demo/recaptcha-v2'

    # Настройки user agent
    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"
    options = webdriver.ChromeOptions()

    options.add_experimental_option("excludeSwitches", ["enable-automation"])
    options.add_experimental_option('useAutomationExtension', False)
    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         # Полчаем 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