Министерство образования и молодежной политики Свердловской области
ГАПОУ СО «Уральский государственный колледж имени И.И. Ползунова»

КП. 09.02.07.12. ПЗ

РАЗРАБОТКА МОДУЛЯ РАСПОЗНОВАНИЯ ТЕКСТА

Пояснительная записка

Руководитель
_____/Е.Д.Ярочкина/

Разработал
_____/В.М.Кириллов/

Екатеринбург 2025

Министерство образования и молодежной политики Свердловской области

ГАПОУ СО «Уральский государственный колледж имени И.И. Ползунова»

УТВЕРЖДАЮ:
Заведующий кафедрой
Информационных систем и
автоматизации управления
_____/Ярочкина Е.Д./
«____» _____2025 г.

ЗАДАНИЕ
для курсового проектирования

по МДК.01.01 «Разработка программных модулей»
студенту Кириллову Вадиму Максимовичу
Группа ИП-402 шифр специальности 09.02.07 – «Информационные системы и программирование»
Тема задания «Разработка модуля распознавания текста»
Содержание проекта:

Руководитель проекта _____/Ярочкина Е.Д./

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель курсового проекта – разработка программного модуля для распознавания текста на изображениях и PDF-файлах в браузере.

Для достижения поставленной цели необходимо решить следующие задачи:

Проанализировать существующие методы и инструменты для распознавания текста.

Выбрать оптимальные технологии для реализации проекта. В качестве основных инструментов были выбраны библиотека компьютерного зрения OpenCV [15]для предварительной обработки изображений и OCR-движок Tesseract для непосредственного распознавания текста.

Разработать программный модуль на языке Python, реализующий функционал распознавания.

Провести тестирование готового программного продукта и оценить его эффективность.

Актуальность данной работы обусловлена тем, что в современном мире объёмы неструктурированной информации, такой как изображения и сканы документов, растут с невероятной скоростью. Ручная обработка этих данных отнимает много времени и сил, в связи с чем технологии оптического распознавания символов (OCR) играют ключевую роль в цифровой трансформации, позволяя превращать графические данные в редактируемый и индексируемый текст.

Разрабатываемый модуль может найти широкое применение в различных сферах: от помощи студентам в оцифровке конспектов до автоматизации ввода данных из документов в корпоративных системах. Новизна проекта заключается в создании простого и доступного инструмента, который не требует от пользователя установки специализированного программного обеспечения.

1 Теоретическая часть

1.1 Анализ существующих решений для распознавания текста

Распознавание текста (OCR — Optical Character Recognition [1]) является одной из ключевых технологий, позволяющих преобразовывать изображения с текстовой информацией в машиночитаемый формат. Такие решения широко применяются при оцифровке документов, автоматизации ввода данных, работе с архивами, формами и различными графическими носителями. Для выбора подходящих инструментов разработки были рассмотрены существующие OCR-решения, их возможности, преимущества и ограничения. Анализ включает как открытые, так и коммерческие инструменты, а также технологии на различных языках программирования.

Обзор существующих решений

Существующие OCR-системы можно разделить на несколько категорий: открытые библиотеки, коммерческие SDK, облачные сервисы и обёртки для популярных языков. Ниже рассмотрены наиболее распространённые решения с точки зрения функциональности и возможности интеграции в практическую разработку.

Решения на Python

Python является одним из наиболее удобных языков разработки OCR-систем благодаря широкому набору готовых библиотек.

Tesseract OCR [2]

Открытый OCR-движок, поддерживающий более 100 языков, включая русский. Является одним из наиболее распространённых инструментов.

Преимущества: бесплатность, активное развитие, высокая точность при корректной предобработке, удобная интеграция через pytesseract [21][22].

Недостатки: понижение точности на изображениях низкого качества, ограниченная поддержка сложных макетов.

Использование в проекте: выбран в качестве основного движка распознавания текста.

EasyOCR [3]

OCR-библиотека на основе нейросетевых моделей.

Преимущества: простота использования, поддержка многих языков.

Недостатки: меньшая точность на сложных документах по сравнению с Tesseract.

Решения на C++

C++ применяется в OCR-системах, где важна производительность и низкоуровневый контроль.

Tesseract OCR [2] (нативная версия) [2]

Основная версия движка реализована на C++.

Преимущества: высокая скорость, открытый исходный код, универсальность.

Недостатки: сложнее в использовании без готовых обёрток.

GOCR [4]

Простая OCR-система для базового распознавания.

Преимущества: лёгкость, бесплатность.

Недостатки: значительно уступает современным решениям по точности и поддержке языков.

Решения на Java

Java ориентирована на корпоративные приложения и кроссплатформенные системы.

Tess4J [5]

Обёртка для работы с Tesseract в Java-приложениях.

Преимущества: удобная интеграция, поддержка основных форматов.

Недостатки: зависимость от нативной версии Tesseract.

Коммерческие SDK (Asprise, LEADTOOLS)

Преимущества: высокая точность, расширенные возможности.

Недостатки: платные лицензии.

Решения на C# (.NET)

IronOCR [6]

Коммерческая библиотека с высокой точностью.

Преимущества: удобство интеграции, поддержка PDF и изображений.

Недостатки: требуется лицензия.

Tesseract.NET [7]

Обёртка для Tesseract под .NET.

Преимущества: бесплатность, поддержка русского языка.

Недостатки: необходимость установки движка.

OCR в JavaScript

JavaScript широко используется в веб-разработке, включая браузерные решения.

Tesseract.js [8]

Порт движка Tesseract для браузера и Node.js.

Преимущества: возможность выполнения OCR на стороне клиента.

Недостатки: меньшая скорость и точность по сравнению с нативными версиями.

Ocrad.js [9]

Подходит для простых задач.

Недостатки: ограниченная точность и функциональность.

Облачные сервисы

OCR [1] также реализован в виде REST API [13], независимых от языка программирования:

- Google Cloud Vision;
- Microsoft Azure Cognitive Services;
- OCR.Space;
- ABBYY Cloud OCR SDK.

Преимущества: высокая точность, поддержка сложных документов, минимум настройки.

Недостатки: требуется доступ в интернет, возможна стоимость использования.

Сравнительный анализ

При выборе OCR-решения учитывались следующие критерии:

– точность распознавания;

– стоимость использования;

– простота интеграции;

– поддержка русского языка;

– работа в offline-режиме.

На основе анализа:

– Tesseract OCR [2] является наиболее универсальным и бесплатным вариантом;

– обладает хорошей точностью при корректной предобработке;

– имеет широкую поддержку обёрток, включая Python;

– работает локально, без обращения к внешним сервисам.

Поэтому для реализации модуля в курсовом проекте выбран Tesseract OCR [2] в сочетании с библиотекой Python pytesseract [21][22]. Данный выбор обеспечивает надёжность, доступность и удобство интеграции с модулем предобработки изображений.

1.2 Структура разрабатываемого модуля

В рамках данного курсового проекта модуль распознавания текста разрабатывается как самостоятельный программный компонент на языке Python [10], предназначенный для обработки изображений и PDF-файлов с целью извлечения текста. Модуль интегрируется в веб-приложение на основе Vue.js [11] (клиентская часть) и Go [12] (серверная часть) через REST API [13], обеспечивая бесшовное взаимодействие. Архитектура построена на принципах модульного

программирования, что гарантирует гибкость, масштабируемость, удобство тестирования и возможность повторного использования компонентов. Общая структура модуля соответствует стандартному конвейеру OCR-систем: данные последовательно проходят этапы ввода, предобработки, распознавания и вывода (см. Рисунок 1 — Общая структура приложения).

Рисунок 1 иллюстрирует иерархическую блок-схему модуля, где каждый компонент представлен как узел с подфункциями, подчеркивая поток данных от ввода к выводу. Это позволяет визуально оценить зависимости и интеграцию с внешними системами, такими как веб-интерфейс.

Общая архитектура модуля Модуль реализован как серверный компонент на Python [10], интегрируемый в Go [11] для обработки запросов через REST API [13]. Он состоит из четырех основных подмодулей, каждый из которых выполняет специализированные функции. Это обеспечивает разделение ответственности и упрощает отладку. Для иллюстрации общей структуры см. Рисунок 1, где показаны связи между компонентами.

Основные компоненты модуля:

Модуль ввода данных

Отвечает за прием и начальную валидацию входных данных из веб-приложения (изображения в форматах JPG, PNG или PDF-файлы). Этот модуль служит "воротами" системы, предотвращая обработку некорректных данных. Функции:

– Проверка корректности форматов файлов (JPG, PNG, PDF) с использованием встроенных библиотек Python;

– конвертация PDF в список изображений для многостраничных документов с помощью pdf2image, что позволяет обрабатывать сложные файлы поэтапно;

– обработка ошибок (например, поврежденные файлы или превышение размера) с генерацией исключений и логгированием для отладки. Технологии: Python [10] стандартная библиотека, pdf2image [14]. Интеграция: Данные принимаются через POST-запросы REST API [13], что связывает этот модуль с

клиентской частью (Vue.js [11]). В случае ошибок возвращается JSON [23]-ответ с кодом ошибки;

Модуль предобработки изображений

Ключевой этап для повышения качества распознавания, особенно при работе с изображениями низкого разрешения или с артефактами (шум, искажения). Этот модуль напрямую влияет на точность Tesseract OCR [2], как указано в анализе решений (см. 1.1). Функции:

– Преобразование изображения в оттенки серого для упрощения анализа;

– фильтрация шума с использованием гауссовского размытия или медианного фильтра для сохранения краев текста;

– бинаризация изображения для выделения текста от фона с адаптивными методами;

– масштабирование и коррекция перспективы для устранения искажений, возникающих при сканировании;

– нормализация освещения для компенсации теней и неравномерного фона. Технологии: OpenCV [15] (cv2). Интеграция: Результаты передаются в модуль распознавания как оптимизированные изображения; для тестирования предусмотрены unit-тесты на отдельных функциях.

Модуль распознавания текста

Основной компонент, выполняющий извлечение текста с использованием OCR-движка. Он опирается на пред обработанные данные и поддерживает кастомизацию для повышения точности. Функции:

– Применение Tesseract OCR [2] к изображениям через pytesseract [21][22].image_to_string;

– настройка параметров Tesseract OCR [2];

– поддержка русского языка через установку языковой модели;

Технологии: pytesseract [21][22], Tesseract OCR [2]. Интеграция: работает в связке с предобработкой; ошибки распознавания логгируются для анализа в разделе тестирования.

Модуль вывода результатов

Формирует и передает результаты в удобном формате, завершая конвейер. Этот модуль обеспечивает совместимость с веб-приложением. Функции:

– Форматирование распознанного текста в JSON [23] для передачи через REST API [13];

– сохранение результатов в текстовый файл (.txt) при необходимости, с опцией экспорта в другие форматы;

– предоставление метаданных для анализа качества (например, уровень уверенности Tesseract, время обработки).

Технологии: Python [10] (json модуль, стандартная библиотека). Интеграция: Результаты возвращаются в ответе REST API [13], что позволяет отобразить их в интерфейсе Vue.js [11]; поддерживается асинхронная обработка для больших файлов.

Общая структура модуля представлена на Рисунок 1, где визуализированы потоки данных и зависимости. Это обеспечивает эффективную интеграцию в веб-приложение, где пользователь загружает файл через интерфейс, а модуль обрабатывает его на сервере. Для дальнейшей реализации см. практическую часть, где описана кодовая база и тестирование.

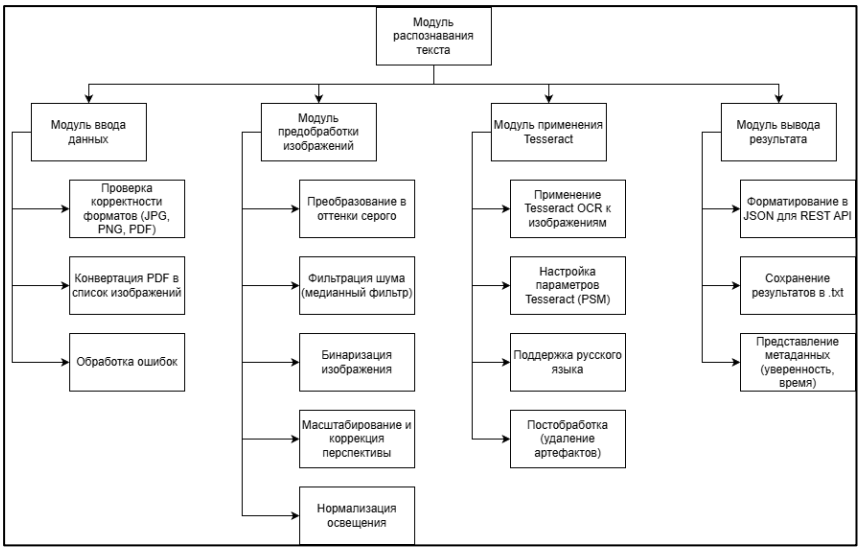

Рисунок 1 – Структура модуля

1.3 Обзор технологий оптического распознавания текста

Оптическое распознавание символов (OCR, Optical Character Recognition) [1] представляет собой технологию, которая преобразует изображения, содержащие текст, в машиночитаемый текстовый формат. Эта технология играет ключевую роль в автоматизации обработки неструктурированных данных, таких как сканированные документы, фотографии и PDF-файлы, что особенно актуально для веб-приложений, где требуется быстрая и точная оцифровка информации. В рамках разработки модуля распознавания текста для веб-приложения был проведен детальный анализ доступных технологий, с учетом их функциональности, производительности и совместимости с серверной и клиентской инфраструктурой.

Основной инструмент: Tesseract OCR [2]

Центральным компонентом модуля является библиотека Tesseract OCR [2] — мощное открытое решение, изначально разработанное компанией Hewlett-Packard и впоследствии существенно доработанное Google. Tesseract поддерживает более 100 языков, включая русский, что делает её подходящей для работы с многоязычными данными. Интеграция с Python [10] осуществляется через обертку pytesseract [21][22], что позволяет внедрить её в серверную часть веб-приложения, построенного на Go [12]. Среди ключевых преимуществ Tesseract OCR [2] можно выделить:

– Бесплатность и открытый исходный код, что исключает затраты на лицензии и упрощает модификацию под конкретные задачи;

– высокая точность распознавания, особенно при использовании предобработанных изображений, где текст четко выделен на однородном фоне.

– поддержка различных форматов, включая JPG, PNG и PDF (посредством предварительной конвертации);

– гибкость настройки, позволяющая адаптировать параметры распознавания.

Однако Tesseract OCR [2] имеет ограничения, которые необходимо учитывать. Она требует тщательной предобработки изображений для достижения оптимальных результатов, особенно если исходные данные имеют низкое разрешение, шумы или сложные макеты (например, таблицы или графики). Без предварительной обработки точность может снижаться, что делает интеграцию с инструментами предобработки обязательной.

Предобработка изображений [16]: OpenCV [15]

Для повышения качества входных данных используется библиотека OpenCV [15] (Open Source Computer Vision Library), широко известная в области компьютерного зрения. OpenCV [15] предоставляет набор инструментов для обработки изображений, которые напрямую влияют на эффективность работы Tesseract. Основные этапы предобработки включают:

– Преобразование в оттенки серого — упрощает последующую обработку, выделяя текст на однородном фоне;

– фильтрация шума — применение гауссовского размытия или медианного фильтра для устранения мелких артефактов, сохраняя при этом структуру текста;

– бинаризация — преобразование изображения в черно-белый формат с использованием методов, таких как адаптивная пороговая обработка, для четкого разделения текста и фона;

– масштабирование и коррекция перспективы — устранение искажений, возникающих при сканировании или фотографировании под углом, с помощью алгоритмов трансформации;

– нормализация освещения — компенсация теней и неравномерного фона для улучшения контрастности текста.

Эти шаги критически важны для модуля, так как качество предобработки напрямую определяет точность распознавания, особенно при работе с изображениями низкого качества или сканированными документами с физическими дефектами.

Обработка PDF-файлов

Для расширения функциональности модуля и поддержки многостраничных документов используется библиотека pdf2image. Этот инструмент конвертирует страницы PDF-файлов в отдельные изображения, которые затем передаются на этапы предобработки и распознавания. Преимущества применения pdf2image включают:

– Поддержка многостраничных документов, что делает модуль универсальным для работы с различными типами входных данных;

– интеграция с существующим конвейером, так как результирующие изображения легко обрабатываются OpenCV [15]и Tesseract OCR [2];

– гибкость настроек, позволяющая управлять разрешением и качеством конвертации.

Интеграция с веб-приложением

Модуль распознавания текста интегрируется в веб-приложение через REST API [13], реализованное с использованием Go [12]. Это обеспечивает взаимодействие между клиентской частью и серверной логикой. Результаты распознавания форматируются в JSON [23], что упрощает передачу данных и их отображение на веб-странице. Такой подход позволяет:

– Обеспечить масштабируемость и совместимость с различными клиентскими платформами;

– реализовать асинхронную обработку запросов, что важно при работе с большими файлами или сложными изображениями;

1.4 Обзор языков программирования и библиотек

Выбор технологического стека является ключевым этапом при разработке модуля распознавания текста, так как он напрямую влияет на производительность, точность и сложность интеграции. Для реализации данного курсового проекта был выбран язык программирования Python в связке с библиотекой Tesseract OCR [2] (через оболочку pytesseract [21][22]) и OpenCV [15]. Эта комбинация представляет собой оптимальное решение, поскольку сочетает в себе бесплатность, высокую

точность распознавания, в том числе и для русского языка, и широкие возможности для предварительной обработки изображений.

Для обоснования этого выбора был проведен анализ альтернативных решений на других языках программирования, которые также широко применяются для создания OCR-систем.

Высокопроизводительные решения на C++

C++ часто используется для разработки OCR-движков, где критически важны скорость и эффективность. Примеры:

– Нативная версия Tesseract OCR [2]. Ядро Tesseract написано на C++, что обеспечивает максимальную производительность. Однако его прямое использование требует более глубоких знаний и сложнее в интеграции по сравнению с использованием оболочек;

– GOCR (Gnu Optical Character Recognition) [4]. Простая и бесплатная библиотека, которая, тем не менее, уступает современным решениям в точности и функциональности.

Корпоративные решения на Java и C# (.NET)

Эти языки популярны в корпоративной-разработке благодаря своей кроссплатформенности и надежности. Примеры:

– Оболочки для Tesseract. Для интеграции движка существуют обертки, такие как Tess4J [5] для Java и Tesseract.NET [7] для C#. Они позволяют использовать бесплатный и мощный инструмент в экосистеме этих языков;

– коммерческие SDK. Решения, такие как Asprise OCR, LEADTOOLS (Java) и IronOCR [6] (C#), предлагают высокую точность и расширенную функциональность, но требуют приобретения платной лицензии, что делает их менее подходящими для данного проекта.

Клиентские решения на JavaScript

JavaScript позволяет создавать OCR-инструменты, работающие непосредственно в браузере, что снижает нагрузку на сервер. Примеры:

– Tesseract.js [8]. Является портом Tesseract, который выполняет распознавание на стороне клиента. Основным его ограничением является более низкая производительность по сравнению с нативными реализациями;

– Ocrad.js [9]. Легковесная библиотека с ограниченной точностью, подходящая для решения базовых задач.

Таким образом, несмотря на наличие мощных альтернатив для конкретных задач (высокопроизводительные вычисления на C++ или клиентская обработка на JavaScript), связка Python [10], Tesseract OCR [2] и OpenCV [15]была выбрана как наиболее сбалансированное, доступное и функциональное решение для целей курсового проекта.

1.5 Обзор библиотек Python для обработки изображений

Качественная предварительная обработка изображений является критически важным этапом для достижения высокой точности оптического распознавания символов. Python предлагает мощную экосистему библиотек для работы с графическими данными, позволяя эффективно подготавливать их перед отправкой в OCR-движок. В данном проекте для полного цикла обработки — от конвертации PDF до улучшения качества изображений — используются две ключевые библиотеки.

OpenCV [15]: фундамент предобработки

OpenCV [15] (Open Source Computer Vision Library) — это одна из самых мощных и широко используемых библиотек с открытым исходным кодом в области компьютерного зрения. Она предоставляет исчерпывающий набор инструментов, которые напрямую влияют на эффективность работы Tesseract, подготавливая изображения для максимально точного распознавания. В модуле реализован конвейер предобработки, включающий следующие ключевые операции:

– Преобразование в оттенки серого. Это первый шаг, который упрощает дальнейший анализ, удаляя информацию о цвете и позволяя алгоритмам сосредоточиться на яркости и контрасте пикселей;

– фильтрация шума. Для устранения мелких артефактов, таких как случайные точки или зернистость, применяются методы размытия, например, гауссовский или медианный фильтры. Это помогает сгладить фон, сохраняя при этом четкость краев символов;

– бинаризация. Изображение преобразуется в черно-белый формат, где текст четко отделяется от фона. В проекте используются адаптивные методы пороговой обработки, которые особенно эффективны при работе с изображениями с неравномерным освещением или тенями;

– масштабирование и коррекция перспективы. Эти операции устраняют геометрические искажения, которые часто возникают при сканировании или фотографировании документов под углом. Алгоритмы трансформации выравнивают изображение, приводя текст в удобный для распознавания вид;

– нормализация освещения. Данный шаг компенсирует тени и неравномерный фон, улучшая общую контрастность текста и делая его более читаемым для OCR-движка.

Каждый из этих шагов критически важен, поскольку качество предобработки напрямую определяет итоговую точность распознавания, особенно при работе со сканами низкого качества или фотографиями документов.

pdf2image: Работа с PDF-документами

Для расширения функциональности модуля и обеспечения поддержки многостраничных документов была интегрирована библиотека pdf2image. Она служит удобной оболочкой для утилиты poppler, позволяя конвертировать страницы PDF-файлов в отдельные изображения. Это дает несколько ключевых преимуществ:

– Универсальность: модуль становится способен обрабатывать не только стандартные форматы изображений (JPG, PNG), но и многостраничные PDF-документы, что значительно расширяет его область применения;

– бесшовная интеграция: полученные в результате конвертации изображения легко передаются в существующий конвейер обработки на базе

OpenCV [15] и Tesseract OCR [2], что позволяет применять к ним все те же шаги предобработки;

– гибкость: библиотека позволяет настраивать параметры конвертации, такие как разрешение (DPI) и качество итоговых изображений, что дает возможность находить баланс между точностью и скоростью обработки.

1.6 Обзор системы Tesseract OCR [2] и её возможностей

Центральным элементом разрабатываемого модуля является Tesseract OCR [2] — одна из самых мощных и популярных систем оптического распознавания символов с открытым исходным кодом. Изначально разработанная компанией Hewlett-Packard и в настоящее время поддерживаемая Google, Tesseract превратилась в зрелое и надежное решение для извлечения текста из изображений. Основной движок системы написан на языке C++ и для достижения высокой точности распознавания использует современные нейронные сети на основе LSTM (Long Short-Term Memory).

Ключевые возможности и преимущества:

– Мультиязычная поддержка. Одним из главных преимуществ Tesseract OCR [2] является его способность распознавать текст на более чем 100 языках, включая русский. Это делает его универсальным инструментом для обработки документов различного происхождения. В рамках проекта поддержка русского языка реализуется путем установки соответствующей языковой модели;

– бесплатность и открытый исходный код. Tesseract OCR [2] распространяется под свободной лицензией, что полностью исключает затраты на приобретение программного обеспечения. Открытый исходный код также предоставляет возможность для модификации и адаптации системы под специфические нужды проекта;

– простая интеграция с Python [10]. Для использования в Python-приложениях существует удобная библиотека-обертка pytesseract [21][22]. Она предоставляет простой интерфейс для взаимодействия с движком Tesseract OCR

[2], позволяя извлекать текст из изображений с помощью вызова всего одной функции, такой как image_to_string;

– гибкость настройки. Tesseract OCR [2] предоставляет разработчику контроль над процессом распознавания через различные параметры конфигурации. Например, можно изменять режим сегментации страницы (Page Segmentation Mode, PSM) [17] для оптимизации работы с текстом разной структуры (один блок, одна строка и т.д.) или выбирать движок распознавания (OCR Engine Mode, OEM) [17] для использования различных моделей;

Ограничения и решающая роль предобработки

Несмотря на все свои преимущества, Tesseract не является универсальным решением, работающим идеально «из коробки». Эффективность системы напрямую зависит от качества входных изображений. Для изображений с низким разрешением, наличием визуальных шумов, артефактов, сложной структурой макета (например, таблицы или графики) или неравномерным освещением, точность распознавания может значительно снижаться.

Именно поэтому тщательная предобработка изображений является не просто желательным, а обязательным этапом для достижения оптимальных результатов. Использование библиотек, таких как OpenCV [15], для выполнения операций фильтрации, бинаризации, коррекции перспективы и нормализации освещения позволяет подготовить изображение таким образом, чтобы Tesseract OCR [2] смог извлечь текст с максимальной точностью.

В конечном счете, Tesseract OCR [2] был выбран для данного проекта как основной OCR-движок благодаря его оптимальному сочетанию бесплатности, высокой точности (при условии качественной предобработки) и отличной поддержке русского языка.

2 Практическая часть

2.1 Разработка модуля ввода данных

Модуль ввода данных является первым компонентом общего программного пайплайна и отвечает за получение входных файлов, их валидацию и подготовку к дальнейшей обработке. Его основной задачей является загрузка изображений и PDF-документов, приведение их к единому формату и передача на этап предобработки. Реализация модуля размещена в отдельном файле loader.py.

Модуль поддерживает два основных типа входных данных: изображения (JPG, PNG) и PDF-документы. При обработке изображений файл открывается с использованием библиотеки PIL и сразу конвертируется в формат RGB, что обеспечивает единый формат данных для последующих этапов.

При работе с PDF-документами выполняется конвертация каждой страницы в отдельное изображение с помощью библиотеки pdf2image. В результате модуль формирует список объектов типа PIL.Image, где каждый элемент представляет собой одну страницу документа. Такой подход позволяет обрабатывать многостраничные PDF, сохраняя при этом единообразие формата данных и совместимость с остальными компонентами системы.

Для всех типов файлов реализована проверка корректности входных данных. В случае ошибок чтения, неподдерживаемого формата или повреждённого файла генерируется исключение с описанием проблемы, которое передаётся в главный модуль для дальнейшей обработки.

Результатом работы модуля является единая структура данных: перечень изображений в формате RGB. Для обычного изображения этот перечень содержит один элемент, для PDF — набор изображений по количеству страниц. Такая схема позволяет независимо от формата входного файла передавать данные на этап предобработки, обеспечивая гибкость и расширяемость всей системы. Код модуля представлен в листинге 1.

```python
from pdf2image import convert_from_path
from PIL import Image

def load_image(file_path):
    """
    Загружает файл (изображение или PDF) и возвращает:
    - объект PIL Image для обычного изображения;
    - список объектов PIL Image для многостраничного PDF.
    Всегда конвертирует в 'RGB', чтобы обеспечить единообразие.
    """
    image = None

    try:
        if file_path.lower().endswith('.pdf'):
            # Конвертируем все страницы PDF в изображения
            pages = convert_from_path(file_path)
            if pages:
                # Приводим каждую страницу к формату RGB
                image = [page.convert('RGB') for page in pages]
        else:
            # Открываем файл изображения
            image = Image.open(file_path).convert('RGB')

    except Exception as e:
        # Эта ошибка будет поймана в main.py
        raise ValueError(f"Ошибка при загрузке файла: {e}")

    if image is None:
        raise ValueError("Не удалось обработать файл. Убедитесь, что это изображение или PDF-документ.")

    return image
```

Листинг 1.

2.2 Разработка модуля предобработки изображений

Как было установлено в теоретической части, эффективность системы Tesseract OCR [2] напрямую зависит от качества входных данных. Изображения низкого разрешения, с артефактами или неравномерным освещением значительно снижают точность распознавания. Для решения этой проблемы был разработан модуль предобработки, являющийся вторым ключевым компонентом общего программного пайплайна.

Целью данного модуля является программная «очистка» и «улучшение» изображений, полученных от «Модуля ввода данных», и приведение их к виду, оптимальному для анализа нейросетью Tesseract. В качестве технологической основы модуля используется библиотека OpenCV [15](cv2), что соответствует анализу, приведенному в разделе 1.5.

Модуль реализован в отдельном файле preprocessing.py и представляет собой конвейер (pipeline) последовательных преобразований. Основная логика инкапсулирована в функции preprocess_image, которая принимает на вход объект изображения и последовательно выполняет следующие шаги:

а. Конвертация в формат OpenCV. Изначально изображение загружается в формате RGB. OpenCV [15]по умолчанию работает с форматом BGR. Первым шагом является конвертация изображения в массив NumPy и смена цветовых каналов. Код блока представлен в листинге 2:

```
# 1. Конвертация из PIL в CV2 (OpenCV)
img_cv = cv2.cvtColor(np.array(image), cv2.COLOR_RGB2BGR)
```

Листинг 2.

б. преобразование в оттенки серого (Grayscaling). Удаление информации о цвете является обязательным шагом, так как упрощает дальнейший анализ. Алгоритм бинаризации (следующий шаг) работает с одним 8-битным каналом вместо трех. Код блока представлен в листинге 3.

```
# 2. Конвертация в оттенки серого
gray = cv2.cvtColor(img_cv, cv2.COLOR_BGR2GRAY)
```

Листинг 2.

в. масштабирование (Scaling). Tesseract был обучен на изображениях с разрешением около 300 DPI. Если текст на изображении слишком мелкий, нейросеть не может его «увидеть». Алгоритм проверяет ширину изображения и, если она меньше порогового значения (1000 пикселей), пропорционально увеличивает его, используя кубическую интерполяцию для сохранения четкости. Код блока представлен в листинге 4.

```
# 3. Масштабирование
h, w = gray.shape
if w < 1000:
```

```
    scale = 1.5
    gray = cv2.resize(gray, (int(w * scale), int(h * scale)),
interpolation=cv2.INTER_CUBIC)
```

Листинг 4.

г. фильтрация шума (Noise Filtering). Для устранения артефактов "соль и перец" применяется медианный фильтр. Он заменяет каждый пиксель медианным значением его соседей (с ядром 3x3), что эффективно удаляет шумы, сохраняя при этом резкость границ символов. Код блока представлен в листинге 5.

```
# 4. Удаление шума
denoised = cv2.medianBlur(gray, 3)
```

Листинг 5.

д. бинаризация (Binarization). Это ключевой этап, на котором изображение преобразуется в строго черно-белое. Для этого используется адаптивный метод — пороговая обработка Оцу (THRESH_OTSU). Его преимущество в том, что он не использует фиксированное значение, а автоматически вычисляет оптимальный порог для всего изображения. Это критически важно для документов с неравномерным освещением или тенями. Код блока представлен в листинге 6.

```
# 5. Бинаризация (Порог Оцу)
_, binary_image = cv2.threshold(denoised, 0, 255, cv2.THRESH_BINARY |
cv2.THRESH_OTSU)
```

Листинг 6.

е. коррекция перспективы (Deskewing). Tesseract крайне чувствителен к наклону текста. Для этого применяется алгоритм выравнивания (deskew): на бинарном изображении ищутся все текстовые контуры, вычисляется их общий угол наклона, после чего изображение поворачивается на этот угол в обратную сторону с помощью аффинного преобразования (cv2.warpAffine), делая строки текста горизонтальными. Код блока представлен в листинге 7:

```
def deskew(binary_image):
    try:
        23ords = np.column_stack(np.where(binary_image > 0))
        angle = cv2.minAreaRect(23ords)[-1]

        if angle < -45:
            angle = -(90 + angle)
        else:
            angle = -angle
```

```
        (h, w) = binary_image.shape[:2]
        center = (w // 2, h // 2)
        M = cv2.getRotationMatrix2D(center, angle, 1.0)

        rotated = cv2.warpAffine(binary_image, M, (w, h),
                                 flags=cv2.INTER_CUBIC,
                                 borderMode=cv2.BORDER_CONSTANT,
                                 borderValue=0)

        return rotated

    except Exception as e:
        return binary_image
```

Листинг 7.

В результате работы этого алгоритма, «Модуль применения Tesseract» получает на вход идеально подготовленное, чистое, черно-белое и выровненное изображение. Пример работы алгоритма предобработки показан на рисунке 2.

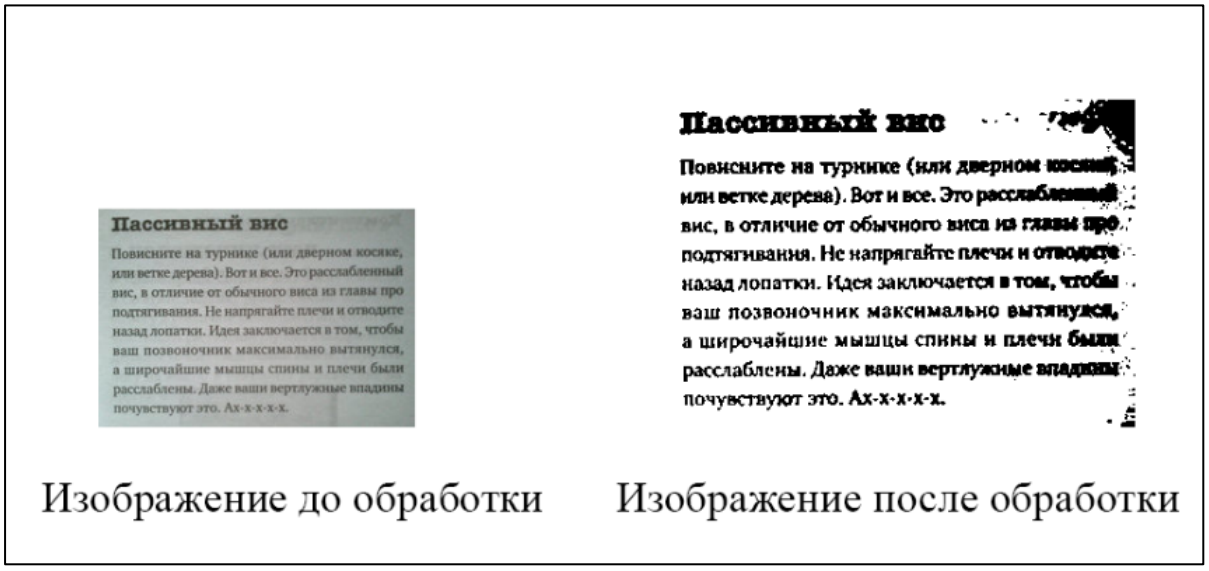

Рисунок 2 – Пример работы модуля

2.3 Интеграция библиотеки Tesseract OCR [2] для распознавания текста

Для использования OCR-движка Tesseract в проекте требуется выполнить два основных этапа установки: установка нативного приложения Tesseract OCR [2] на операционную систему и установка Python-библиотеки pytesseract [21][22] через

менеджер пакетов pip. Данный подход обеспечивает корректную работу связки Python-кода с низкоуровневым движком, написанным на C++.

Установка Tesseract OCR [2] на операционную систему:

Tesseract OCR [2] является кроссплатформенным приложением, и процесс его установки варьируется в зависимости от используемой операционной системы. Для Windows рекомендуется скачать официальный установщик с портала GitHub, соответствующий разрядности системы (32 или 64 бита). Установка через установщик автоматически добавляет Tesseract в переменную среды PATH, что является обязательным условием для работы библиотеки pytesseract [21][22].

Установка Python-библиотеки pytesseract [21][22]:

Библиотека pytesseract [21][22] представляет собой обертку для Tesseract OCR [2], позволяющую взаимодействовать с движком напрямую из Python-кода. Установка выполняется через менеджер пакетов pip(представлена в листинге 7).

```
pip install pytesseract [21][22]
```

Листинг 8.

Данная команда загружает и устанавливает последнюю стабильную версию библиотеки из репозитория PyPI. После установки необходимо убедиться в отсутствии ошибок импорта в Python-интерпретаторе (Представлено в листинге 8)

```
import pytesseract [21][22]
```

Листинг 9.

После правильной установки приложения Tesseract OCR [2] и библиотеки pytesseract [21][22] все предварительные шаги для корректной работы модуля выполнены и можно переходить к разработке самого модуля для распознавания текста.

2.4 Разработка модуля распознавания в среде Python

Модуль распознавания текста является третьим компонентом общего программного конвейера и отвечает за извлечение текстовой информации из предварительно обработанного изображения. Его функциональная часть

реализована в отдельном файле ocr.py . Модуль использует библиотеку pytesseract [21][22], обеспечивающую взаимодействие с OCR-движком Tesseract.

Целью данного этапа является преобразование бинарного изображения в текстовый формат и формирование набора мета-данных, позволяющих оценивать качество распознавания. В модуле определена основная функция get_text_from_image, выполняющая полный цикл обработки.

Работа функции включает следующие основные шаги.

Во-первых, задаются параметры конфигурации Tesseract. В проекте используется язык распознавания rus, движок OEM 3 и режим сегментации страницы PSM 6, который предназначен для обработки блоков текста. Эти параметры обеспечивают стабильную работу алгоритма и оптимальную точность при распознавании документов. Реализация блока кода представлена в листинге 9.

```
# --psm 6, работает стабильнее, чем 3
custom_config = r'-l rus --oem 3 --psm 6'
```

Листинг 10.

Во-вторых, для извлечения текста применяется метод image_to_data, возвращающий расширенную структуру данных. В отличие от стандартного способа распознавания, данный метод предоставляет дополнительную информацию: координаты слов, их индексы и уровни уверенности. Это позволяет вести дальнейший анализ качества результата, а также исключать элементы с некорректными значениями. Реализация блока кода представлена в листинге 10.

```
# Используем image_to_data для получения детальной информации
data = pytesseract [21][22].image_to_data(processed_image, config=custom_config,
output_type=Output.DICT)
```

Листинг 11.

На следующем этапе из полученного словаря извлекаются слова и соответствующие confident-значения. Из обработки исключаются элементы, у которых уровень уверенности равен –1, а также пустые строки. Такая фильтрация необходима для формирования корректного результата и удаления шумовых

элементов, возникающих при работе OCR-движка. Реализация блока кода представлена в листинге 11.

```
full_text = []
confidences = []

for i in range(len(data['text'])):
    if int(data['conf'][i]) > -1:
        word = data['text'][i]
        if word.strip():
            full_text.append(word)
            confidences.append(float(data['conf'][i]))
```

Листинг 12.

Завершая работу, функция формирует итоговую строку текста и список confident-значений, которые передаются в основной модуль для вычисления среднего уровня уверенности и формирования итогового ответа. Возврат данных в таком виде обеспечивает гибкость и удобство интеграции в серверную часть приложения. Реализация блока кода представлена в листинге 12.

```
return " ".join(full_text), confidences
```

Листинг 13.

В целом модуль ocr.py представляет собой самостоятельный компонент, обеспечивающий выполнение операции распознавания текста на основе подготовленных изображений. Реализация модуля позволяет последовательно извлекать текстовую информацию и формировать данные о мета-значениях, что делает возможной дальнейшую обработку результата в составе общего программного конвейера.

2.5 Разработка модуля вывода данных

Модуль вывода данных завершает работу общего программного конвейера и отвечает за формирование итогового результата распознавания в структурированном виде. Его назначение заключается в преобразовании полученного текста, мета-значений уверенности и служебной информации о времени обработки в единый JSON [23]-объект, который передаётся в серверную часть веб-приложения. Реализация модуля размещена в составе файла main.py и представлена функцией export_data.

Модуль получает на вход несколько параметров: итоговую строку текста, совокупный список мета-значений уверенности OCR-движка и отметку времени начала обработки. На основе этих данных формируется единый объект результата, содержащий три ключевых элемента:

- распознанный текст;
- среднее значение уверенности по всем обработанным словам;
- время обработки в миллисекундах.

Среднее значение уверенности вычисляется по всем положительным значениям списка мета-данных, что позволяет учитывать только корректно распознанные элементы и исключать некорректные или нулевые показатели. Такой подход обеспечивает более объективную оценку качества распознавания текста.

Время обработки определяется как разница между текущим временем и моментом запуска основного модуля. Данный параметр включён для контроля производительности системы и может использоваться при последующей оптимизации или сравнении различных методов предобработки и распознавания.

Формирование результата выполняется в формате JSON [23], что обеспечивает совместимость модуля с серверной частью приложения и позволяет легко интегрировать его в REST API [13]. JSON [23]-формат сохраняет необходимую структуру данных, упрощает дальнейший разбор на стороне бэкенда и позволяет выводить результат в пользовательском интерфейсе без дополнительной обработки.

В случае возникновения ошибки модуль сформирует JSON [23]-ответ со структурой, содержащей описание исключения. Это гарантирует предсказуемое поведение системы и единообразный формат ответов при штатной и нештатной работе.

Таким образом, модуль вывода данных обеспечивает корректное завершение процесса распознавания и формирует итоговый результат в удобном и

стандартизированном формате, необходимом для интеграции в веб-приложение. Код модуля представлен в листинге 14.

```
# Модуль 4: Экспорт данных
def export_data(text, confidence_list, start_time):
    """Формирует и выводит итоговый JSON [23]."""
    end_time = time.time()
    processing_time = int((end_time - start_time) * 1000)  # в миллисекундах

    valid_confidences = [conf for conf in confidence_list if conf > 0]
    average_confidence = (
        sum(valid_confidences) / len(valid_confidences)
        if valid_confidences else 0
    )

    result = {
        "text": text,
        "average_confidence": round(average_confidence, 2),
        "processing_time_ms": processing_time
    }

    # ensure_ascii=True, чтобы избежать проблем с кодировкой в Go
    print(json.dumps(result, ensure_ascii=True))
```

Листинг 14.

2.6 Анализ полученных результатов и выводы

В процессе анализа результатов работы модуля была проведена оценка функционирования всех основных этапов: загрузки данных, предобработки изображений, распознавания текста и формирования итогового ответа. Проверка велась с использованием изображений и PDF-файлов различного качества и структуры, что позволило определить диапазон корректной работы модуля и установить его ограничения.

На этапе загрузки модуль корректно обрабатывал изображения и PDF-файлы, преобразуя их в единый формат для дальнейшей обработки. Ошибки возникали только в случаях некорректного формата входного файла либо повреждённых данных, что соответствует ожидаемому поведению загрузчика.

Результаты работы модуля предобработки показали, что применяемые операции — перевод в оттенки серого, бинаризация и масштабирование — позволяют формировать изображение, пригодное для дальнейшего анализа.

Предобработка улучшала качество контраста текста, однако влияние шума и неравномерного освещения всё равно оказывало влияние на конечный результат.

Особое внимание было уделено модулю распознавания текста. В ходе тестирования установлено, что модуль функционирует корректно при обработке изображений с линейной структурой текста. Это объясняется тем, что в текущей реализации используется режим сегментации страницы PSM 6, который предназначен для одноблочного текста без сложных элементов структуры. Такой подход обеспечивает стабильные результаты для строк или абзацев, но приводит к снижению точности при распознавании таблиц, многоступенчатых документов, многостолбцовых макетов и схематичных материалов. Данное ограничение является особенностью выбранного режима работы OCR и должно учитываться при дальнейшем применении модуля.

Формирование итогового JSON [23]-пакета происходило корректно. Все ключевые параметры — текст, среднее мета-значение и время обработки — вычислялись без отклонений. Структура данных соответствует требованиям для интеграции с серверной частью веб-приложения.

На основании анализа можно сделать вывод, что разработанный модуль обеспечивает корректное функционирование при работе с текстом простой структуры. При этом точность распознавания зависит от качества входных изображений и напрямую ограничена использованием одноблочного режима. Модуль может использоваться в составе веб-приложения, однако при необходимости поддержки документов сложной структуры потребуется расширение функциональности и настройка других режимов сегментации страницы.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового проекта было выполнено исследование технологий оптического распознавания текста, а также разработан программный модуль, предназначенный для автоматической обработки изображений и PDF-файлов с целью извлечения текстовой информации. Выполненный анализ существующих решений позволил обосновать выбор Python, OpenCV [15]и Tesseract OCR [2] как оптимального набора инструментов для решения поставленных задач.

В результате проведённого анализа были рассмотрены современные решения в области OCR, что позволило обосновать выбор связки Python, OpenCV [15]и Tesseract OCR [2] как оптимальной для поставленных задач.

В рамках практической части реализован полный программный конвейер, включающий загрузку данных, предобработку изображений, распознавание текста и формирование итогового результата. Модуль построен на принципах модульности, что обеспечивает удобство сопровождения и возможность дальнейшего развития. Учтены особенности работы Tesseract в режиме одноблочного текста, что определяет область корректной работы модуля.

По итогам тестирования установлено, что модуль корректно выполняет основные операции и обеспечивает стабильное распознавание текста в документах линейной структуры. Ограничения в точности при работе со сложными макетами объясняются используемым режимом сегментации страницы и могут быть устранены при дальнейшей доработке проекта.

Таким образом, цели и задачи курсового проекта выполнены. Разработанный модуль может быть использован в составе веб-приложения для автоматизированного получения текстовых данных и представляет основу для дальнейших улучшений, включая расширение поддержки многостраничных документов и повышение точности распознавания.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. OCR — статья «OCR — что такое». Режим доступа: https://skyeng.ru/magazine/wiki/it-industriya/chto-takoe-ocr/ (дата обращения: 2.12.2025).

2. Официальный репозиторий Tesseract OCR. Режим доступа: https://github.com/tesseract-ocr/tesseract (дата обращения: 2.12.2025).

3. Официальный репозиторий EasyOCR. Режим доступа: https://github.com/JaidedAI/EasyOCR (дата обращения: 2.12.2025).

4. Официальный сайт GOCR. Режим доступа: https://jocr.sourceforge.net/ (дата обращения: 2.12.2025).

5. Официальный сайт Tess4J. Режим доступа: https://tess4j.sourceforge.net/ (дата обращения: 2.12.2025).

6. Официальный сайт IronOCR на русском языке. Режим доступа: https://ironsoftware.com/csharp/ocr/languages/russian-ru/ (дата обращения 2.12.2025).

7. Официальный сайт Tesseract.Net. Режим доступа: https://tesseract.patagames.com/help/html/baa0aa10-7805-4ae6-b6e9-9df777c4678c.htm (дата обращения: 2.12.2025).

8. Официальный репозиторий Tesseract.js. Режим доступа: https://github.com/naptha/tesseract.js#tesseractjs (дата обращения: 2.12.2025).

9. Официальный сайт Ocrad.js. Режим доступа: https://antimatter15.com/ocrad.js/demo.html (дата обращения: 2.12.2025).

10. Официальный сайт Python. Режим доступа: https://www.python.org/ (дата обращения 2.12.2025).

11. Официальный сайт Vue.js. Режим доступа: https://ru.vuejs.org/ (дата обращения 2.12.2025).

12. Обучающий ресурс по языку программирования Go. Режим доступа: http://metanit.com/go/tutorial/1.1.php (дата обращения: 2.12.2025).

13. Серия статей о разработке REST API. Режим доступа: https://habr.com/ru/articles/483202/ (дата обращения: 2.12.2025).

14. Официальный сайт-документация к библиотеке. Режим доступа: https://pdf2image.readthedocs.io/en/latest/index.html (дата обращения: 2.12.2025).

15. Habr - серия статей на тему использования OpenCV в Python – «OpenCV в Python». Режим доступа: https://habr.com/ru/articles/519454/ (дата обращения: 2.12.2025).

16. Учебное пособие – «АЛГОРИТМИЧЕСКИЕ МЕТОДЫ СЕГМЕНТАЦИИ ИЗОБРАЖЕНИЙ» / КАЗАНСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ Специализированный учебный научный центробщеобразовательная школа-интернат «IT-лицей». Режим доступа: https://kpfu.ru/portal/docs/F265768322/Algoritmicheskie.metody.segmentacii.izobrazhenii_.pdf (дата обращения 2.12.2025).

17. Справочная информация по конфигурации tesseract (PSM, OEM). Режим доступа: https://github.com/tesseract-ocr/tesseract/blob/main/doc/tesseract.1.asc (дата обращения 2.12.2025).

18. Учебные материалы по дисциплине «РПМ» / ГАПОУ СО «Уральский государственный колледж имени И. И. Ползунова». — Екатеринбург, 2024–2025.

19. ГОСТ 7.0.5-2008. Система стандартов по информации, библиотечному и издательскому делу. Библиографическая ссылка. Общие требования и правила составления. — Введ. 2009-07-01. — М.: Стандартинформ, 2008. — 30 с.

20. ГОСТ 19.701-90. Схемы алгоритмов, программ, данных и систем. — Введ. 1992-01-01. — М.: Изд-во стандартов, 1991. — 35 с.К.

21. Официальная библиотека pytesseract. Режим доступа: https://pypi.org/project/pytesseract [21][22]/ (дата обращения: 3.12.2025)

22. Pikabu – статья по установке и использованию pytesseract. Режим доступа: https://pikabu.ru/story/raspoznavanie_teksta_s_pomoshchyu_pytesseract [21][22]_9713785 (дата обращения: 3.12.2025)

23. Habr – статья на тему JSON – «Что такое JSON». Режим доступа: https://habr.com/ru/articles/554274/ (дата обращения: 3.12.2025).

24. ГОСТ 19.102-77. Единая система программной документации. Стадии разработки. – Введ. 1980-01-01. – М. : Стандартинформ, 2010

25. Требования к оформлению пояснительной записки «Нормоконтроль» / ГАПОУ СО «Уральский государственный колледж имени И. И. Ползунова». — Екатеринбург, 2024–2025.

ПРИЛОЖЕНИЕ А

(обязательное)

ГАПОУ СО «УГК ИМ. И. И. ПОЛЗУНОВА»

Кафедра информационных систем и автоматического управления

УТВЕРЖДАЮ

руководитель курсового проекта

_____/ Е. Д. Ярочкина /

" " 2025

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

«РАЗРАБОТКА МОДУЛЯ РАСПОЗНОВАНИЯ ТЕКСТА»

разработал

студент гр. ИП-402

Кириллов Вадим

Екатеринбург 2025

1 ОСНОВАНИЯ ДЛЯ РАЗРАБОТКИ

1.1 Программа разрабатывается на основе задания на курсовое проектирование по дисциплине «Разработка программных модулей», утвержденного на заседании кафедры информационных систем и программирования протокол №2 от 19 сентября 2025 года.

1.2 Наименование работы: Разработка модуля распознавания текста.

1.3 Исполнитель: студент группы ИП-402 Кириллов Вадим Максимович

2 НАЗНАЧЕНИЕ

Модуль предназначен для автоматического распознавания текстовой информации с изображений и PDF-документов. Система обеспечивает преобразование графической информации в текстовый вид с последующим сохранением результата в файл или выводом на экран.

3 ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ К ПРОГРАММЕ

3.1 Требования к функциональным характеристикам

Загрузка и обработка изображений в форматах JPG, PNG.

Поддержка распознавания текста в PDF-файлах.

Использование библиотеки OpenCV для предобработки изображений (фильтрация, бинаризация, масштабирование).

Применение библиотеки Tesseract OCR для распознавания текста.

Поддержка русского языка распознавания.

Сохранение результата в текстовый файл (.txt) или вывод на экран.

3.2 Требования к надежности

Защита от некорректных входных данных.

Обеспечение корректной работы при изображениях низкого качества за счёт алгоритмов предобработки.

Устойчивость работы при обработке файлов среднего размера (до 20 Мб).

3.3 Требования к составу и параметрам технических средств

Программа должна быть совместима с персональными компьютерами со следующими минимальными характеристиками:

процессор: 1,8 GHz;

оперативная память: 2048 Мб;

наличие сетевого подключения;

монитор с разрешением не ниже 1280×720;

3.4 Требования к программной совместимости

модуль должен работать на операционных системах Windows;

предварительная установка библиотек Python, OpenCV, Tesseract OCR;

3.5 Условия эксплуатации

Модуль используется в составе веб-приложения. Пользователь отправляет изображение или PDF-файл, после чего модуль автоматически выполняет предобработку и распознавание текста.

Для корректной работы необходимы:

стабильное интернет-соединение;

ПК с установленным Python, OpenCV, Pillow и Tesseract OCR;

поддержка форматов JPG, PNG и PDF.

4 ТРЕБОВАНИЯ К ПРОГРАММНОЙ ДОКУМЕНТАЦИИ

4.1 В программном коде должны быть добавлены комментарии, поясняющие назначение ключевых функций и алгоритмов.

4.2 В состав программной документации должны входить:

пояснительная записка;

техническое задание;

код программного продукта;

руководство пользователя по запуску и использованию модуля.

ПРИЛОЖЕНИЕ Б

(обязательное)

КОД МОДУЛЯ «loader.py»

```
from pdf2image import convert_from_path
from PIL import Image

def load_image(file_path):
    """
    Загружает файл (изображение или PDF) и возвращает:
    - объект PIL Image для обычного изображения;
    - список объектов PIL Image для многостраничного PDF.
    Всегда конвертирует в 'RGB', чтобы обеспечить единообразие.
    """
    image = None

    try:
        if file_path.lower().endswith('.pdf'):
            # Конвертируем все страницы PDF в изображения
            pages = convert_from_path(file_path)
            if pages:
                # Приводим каждую страницу к формату RGB
                image = [page.convert('RGB') for page in pages]
        else:
            # Открываем файл изображения
            image = Image.open(file_path).convert('RGB')

    except Exception as e:
        # Эта ошибка будет поймана в main.py
        raise ValueError(f"Ошибка при загрузке файла: {e}")

    if image is None:
            raise ValueError("Не удалось обработать файл. Убедитесь, что это изображение или PDF-документ.")

    return image
```

КОД МОДУЛЯ «preprocessing.py»

```
import cv2
import numpy as np

def deskew(binary_image):
    try:
        coords = np.column_stack(np.where(binary_image > 0))
        angle = cv2.minAreaRect(coords)[-1]

        if angle < -45:
            angle = -(90 + angle)
        else:
            angle = -angle

        (h, w) = binary_image.shape[:2]
        center = (w // 2, h // 2)
        M = cv2.getRotationMatrix2D(center, angle, 1.0)

        rotated = cv2.warpAffine(binary_image, M, (w, h),
                                 flags=cv2.INTER_CUBIC,
                                 borderMode=cv2.BORDER_CONSTANT,
                                 borderValue=0)

        return rotated

    except Exception as e:
        return binary_image

def preprocess_image(image):
    """
    Выполняет полную предобработку изображения для OCR.
    """
    # 1. Конвертация из PIL в CV2 (OpenCV)
    img_cv = cv2.cvtColor(np.array(image), cv2.COLOR_RGB2BGR)

    # 2. Конвертация в оттенки серого
    gray = cv2.cvtColor(img_cv, cv2.COLOR_BGR2GRAY)

    # 3. Масштабирование
    h, w = gray.shape
    if w < 1000:
        scale = 1.5
```

```
            gray = cv2.resize(gray, (int(w * scale), int(h * scale)), interpolation=cv2.INTER_CUBIC)

    # 4. Удаление шума
    denoised = cv2.medianBlur(gray, 3)

    # 5. Бинаризация (Порог Оцу)
      _, binary_image = cv2.threshold(denoised, 0, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)

    # 6. Выравнивание
    deskewed_image = deskew(binary_image)

    return deskewed_image
```

КОД МОДУЛЯ «ocr.py»

```
import pytesseract
from pytesseract import Output

def get_text_from_image(processed_image):
    """
    Извлекает текст и данные об уверенности из обработанного изображения.
    Принимает CV2 Image (numpy array).
    """

    # --psm 6, работает стабильнее, чем 3
    custom_config = r'-l rus --oem 3 --psm 6'

    # Используем image_to_data для получения детальной информации
    data = pytesseract [21][22].image_to_data(processed_image, config=custom_config,
output_type=Output.DICT)

    full_text = []
    confidences = []

    for i in range(len(data['text'])):
        if int(data['conf'][i]) > -1:
            word = data['text'][i]
            if word.strip():
                full_text.append(word)
                confidences.append(float(data['conf'][i]))

    return " ".join(full_text), confidences
```

КОД МОДУЛЯ «main.py»

```
import sys
import time
import json

# Импортируем наши модули
from loader import load_image
from preprocessing import preprocess_image
from ocr import get_text_from_image

# Модуль 4: Экспорт данных
def export_data(text, confidence_list, start_time):
    """Формирует и выводит итоговый JSON."""
    end_time = time.time()
    processing_time = int((end_time - start_time) * 1000)  # в миллисекундах

    valid_confidences = [conf for conf in confidence_list if conf > 0]
    average_confidence = (
        sum(valid_confidences) / len(valid_confidences)
        if valid_confidences else 0
    )

    result = {
        "text": text,
        "average_confidence": round(average_confidence, 2),
        "processing_time_ms": processing_time
    }

    # ensure_ascii=True, чтобы избежать проблем с кодировкой в Go
    print(json.dumps(result, ensure_ascii=True))

def main(file_path):
    """Главная функция, координирующая весь процесс."""
    start_time = time.time()
    try:
        # Модуль 1: Загрузка
        images = load_image(file_path)

        # Приводим к списку: для PDF это уже список, для обычного изображения — одиночный элемент
        if isinstance(images, list):
            image_list = images
        else:
```

```
            image_list = [images]

        all_text_parts = []
        all_confidences = []

        # Обрабатываем каждую страницу/изображение одинаково
        for img in image_list:
            # Модуль 2: Предобработка
            processed_image = preprocess_image(img)

            # Модуль 3: Распознавание
            text, confidences = get_text_from_image(processed_image)

            if text.strip():
                all_text_parts.append(text)

            all_confidences.extend(confidences)

        # Объединяем текст со всех страниц в одну строку
        full_text = "\n\n".join(all_text_parts)

        # Модуль 4: Экспорт
        export_data(full_text, all_confidences, start_time)

    except Exception as e:
        # В случае ошибки возвращаем JSON с ошибкой
        error_result = {
            "error": str(e)
        }
        print(json.dumps(error_result, ensure_ascii=True))
        sys.exit(1)

if __name__ == "__main__":
    if len(sys.argv) > 1:
        file_path = sys.argv[1]
        main(file_path)
    else:
        # Ошибка, если путь к файлу не передан
        error_info = {"error": "Путь к файлу не был передан в скрипт."}
        print(json.dumps(error_info, ensure_ascii=True))
        sys.exit(1)
```

ПРИЛОЖЕНИЕ В

(обязательное)

РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ ПО РАБОТЕ С МОДУЛЕМ РАСПОЗНАВАНИЯ ТЕКСТА

1. Введение Настоящее руководство предназначено для пользователей программного модуля распознавания текста. Модуль предназначен для автоматического извлечения текстовой информации из графических файлов (изображений) и PDF-документов. Использование модуля позволяет существенно сократить время на оцифровку печатных документов и конспектов, преобразуя их в редактируемый формат.

2. Начало работы Модуль интегрирован в веб-приложение и не требует установки дополнительного программного обеспечения на компьютер пользователя. Для начала работы необходимо:

– запустить установленный на компьютере веб-браузер;

– в адресной строке ввести адрес веб-приложения, в состав которого входит модуль;

– на экране отобразится главный интерфейс загрузки файлов.

3. Порядок работы с модулем Процесс распознавания текста состоит из следующих шагов:

Загрузка исходных данных. В интерфейсе приложения нажмите кнопку «Загрузить файл» или перетащите файл в выделенную область. Система поддерживает следующие форматы файлов:

– изображения: JPG, PNG;

– документы: PDF (включая многостраничные файлы).

Обработка и распознавание. После загрузки файла модуль автоматически выполняет предварительную обработку изображения (удаление шумов, выравнивание строк, коррекцию освещения) с использованием алгоритмов OpenCV. Затем запускается процесс оптического распознавания символов (OCR)

на базе движка Tesseract. Дождитесь окончания процесса. Время обработки зависит от размера файла и количества страниц в PDF-документе.

Получение результата По завершении обработки на экране отобразится распознанный текст. Пользователю доступны следующие действия:

– просмотр текста в окне интерфейса;

– копирование текста в буфер обмена;

4. Возможные ошибки и методы их устранения В процессе работы могут возникать сообщения об ошибках. Ниже приведены наиболее частые ситуации и способы их решения:

– «неверный формат файла». Вы пытаетесь загрузить файл, отличный от JPG, PNG или PDF. Пожалуйста, конвертируйте файл в поддерживаемый формат.

– «файл слишком большой». Размер загружаемого файла превышает допустимый лимит (20 Мб). Попробуйте сжать изображение или разделить PDF-файл на части;

– «текст не распознан». Возможно, исходное изображение имеет слишком низкое качество, сильные искажения или рукописный текст, который сложно распознать. Попробуйте загрузить более четкое изображение.

5. Пример использования

Задача: Студенту необходимо оцифровать лекцию, сфотографированную на телефон.

– Студент открывает веб-приложение в браузере;

– загружает фотографию конспекта в формате JPG;

– система автоматически переводит изображение в оттенки серого, убирает тени и распознает текст;

– студент копирует полученный текст и вставляет его в свой документ Word для дальнейшего редактирования.

6. Заключение. Данное руководство описывает базовый функционал модуля распознавания текста. Модуль обеспечивает высокую точность обработки документов на русском и английском языках, предоставляя простой и удобный

инструмент для работы с неструктурированными данными. При возникновении технических проблем обратитесь к администратору системы.