

Министерство науки и высшего образования Российской Федерации

Санкт-Петербургский политехнический университет Петра Великого

Физико-механический институт

Высшая школа теоретической механики и математической физики

Работа допущена к защите

Директор ВШТМиМФ,

д.ф.-м.н., чл.-корр. РАН

_____ А.М. Кривцов

« » _____ 2022 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

магистерская диссертация

ВНЕДРЕНИЕ ЦИФРОВОЙ ТЕХНОЛОГИИ ОБРАБОТКИ РУКОПИСНЫХ ВЫРАЖЕНИЙ В СТРОИТЕЛЬНОЕ ПРОИЗВОДСТВО

по направлению подготовки 01.04.03 Механика и математическое
моделирование

Направленность 01.04.03_03 Механика и цифровое производство

Выполнил
студент гр. 5040103/00301

А.Д. Ефименко

Руководитель
доцент ВШТМиМФ, к.ф.-м.н.

М.Б. Бабенков

**САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА
ВЕЛИКОГО**

**Физико-механический институт
Высшая школа теоретической механики и математической физики**

УТВЕРЖДАЮ

Директор ВШТМиМФ

А.М. Кривцов

« »

2022 г.

ЗАДАНИЕ

по выполнению выпускной квалификационной работы

студенту Ефименко Александре Дмитриевне, группы 5040103/00301
фамилия, имя, отчество (при наличии), номер группы

1. Тема работы: внедрение цифровой технологии обработки рукописных выражений в строительное производство
2. Срок сдачи студентом законченной работы: 01.06.2022.
3. Исходные данные по работе: научные статьи, интернет ресурсы, литература.
4. Содержание работы (перечень подлежащих разработке вопросов): исследование существующих на сегодняшний день программных решений, распознающих и обрабатывающих выражения. Выбор архитектуры и разработка нейронной сети. Сборка обучающей выборки из рукописных символов и обучение свёрточной нейронной сети. Разработка алгоритма по обработке рукописных выражений. Ввод в эксплуатацию.
5. Перечень графического материала (с указанием обязательных чертежей): структурные элементы сервиса, образец обучающей выборки, результаты работы программы.
6. Консультанты по работе: -
7. Дата выдачи задания 13.05.2022.

Руководители ВКР

М.Б. Бабенков

(подпись)

Задание принял к исполнению 13.05.2022.

(дата)

Студент

А.Д. Ефименко

(подпись)

РЕФЕРАТ

На 38 с., 28 рисунков, 1 таблица

КЛЮЧЕВЫЕ СЛОВА: СВЕРТОЧНЫЕ НЕЙРОННЫЕ СЕТИ, СТРУКТУРА ДАННЫХ, ДЕРЕВО ВЫРАЖЕНИЙ, ОБУЧАЮЩАЯ ВЫБОРКА, МОДЕЛИРОВАНИЕ, РЕАЛИЗАЦИЯ АЛГОРИТМА

Тема выпускной квалификационной работы: «Внедрение цифровой технологии обработки рукописных выражений в строительное производство»

Целью дипломной работы является разработка программы по обработке рукописных математических выражений, вводимых пользователем. Программа должна быть представлена в виде сервиса, то есть должна обладать стандартизированным интерфейсом, чтобы её можно было легко внедрить в любой сторонний продукт. Сервис должен принимать в качестве входных данных изображение, содержащее одно нарисованное от руки математическое выражение. На выходе должна получаться древовидная структура, описывающая данное выражение и пригодная для дальнейшей обработки.

Для достижения поставленной цели следует решить следующие задачи:

1. Провести анализ предметной области.
2. Разработать алгоритм, обрабатывающий символы и выдающий структуру данных – дерево выражения.
3. Собрать обучающую выборку из рукописных символов.
4. Смоделировать и реализовать нейронную сеть, распознающую математические символы.
5. Протестировать проект.
6. Присоединить проект к стороннему программному обеспечению.

В ходе работы был проведен анализ предметной области, собрана обучающая выборка символов, смоделирована и разработана нейронная сеть,

распознающая эти символы, реализован алгоритм, обрабатывающий распознанные символы и строящий по ним дерево выражения.

В результате работы был разработан сервис по обработке рукописных математических выражений.

ABSTRACT

38 pages, 28 figures, 1 tables

KEYWORDS: CONVOLUTIONAL NEURAL NETWORKS, DATA STRUCTURE, EXPRESSION TREE, TRAINING SAMPLE, MODELING, ALGORITHM IMPLEMENTATION

The subject of the graduate qualification work is "The introduction of digital technology for processing handwritten expressions in the construction industry"

The purpose of the thesis is to develop a program for processing handwritten mathematical expressions entered by the user. The program must be presented as a service, that is, it must have a standardized interface so that it can be easily implemented in any third-party product. The service must accept as input an image containing a single hand-drawn mathematical expression. The output should be a tree structure that describes this expression and is suitable for further processing.

To achieve this goal, the following tasks should be solved:

1. Conduct an analysis of the subject area.
2. Develop an algorithm that processes symbols and produces a data structure - an expression tree.
3. Collect a training sample from handwritten characters.
4. Model and implement a neural network that recognizes mathematical symbols.
5. Test the project.
6. Attach the project to third-party software.

In the course of the work, an analysis of the subject area was carried out, a training sample of characters was collected, a neural network was modeled and developed that recognizes these characters, an algorithm was implemented that processes the recognized characters and builds an expression tree based on them.

As a result of the work, a service was developed for processing handwritten mathematical expressions.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	5
ГЛАВА 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ.....	7
1.2. ВВЕДЕНИЕ В РАСПОЗНАВАНИЕ ВЫРАЖЕНИЙ	7
1.3. ОБЗОР СУЩЕСТВУЮЩИХ РЕШЕНИЙ	8
1.4. ОБЗОР ИНСТРУМЕНТОВ РАЗРАБОТКИ.....	12
1.5. ВЫВОД ПО ГЛАВЕ	14
ГЛАВА 2. ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ СЕРВИСА.....	15
2.2. ПРОЕКТИРОВАНИЕ СЕРВИСА.....	15
2.3. РЕАЛИЗАЦИЯ IMAGE PARSER.....	16
2.4. РЕАЛИЗАЦИЯ LAYOUT PASS	18
2.5. РЕАЛИЗАЦИЯ TRANSFORM PASS	29
2.6. РАЗРАБОТКА МОДЕЛИ НЕЙРОННОЙ СЕТИ	30
2.7. ФОРМИРОВАНИЕ ОБУЧАЮЩЕЙ ВЫБОРКИ.....	32
2.8. ВЫВОД ПО ГЛАВЕ	33
ГЛАВА 3. ДОПОЛНИТЕЛЬНАЯ РАЗРАБОТКА И ТЕСТИРОВАНИЕ.....	34
3.2. ИНТЕГРАЦИЯ С WOLFRAM ENGINE.....	34
3.3. РАЗРАБОТКА ИНТЕРФЕЙСА ДЛЯ ТЕСТИРОВАНИЯ.....	35
3.4. РАЗРАБОТКА АЛГОРИТМА ПО АВТОМАТИЗАЦИИ ТЕСТИРОВАНИЯ.....	37
3.5. ТЕСТИРОВАНИЕ ПРОЕКТА	38
3.6. ВЫВОД ПО ГЛАВЕ	39
ЗАКЛЮЧЕНИЕ	40
СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ	41

ВВЕДЕНИЕ

Человечество всегда стремилось упростить, автоматизировать процесс выполнения любой работы. С появлением и развитием компьютерной техники, появилось ещё больше возможностей для этого. Люди стали всё реже писать на бумаге, ведь куда удобнее и быстрее напечатать текст на клавиатуре и хранить его в компьютере. В связи с этим, на рынке программного обеспечения стали появляться продукты, позволяющие распознавать и оцифровать текст. Сегодня имеется немало решений, таких как Abby FineReader [1], Google Docs [2], которые обладают большой функциональностью, имеют хорошую точность при распознавании и поддерживают множество языков.

Однако, это не решило другую проблему. Использование клавиатуры для ввода математических выражений вызывает затруднение у многих пользователей, привыкших записывать их при помощи ручки или стилуса. Поэтому необходимо было придумать средства автоматизации, позволяющие пользователям преобразовывать графически записанные выражения в понятный для обработки и хранения на компьютере формат. Проблема распознавания математического текста является более комплексной, по сравнению с распознаванием обычного текста, потому что выражение может представляться двумерной структурой, а не одномерной, как текст.

Целью дипломной работы является разработка программы по обработке рукописных математических выражений, вводимых пользователем. Программа должна быть представлена в виде сервиса, то есть должна обладать стандартизированным интерфейсом, чтобы её можно было легко внедрить в любой сторонний продукт. Сервис должен принимать в качестве входных данных изображение, содержащее одно нарисованное от руки математическое выражение. На начальном этапе разработки алгоритм должен уметь работать с числами, основными математическими операциями, скобками и дробями, а также со степенями и индексами символов. На выходе должна получаться древовидная структура, описывающая данное выражение и пригодная для дальнейшей обработки.

Для достижения данной цели следует сформулировать и решить следующие задачи:

7. Провести анализ предметной области.
8. Собрать обучающую выборку из рукописных символов.
9. Смоделировать и реализовать нейронную сеть, распознающую математические символы.
10. Разработать алгоритм, обрабатывающий распознанные символы и выдающий структуру данных – дерево выражения.
11. Протестировать проект.
12. Присоединить проект к стороннему программному обеспечению.
13. Внедрить в строительное производство.

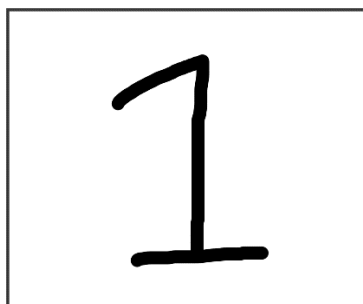
Дипломная работа состоит из введения, трёх глав, заключения и списка используемой литературы. Каждая глава посвящена одному из этапов разработки. В первой главе производится анализ уже существующих решений, учёт их достоинств и недостатков. Также в этой главе представлены инструменты, которые будут применяться для разработки проекта. Во второй главе описываются ключевые моменты проектирования и реализации сервиса. В ней подробно рассматривается его алгоритм работы, а также основные компоненты, в том числе и нейронная сеть. Третья глава посвящена интеграции проекта с программой Wolfram Engine, дополнительной разработке и тестированию проекта.

ГЛАВА 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.2. ВВЕДЕНИЕ В РАСПОЗНАВАНИЕ ВЫРАЖЕНИЙ

За последние десятилетия появилось немало теории и алгоритмов по распознаванию и обработке текстов. Всех их можно разделить на две группы по типу распознавания: **онлайн** и **офлайн** [3]. То же самое деление действует и на алгоритмы по распознаванию математических текстов.

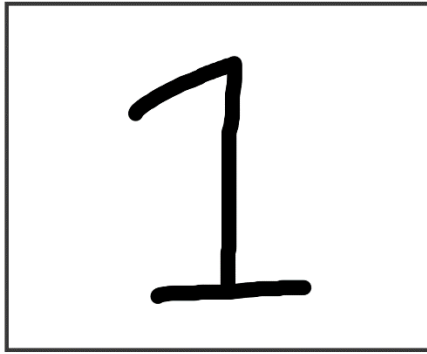
Когда говорится об онлайн распознавании, то обычно подразумевается, что пользователь вводит формулу в реальном времени. На вход алгоритму подаются данные, представленные набором **росчерков** – хронологический набор точек на плоскости, повторяющих перемещения инструмента ввода (пальца, мышки или стилуса). В результате, помимо самих символов, алгоритм получает дополнительную информацию, например, направление написания, скорость движения пера в разные моменты времени и т.д. Это даёт больше материала для анализа, однако, следует учесть, что некоторые символы могут состоять из нескольких росчерков.



```
1 <svg width="800" height="600" xmlns="http://www.w3.org/2000/svg">
2 <g>
3 <title>Layer 1</title>
4 <path d="m258.5,137c1,-1 5.48047,-6.35942 15,-14c12.69537,-10.18957 23.84796,..."
5 <path d="m259.5,276c1,1 6,1 12,1c11,0 18,0 27,0c10,0 21,0 29,0c9,0 16,0 22,0c5,..."
6 </g>
7 </svg>
```

Рисунок 1. Пример представления символа в виде росчерков в формате svg.

Алгоритмы офлайн распознавания подразумевают использование только графической информации в качестве входных данных, то есть цифрового изображения с математическим текстом. Дополнительно здесь встаёт задача выделения символов из фона и группировки их в структуру – выражение.



0	0	0	0	0	0	0
0	0	0	255	0	0	0
0	0	255	255	0	0	0
0	255	0	255	0	0	0
0	0	0	255	0	0	0
0	0	0	255	0	0	0
0	0	0	255	0	0	0
0	0	0	255	0	0	0
0	0	0	255	0	0	0
0	0	0	255	0	0	0
0	255	255	255	255	255	0
0	0	0	0	0	0	0

Рисунок 2. Пример растрового представления символа.

Отсутствие дополнительной информации, имеющейся при онлайн распознавании, негативно сказывается на точности. Отсюда вытекают такие проблемы как:

- наложение символов друг на друга;
- дефекты бумаги при сканировании;
- вариативность начертания символов (размер, наклон и т.д.);
- и другие.

Процесс офлайн распознавания обычно состоит из трёх основных шагов:

1. Этап предобработки: повышение качества изображения, удаление дефектов и т.д.
2. Выделение отдельных символов на изображении.
3. Распознавание символов.

1.3. ОБЗОР СУЩЕСТВУЮЩИХ РЕШЕНИЙ

Перед началом работы над проектом, следует ознакомиться с уже существующими решениями: провести анализ, выявить их достоинства и недостатки, чтобы выбрать правильное направление разработки.

InftyReader [4]

Это приложение позволяет распознавать сканированные страницы научных текстов, которые могут включать математические выражения, и предоставляет результат в одном из форматов на выбор: LaTeX, Plain text, PDF, XML и другие.

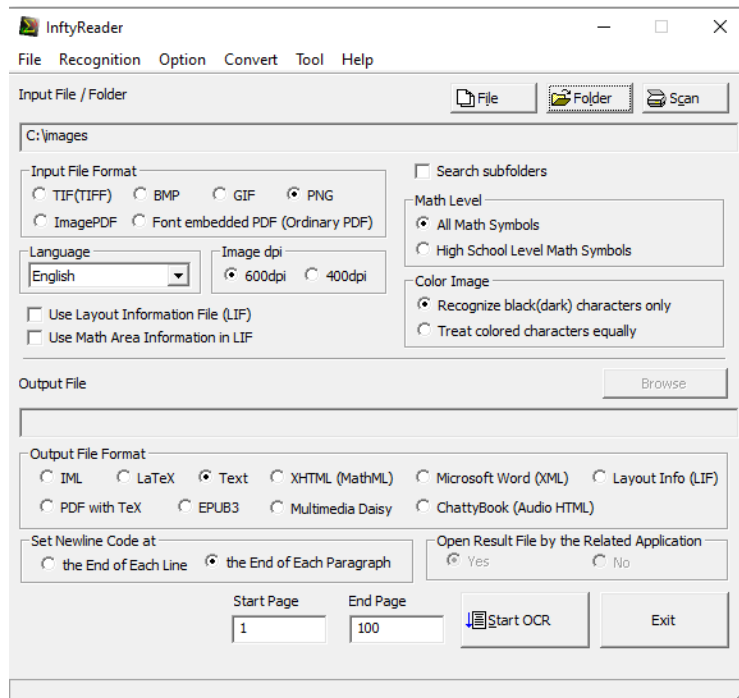


Рисунок 3. Скриншот программы InftyReader

Проект давно разрабатывается и имеет высокую точность распознавания текстов. К недостаткам можно отнести то, что он работает только с печатным текстом.

Mathematical Expression Recognition [5]

Веб-сервис, базирующийся на SESHAT – открытой системе онлайн распознавания рукописных выражений и перевода их в LaTeX и другие форматы.

$$(3 + 8)^2$$

$$(3 + 8)^{\{2\}}$$

Search this in [Google](#) or in [WolframAlpha](#).

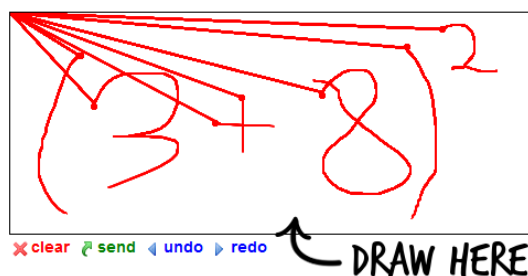


Рисунок 4. Скриншот окна ввода выражения

Сервис поддерживает цифры, буквы латинского алфавита, арифметические и логические операторы, скобки, некоторые греческие символы, а также интеграл, сумму, корень и т.д.

Photomath [6]

Мобильное приложение, разработанное для решения математических примеров, заснятых при помощи камеры. Начало своё развитие с 2014 года и на то время решало только простые математические примеры. На текущий момент программа получила широкое распространение и поддержку работы с более комплексными задачами, например, интегрирование, тригонометрия и т.д.

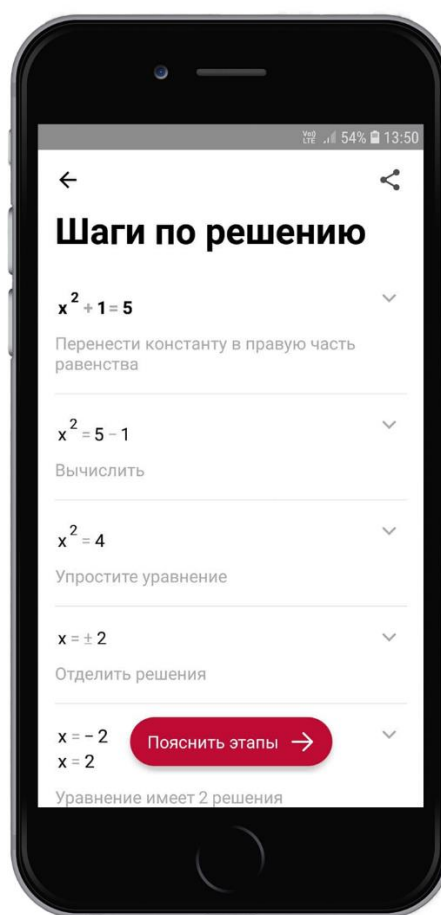


Рисунок 1.5. Скриншот программы

MyScript [7]

Система, позволяющая производить онлайн распознавание росчерков пера. Предоставляется в виде онлайн сервиса или в виде фреймворка, подключаемого к проекту. Поддерживает распознавание математического и обычного текста, диаграмм. Результат предоставляется в формате LaTeX или MathML.

MyScript поддерживает множество языков, а также позволяет добавлять свои собственные языки и задавать параметры для более корректного распознавания.



Рисунок 1.6. Скриншот интерфейса MyScript

Mathpix [8]

Программа по оцифровке рукописного или печатного текста. Является наиболее удобным и мощным средством офлайн распознавания из имеющихся на текущий момент. Позволяет делать скриншот участка экрана, а затем распознавать и переводить текст и выражения в LaTeX. Помимо этого, предоставляется возможность осуществить быстрый поиск по тексту в Интернете или найти значение распознанного выражения.

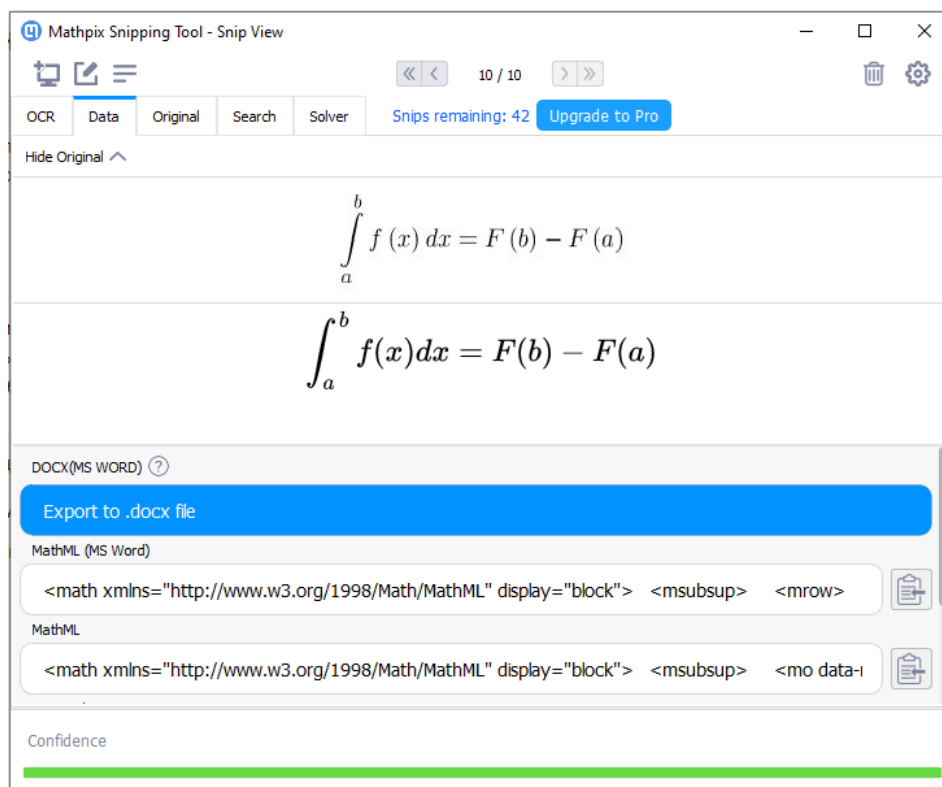


Рисунок 5. Скриншот окна программы Mathpix

Некоторые из рассмотренных решений работают по модели онлайн распознавания, что, конечно, повышает точность распознавания, однако не позволяет работать с уже подготовленными изображениями с математическими текстами, что не соответствует поставленной задаче.

Также большинство рассмотренных проектов не предоставляют возможность управлять параметрами распознавания символов или выражений. Это можно отнести к недостаткам, потому что нельзя контролировать алгоритм распознавания в зависимости от входных параметров, добиваясь более высокой точности на конкретных данных.

Выявленные достоинства и недостатки следует учесть при разработке собственного сервиса.

1.4. ОБЗОР ИНСТРУМЕНТОВ РАЗРАБОТКИ

На сегодняшний день существует большое разнообразие языков и фреймворков, поэтому перед началом работы над проектом следует

проанализировать все доступные инструменты и выбрать наиболее подходящие.

В проекте было решено применять следующие программные средства и инструменты:

Python [9] – это язык программирования общего назначения, поддерживающий несколько парадигм программирования [10]. Благодаря выразительному и минималистичному синтаксису, имеет низкий порог вхождения и позволяет писать сложную логику с минимальными затратами времени. Наличие огромного количества научных библиотек (таких как `numpy` и т.д.) даёт возможность использовать готовые алгоритмы для проведения математических расчётов и написания программ по обработке данных.

PyCharm [11] – интегрированная среда разработки, разработанная специально для языка Python. Предоставляет средства для анализа и редактирования кода, а также визуальный отладчик и графическую оболочку для систем управления версиями. Помимо этого, имеется автодополнение кода и встроенные средства форматирования исходников.

Keras [12] – открытая библиотека, нацеленная на создание нейронных сетей. Данная библиотека проста в понимании и удобна при разработке, потому что она предоставляет программисту высокоуровневый интерфейс, включающий строительные блоки (такие как слои, оптимизаторы, целевые функции) из которых формируется сама сеть. Также большим преимуществом для Keras является наличие хорошей документации и множества примеров.

Git [13] – одна из популярных систем для отслеживания версий файлов, обладающая большими возможностями и хорошей документацией. Позволяет вести историю изменений отдельных файлов. Доступ к истории даёт возможность просматривать все внесённые изменения в проект, возвращаться к предыдущим версиям отдельных файлов или всего проекта.

OpenCV [14] – большая библиотека, включающая функции по обработке изображений и алгоритмы компьютерного зрения. В проекте будет использоваться для выделения отдельных фрагментов (математических символов) на изображении.

Wolfram Engine for developers [15] – программный продукт, включающий язык программирования **Wolfram Language** в качестве

компонента, который можно легко подключить к собственному стеку разработки. Простой интерфейс для взаимодействия позволяет легко внедрить его в свой проект. А благодаря обширному спектру возможностей, wolfram будет способен решать любые выражения, распознанные сервисом.

1.5. ВЫВОД ПО ГЛАВЕ

Анализ предметной области показал, что на текущий момент уже имеются различные подходы и инструменты для распознавания рукописных математических выражений. Однако разработки в данной области только начинают развиваться, поэтому большинство из них имеют недостатки. Серьёзным недостатком большинства проектов является отсутствие возможности конфигурирования алгоритма под себя. В MyScript это учтено, однако MyScript предоставляет только онлайн распознавание, что не позволяет работать с заранее записанными графически математическими выражениями. Все достоинства и недостатки следует учесть при разработке собственного сервиса.

На этапе выбора инструментов разработки были рассмотрены различные варианты и выбраны наиболее подходящие и удобные в использовании. Основным языком программирования был выбран python, как наиболее подходящий для этих целей. Для создания нейронной сети решено использовать фреймворк Keras благодаря простоте и наличию необходимого функционала.

Выбор инструментов разработки позволил более чётко представить конечный вид проекта, определил вектор его развития.

ГЛАВА 2. ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ СЕРВИСА

2.2. ПРОЕКТИРОВАНИЕ СЕРВИСА

На этапе проектирования сервиса были приняты следующие решения. Сервис должен представляться в виде нескольких модулей Python, которые можно легко подключить к любому другому проекту. Разработчику будет доступен API состоящий из трёх функций:

- `prepare_network`
- `math_recognition_image`
- `math_recognition_image_by_path`

Первая функция должна вызываться только один раз для инициализации сервиса. Она строит модель свёрточной сети и загружает сохранённые веса.

Вторая функция принимает в качестве параметра изображение, представленное матрицей $N \times M \times 3$. Третья – путь к изображению. Обе эти функции взаимозаменяемые и отличаются только входными параметрами. Они возвращают дерево выражения, которое в дальнейшем должно обрабатываться самим клиентом.

Изображение должно содержать одно математическое выражение, нарисованное на белом фоне. Каждый символ выражения должен не соприкасаться с другими и не содержать пропусков. Для корректного распознавания позиций, символы следует располагать согласно правилам написания математических текстов.

Разработка сервиса по распознаванию математического текста – это очень объёмная задача, поэтому её было решено поделить на ряд этапов. На текущей итерации разработки алгоритм должен уметь обрабатывать выражения, состоящие из чисел, основных математических операций (сложение, вычитание, умножение и деление), скобок и дробей; определять принадлежность символов к степени или к индексу. Основной алгоритм по обработке выражения было решено реализовать, опираясь на следующие публикации: [16], [17], [18].

На выходе должна получаться структура данных дерево, описывающая выражение и все его особенности. Каждый узел дерева описывает один

символ выражения. Отношения между символами выражаются при помощи ссылок на другие узлы. Пример получаемого дерева изображён на рисунке 6.

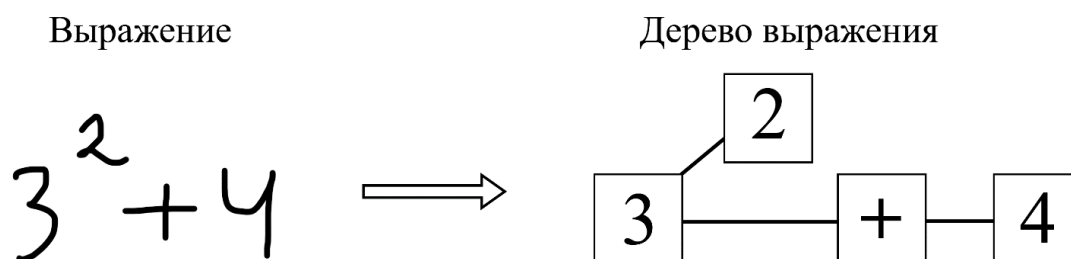


Рисунок 6. Графическое представление дерева выражения.

Сервис решено поделить на несколько взаимосвязанных компонентов, каждый из которых отвечает за различные этапы обработки входного изображения и формирования дерева.

Список основных компонентов проекта:

1. Image parser – получает на вход изображение с формулой и возвращает список распознанных символов вместе с их геометрическим положением.
2. Layout pass – основной компонент сервиса, отвечающий за анализ структуры выражения и построение «сырого» дерева выражения.
3. Transform pass – алгоритм, вносящий коррективы в «сырое» дерево по определённым правилам.
4. Дальнейшая обработка – сторонний компонент, который будет производить дальнейшую обработку дерева выражения. Например, находить его значение.

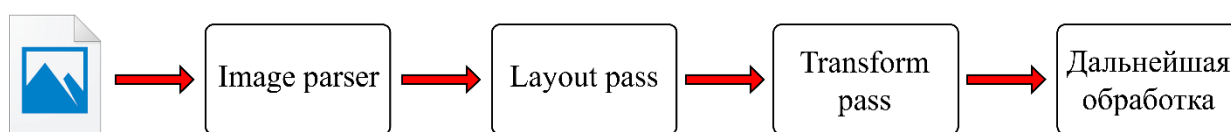


Рисунок 7. Модель сервиса

2.3. РЕАЛИЗАЦИЯ IMAGE PARSER

Image parser делит исходное изображение на фрагменты (по одному на каждый символ), а затем распознаёт каждый символ с помощью нейронной сети.

Предобработка изображения производится с помощью библиотеки OpenCV. Сначала изображение подготавливается: преобразуется в полутоновое и бинаризуется. Это нужно, чтобы упростить дальнейшую обработку. Для нахождения отдельных символов и отделения их от фона используется функция *findContours*. Она строит контур (кривая, описывающая границы объекта) вокруг каждого отдельного символа. Зная контур, мы можем найти геометрические границы символа и затем вырезать его. После того, как были найдены все символы выражения, слишком маленькие из них отбрасываются (шум).

Далее, чтобы нейронная сеть могла распознать изображения, их нужно масштабировать до определённого размера. Путём тестирования был выбран оптимальный размер картинок – 64x64 пикселя. В таком формате несильно теряются детали изображения при сжатии и достаточно немного входных параметров для нейронной сети, чтобы она быстро обучалась и производила классификацию.

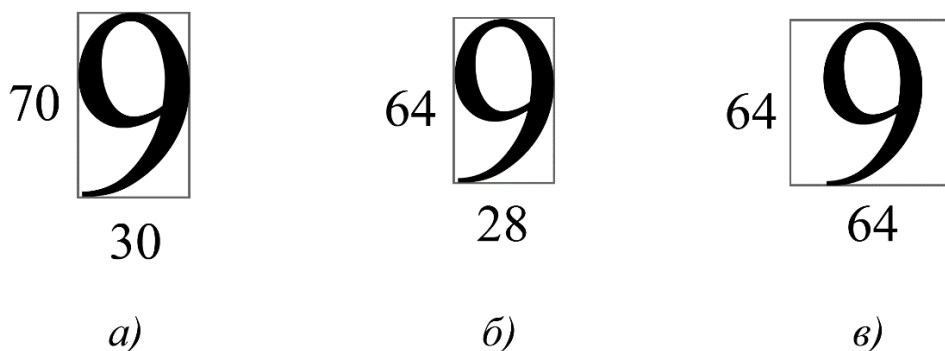


Рисунок 8. Этапы масштабирования символа.

Процесс масштабирования показан на рисунке 8. Вырезанный символ имеет произвольный размер (пункт а). Чтобы прийти к размеру 64x64, выбирается наибольшая сторона и по ней осуществляется масштабирование до 64 пикселей (пункт б) с сохранением пропорций, а наименьшая сторона расширяется пустым пространством (пункт в).

Масштабированные изображения объединяются в список, нормализуются и передаются нейронной сети. Её задача – классифицировать каждое изображение и выдать список из распознанных символов. Также в отдельный список сохраняются размер и позиция исходного символа на изображении. Все три этих списка передаются следующему компоненту.

2.4. РЕАЛИЗАЦИЯ LAYOUT PASS

Теория

В математической нотации важную роль играют не только сами символы, но и их расположение. Оно определяет отношения между символами в выражении. Например, какие символы выступают в роли аргументов, а какие – в роли операторов. Эту информацию можно получить при помощи анализа **доминирования операций и базовых линий** выражения.

Проверка **доминирования** символов используется для нахождения стартового символа при построении базовой линии. Доминирование определяется на основе расположения символов и их класса. Для нашего проекта достаточно следующего определения доминирования: символ А доминирует над символом В, если А – это дробь, а В принадлежит дроби.

Базовая линия в математической нотации – это ряд из горизонтально расположенных символов выражения. Например, в выражении $x^{z-1} + y$ имеются две базовые линии: $(x, +, y)$ и $(z, -, 1)$. В данном примере $(z, -, 1)$ – это **вложенная** базовая линия, то есть базовая линия, которая сдвинута вертикально относительно главной.

Главной называется базовая линия, состоящая из символов, которые не являются вложенными относительно других символов выражения.

Стартовый символ – это крайний левый символ главной базовой линии.

а) б)

Рисунок 9. Пример различных стартовых символов (выделены красным).

В первом случае (а) крайний левый символ и является стартовым, потому что лежит на главной базовой линии. Но во втором случае (б) стартовым является дробь, т.к. она принадлежит главной базовой линии и доминирует над всеми другими символами, расположенными во вложенных базовых линиях.

Алгоритм Layout Pass строит древовидную структуру, которую будем называть **Baselines Tree (BT)**. Она представляет из себя описание самих символов и их расположения относительно друг друга, без привязки к какому-то синтаксису или семантике. Например, Baselines Tree может описать выражение «2 +», несмотря на то, что оно синтаксически и семантически неверно.

Baselines Tree состоит из узлов класса Symbol. Экземпляр класса Symbol представляет один математический символ. Он содержит сам символ, который был распознан нейронной сетью, атрибуты символа (класс, координаты рамки символа, регионы, центроид), а также ссылки на другие узлы.

Часть кода класса Symbol:

```
class Symbol:
    """Класс, описывающий один символ из математического выражения"""
    __symbol_label = None
    __symbol_class = None
    __next = None
    __super = None
    __subsc = None
    __above = None
    __below = None
    __bounds = None
    __centroid = None
    __regions = None

    def __init__(self, label):
        self.__symbol_label = label

    @property
    def symbol_class(self):
        return self.__symbol_class

    @symbol_class.setter
    def symbol_class(self, sym_class):
        self.__symbol_class = sym_class

# .....
```

В выражении часто один символ окружён другими. В зависимости от их расположения, они будут относиться или к степени, или к индексу первого символа, или находиться с ним на одной базовой линии. Назовём области вокруг символа **регионами** и с помощью них будем определять пространственные отношения между символами.

Регион – это область прямоугольной формы со сторонами, параллельными координатным осям. Пример разбиения на регионы показан на рисунке 10.

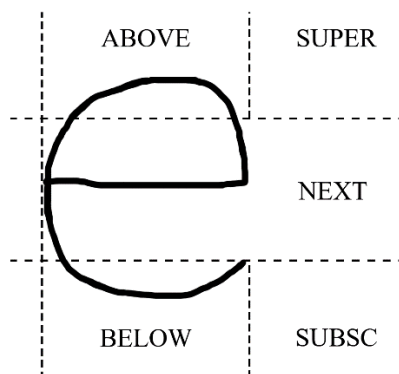


Рисунок 10. Регионы символа.

В проекте решено использовать 5 областей:

- SUPER (степень)
- SUBSC (индекс)
- ABOVE (над)
- BELOW (под)
- NEXT (за)

Этого достаточно для описания простых математических выражений, но, для расширения возможностей системы, можно добавить ещё и другие, например, CONTAINS (содержит) для описания корня.

Если символ В лежит, например, в области SUPER символа А, то это означает, что В является степенью А.

Все пространственные отношения между символами в Baselines Tree фиксируются с помощью ссылок. Каждый экземпляр Symbol содержит пять ссылок с аналогичными названиями: super, subsc, above, below и next. Если какая-то ссылка символа не нулевая, значит он состоит в пространственном отношении с другим символом выражения.

Ссылка next имеет особый статус, потому что она связывает горизонтально расположенные символы, то есть определяет базовые линии.

Экземпляр Symbol

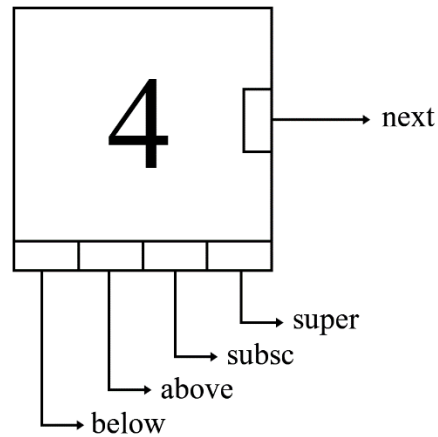


Рисунок 11. Графическое представление экземпляра класса Symbol.

Таким образом, выражение, например $x_{ij} \cdot y + \frac{a+b}{c}$, после обработки алгоритмом будет представляться такой структурой данных:

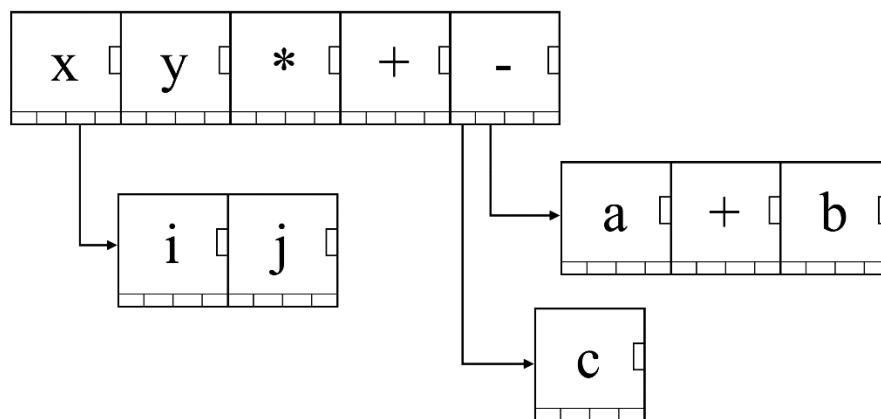


Рисунок 12. Графическое представление выражения в терминах ВТ.

Baselines Tree выражения (рис. 12) состоит из главной базовой линии, включающей символы ($x, y, *, +, -$), и нескольких вложенных: (i, j), ($a, +, b$), (c). Каждый символ из одной базовой линии (за исключением последнего), находится в отношении next со своим соседом.

Символ "x" имеет ссылку subsc на базовую линию (i, j). На рисунке это отображается с помощью стрелки. Данная связь означает, что символы из вложенной базовой линии принадлежат индексу "x". То же самое касается ссылок above и below символа "-", указывающих, что над и под ним имеются другие базовые линии.

Так как каждый символ графически изображается по-разному, то и границы регионов будут у каждого свои. Чтобы упростить работу по

определению границ, сгруппируем все используемые символы по классам, а для каждого класса установим правила нахождения границ. В результате получится следующая таблица:

Класс символа	у- центроид	Пороговые значения			
		BELOW	ABOVE	SUBSC	SUPER
Без индексов +, -	$\frac{1}{2}H$	$\frac{1}{2}H$	$\frac{1}{2}H$	-	-
Открывающаяся скобка (cH	$minY$	$maxY$	-	-
Plain: Ascender 0, ..., 9	cH	tH	$H - tH$	tH	$H - tH$
Plain: Centered , .	$\frac{1}{2}H$	tH	$H - tH$	tH	$H - tH$

Таблица 2.1.

В таблице описаны возможные классы и символы, принадлежащие им. Для каждого класса указано как находить границы (пороги) тех или иных регионов. Значения $maxY$ и $minY$ – это максимальная и минимальная координата рамки (границы) символа по у соответственно; H – высота рамки символа.

Для регулирования пороговых значений в некоторых формулах применяются переменные c (оказывает влияние на нахождение центра символа) и t (влияет на значение порога).

Чтобы получать адекватные для применения пороговые значения, будем соблюдать неравенство:

$$0 \leq t \leq c \leq 0.5$$

Чтобы упростить геометрическую проверку на принадлежность символа к определённому региону, он заменяется на точку – центроид символа. Использование одной точки для этой цели упрощает геометрический анализ. Центроид по x координате всегда находится по центру рамки символа, а по y вычисляется из таблицы.

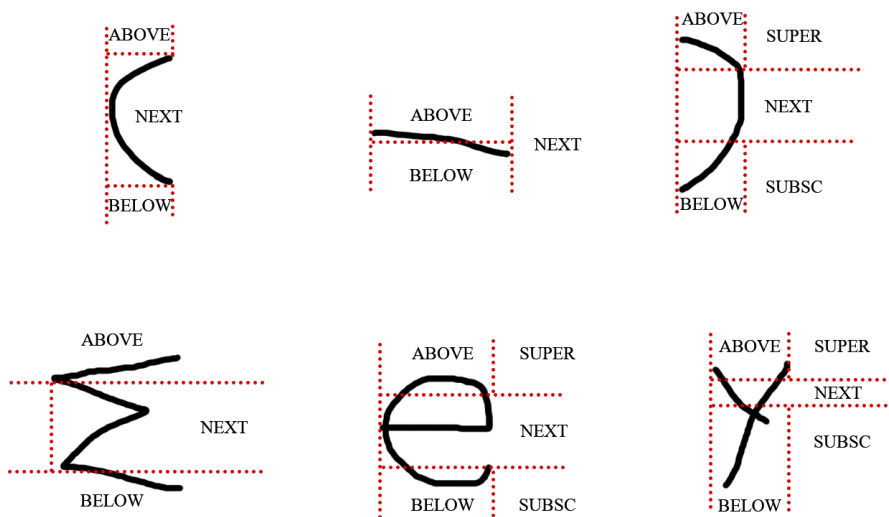


Рисунок 13. Пример регионов для различных классов символов.

Описание Layout pass.

На вход Layout pass получает список из распознанных символов и их геометрические границы. Алгоритм начинается с предобработки входных данных – для каждого символа создаётся экземпляр класса `Symbol`, который инициализируется значением символа и соответствующими атрибутами. Из этих экземпляров формируется список L , длину которого обозначим $n = |L|$. Символы в списке L сортируются по $minX$.

Затем выполняются 5 основных шагов Layout pass, описанных ниже:

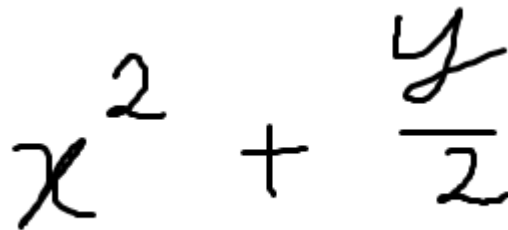
1. Находится стартовый символ $S_{start} = find_start_symbol(L)$. Функция `find_start_symbol` обрабатывает случаи, если стартовый символ не самый крайний левый. Например, числитель или знаменатель может быть немного левее границы черты дроби.
2. Находится символ S_k для S_{start} . Обычно это следующий символ из той же базовой линии, но, для класса «Non-Scripted» и «Open Bracket» – это первый правый сосед. Если S_{start} последний в своей базовой линии, то присваиваем $S_k = null$.
3. Символы, лежащие до S_{start} и между S_{start} и S_k (если $S_k = null$, то до S_n включительно), принадлежат к регионам S_{start} и, поэтому распределяются по четырём спискам: L_{super} , L_{subsc} , L_{above} , L_{below} , в зависимости от того, к какому региону S_{start} они относятся. Все остальные символы S_k, \dots, S_n остаются нераспределёнными и заносятся в отдельный список L_{next} .

4. Каждый непустой список, найденный на предыдущем шаге, обрабатывается рекурсивно алгоритмом и результат добавляется к соответствующему региону символа S_{start} .
5. Алгоритм возвращает S_{start} .

В результате получается Baselines Tree – дерево, корнем которым выступает стартовый символ доминирующей базовой линии.

Ниже описан пример работы алгоритма при разборе конкретного выражения: $x^2 + \frac{y}{2}$.

Так как данное выражение можно по-разному изобразить графически, то возьмём только один вариант и разберём работу алгоритма на нём. Пусть выражение записано следующим образом:



The image shows a handwritten mathematical expression $x^2 + \frac{y}{2}$. The 'x' is written in a cursive style, the '2' is a simple '2', the '+' is a simple plus sign, the 'y' is cursive, and the denominator '2' is a simple '2'.

Рисунок 14.

Сначала производится обработка изображения и формирование списка из распознанных символов. Всё это осуществляется компонентом *image parser*. Далее список передаётся алгоритму *layout pass*.

Для каждого математического символа создаётся объект класса *Symbol*, описывающий его:

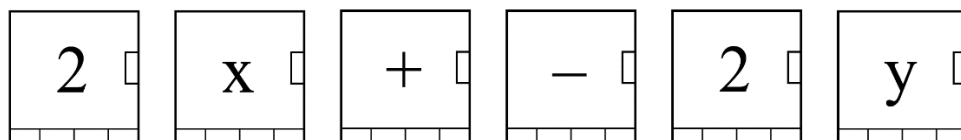


Рисунок 15.

Все объекты объединяются в новый список с именем *symbols* и сортируются по *xmin*:

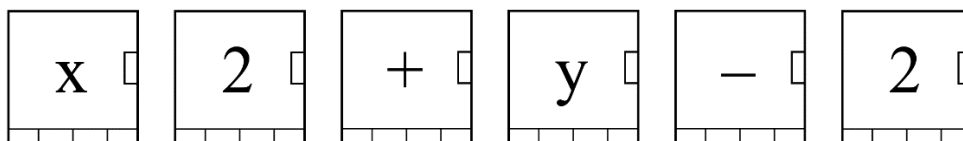


Рисунок 16.

Затем происходит вызов основной функции *layout_pass(symbols)*, которая и формирует Baselines Tree. Рассмотрим этот процесс по шагам. Для наглядности обработка списков регионов выделена в подпункты.

1. Находится стартовый символ $S_{start} = "x"$.
2. При помощи функции *find_next_in_baseline* находится следующий за ним в той же базовой линии. Этим символом будет $" + "$.
3. Согласно 3-му шагу алгоритма, так как символ $"2"$ лежит между S_{start} и S_k , то он принадлежит одному из регионов S_{start} . При помощи функции *belong_region(...)* определяется, что $"2"$ относится к региону *SUPER*, поэтому этот символ добавляется в список L_{super} .
Все остальные символы $(+, y, -, 2)$ заносятся в список L_{next} .
4. На текущем шаге имеются два непустых списка: L_{super} и L_{next} , которые обрабатываются рекурсивно. Рассмотрим сначала обработку списка L_{super} .
 - 4.1 Для L_{super} находится стартовый символ $S_{start} = "2"$.
 - 4.2 Из-за того, что список L_{super} состоит из одного элемента, то S_k не будет найден, то есть будет равен *null*.
 - 4.3 Списки регионов будут пустыми.
 - 4.4 Так как все списки пустые, то текущая ветка рекурсии завершена.
 - 4.5 Рекурсивный вызов вернёт S_{start} .

Возвращённый символ $"2"$ будет присоединён к региону *SUPER* символа $"x"$.

Графически это можно представить так:

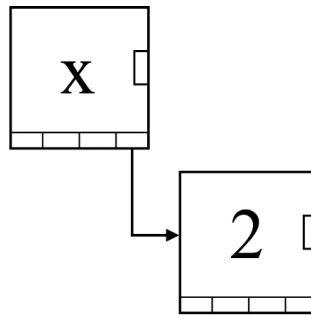


Рисунок 17.

Далее рекурсивно вызывается $layout_pass(L_{next})$.

5. Стартовый символ – это " + " (исходя из его позиции).
6. Следующим для него символом будет $S_k = "y"$.
7. Так как до S_k нет других символов, то списки регионов остаются пустыми, а $L_{next} = (y, -, 2)$.
8. На текущий момент имеется непустой список L_{next} . Вызывается рекурсивно $layout_pass(L_{next})$.
9. Несмотря на то, что "y" левее, стартовым символом будет " – ", потому что он доминирующий.
10. Для символа " – " не имеется следующего символа, поэтому $S_k = null$.
11. Оставшиеся символы $(y, 2)$ распределяются по спискам: $L_{above} = (y)$ и $L_{below} = (2)$. L_{next} остаётся пустым.
12. На текущем этапе имеются только два непустых списка, которые обрабатываются рекурсивно.
 - 12.1.1 Для L_{above} находится стартовый символ $S_{start} = "y"$.
 - 12.1.2 $S_k = null$.
 - 12.1.3 Списки регионов и L_{next} пустые.
 - 12.1.4 Так как все списки пустые, то текущая ветка рекурсии завершена.
 - 12.1.5 Вызов возвращает $S_{start} = "y"$.
 - 12.2.1 Для L_{below} находится стартовый символ $S_{start} = "2"$.
 - 12.2.2 $S_k = null$.
 - 12.2.3 Списки регионов и L_{next} пустые.
 - 12.2.4 Так как все списки пустые, то текущая ветка рекурсии завершена.
 - 12.2.5 Вызов возвращает $S_{start} = "2"$.

Возвращённые символы "у" и "2" присоединяются к регионам *above* и *below* символа " – " соответственно.

Текущее состояние можно наблюдать на картинке:

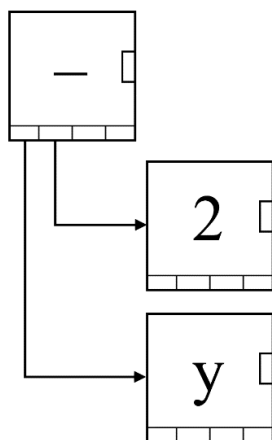


Рисунок 18.

13. Теперь разворачивается рекурсия по базовой линии. Сначала к региону *NEXT* символа " + " будет присоединён символ " – ".

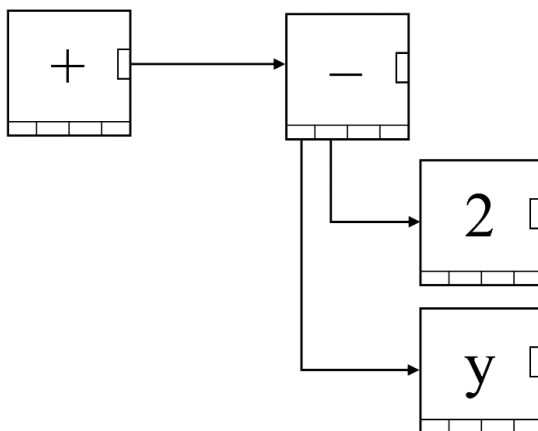


Рисунок 19.

А затем к «х» будет присоединён " + ".

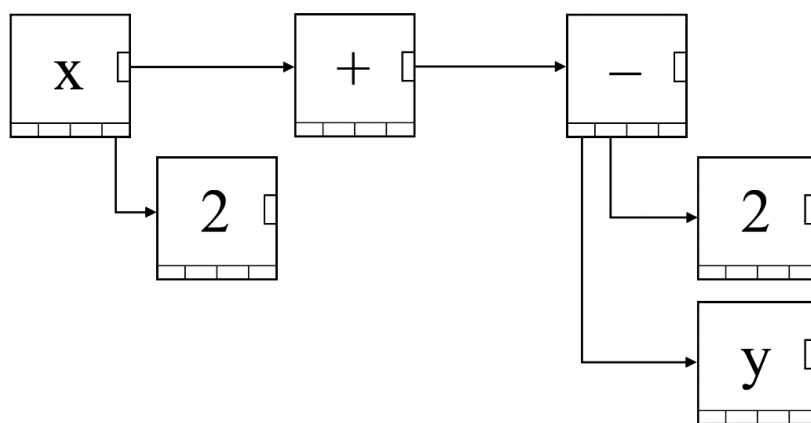


Рисунок 20.

Функция *layout_pass* возвращает самый первый стартовый символ – "x", который является корнем построенного дерева.

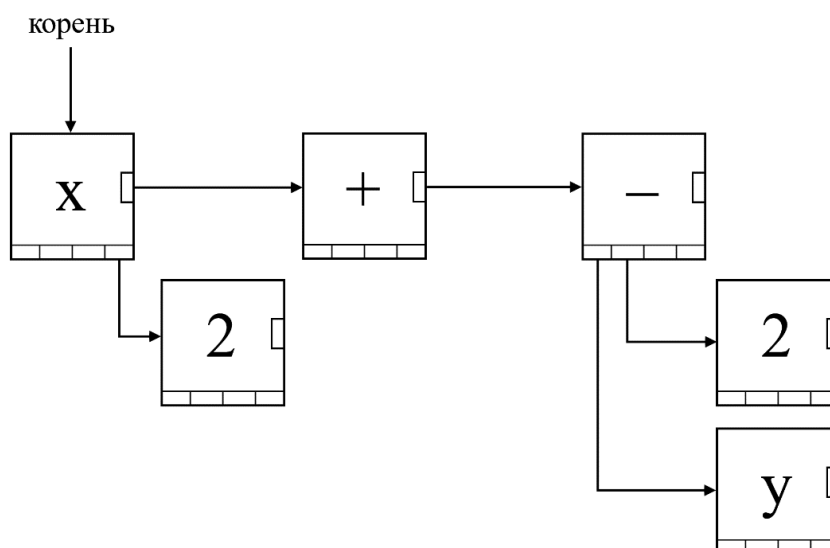


Рисунок 21.

2.5. РЕАЛИЗАЦИЯ TRANSFORM PASS

После построения Baselines Tree в Layout pass, алгоритм Transform pass вносит изменения в ВТ, используя определённый набор правил преобразования. Такие коррективы необходимы из-за того, что некоторые символы математического выражения могут трактоваться по-разному в зависимости от их расположения.

Правила преобразования описаны на рисунке 22. При проходе Baselines Tree на основе установленных правил распознаются шаблоны из отдельных символов или групп и производится модификация дерева базовых линий.

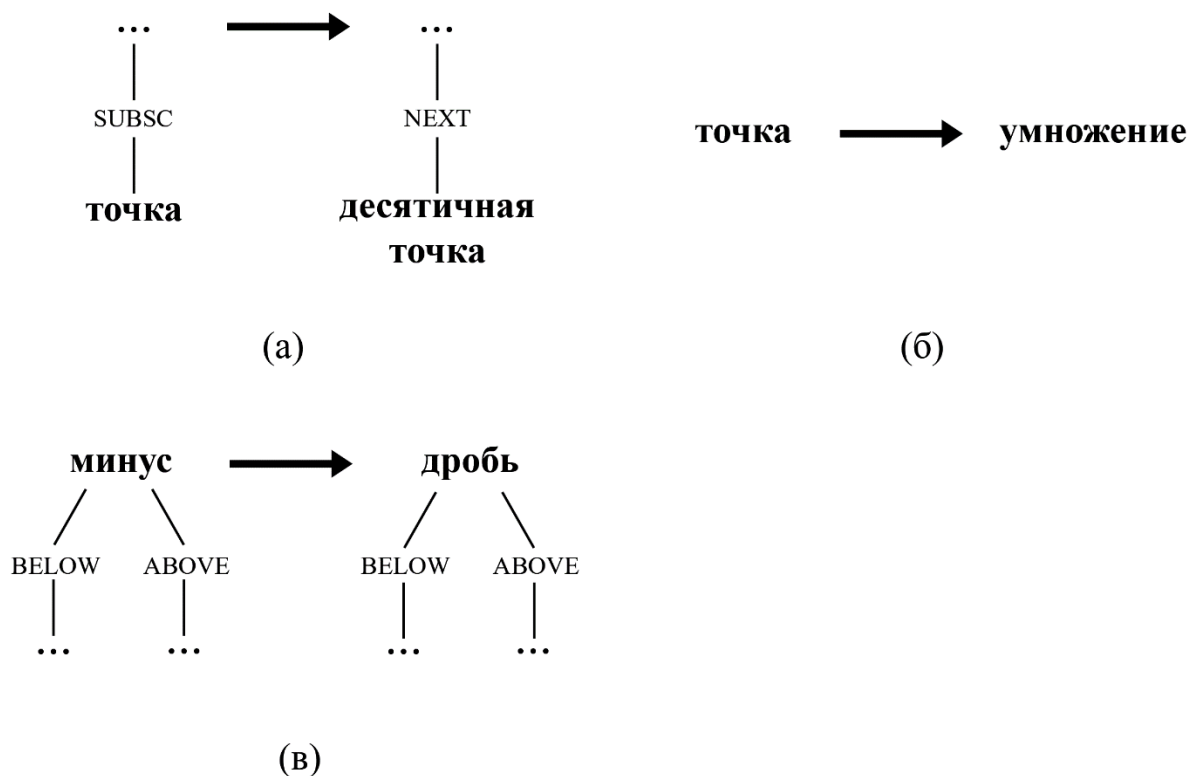


Рисунок 22. Правила преобразования ВТ

Проверка правил выполняется последовательно (от первого к последнему) для каждого символа. Так как десятичная точка алгоритмом Layout pass причисляется к индексу, то первое правило (рис. 22 пункт а) переносит десятичную точку из *subsc* региона в регион *next*. Это требуется для удобной обработки выражения в дальнейшем. Второе правило (рис. 22 пункт б) заменяет все остальные точки на символ умножения. Третье правило (рис. 22 пункт в) преобразует распознанный как минус символ в дробь, если над и под ним есть другие символы.

2.6. РАЗРАБОТКА МОДЕЛИ НЕЙРОННОЙ СЕТИ

С развитием нейронных сетей появилась возможность решать задачи, которые до этого считались труднорешаемыми или совсем нерешаемыми. Основными областями применения сетей являются: распознавание образов, прогнозирование, управление.

Наиболее подходящей для распознавания символов является **свёрточная нейронная сеть (СНС)** – специальная архитектура сети, которая была предложена Яном Лекуном в 1988 году [19].

СНС включает в себя несколько различных видов слоёв: свёрточные, субдискретизирующие и обычные – персептрон [20].

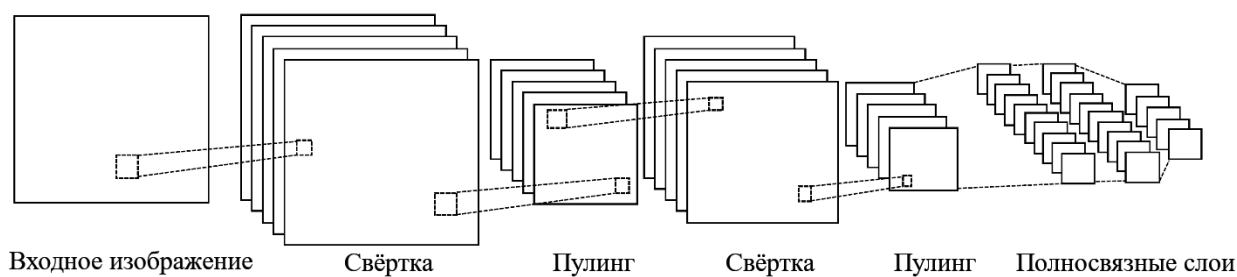


Рисунок 23. Пример СНС

На вход нейронной сети подаётся изображение в виде матрицы $N \times M \times 3$ из пикселей трёх цветов. Далее оно обрабатывается группой из свёрточных и субдискретизирующих слоёв.

Свёрточный слой включает в себя ядро – небольшая матрица (обычно 3×3 или 5×5), состоящая из весовых коэффициентов, которые устанавливаются в процессе обучения. Ядро проходит по всей входной матрице, поэлементно выполняя умножение с теми данными, над которыми оно находится в текущий момент, а затем суммирует полученные числа, получая один элемент выходной матрицы – матрицы признаков. Эта процедура продолжается до тех пор, пока ядро не пройдёт по всей входной матрице. В результате получается матрица, состоящая из взвешенных сумм входных признаков, причём каждый элемент этой матрицы расположен примерно в той же области матрицы, что и входные данные для него.

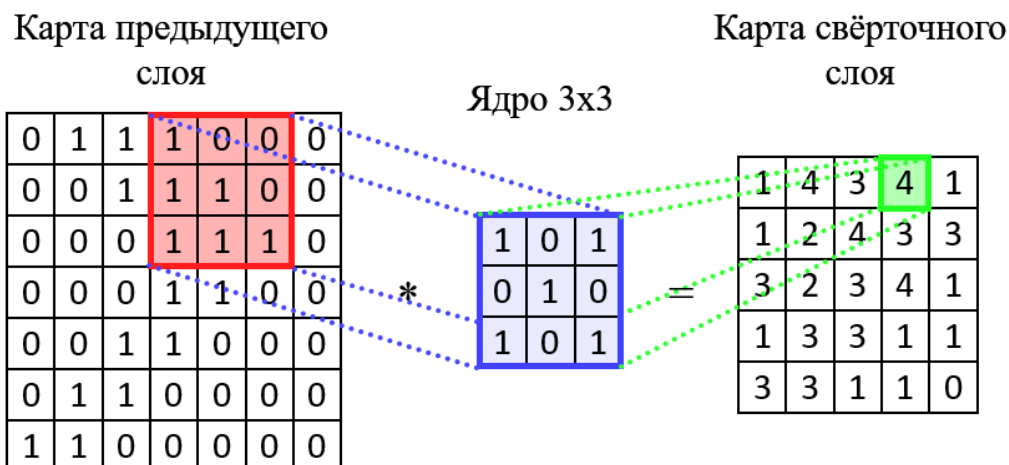


Рисунок 24. Пример работы свёрточного слоя

За свёрточным слоем может располагаться слой субдискретизации (слой пулинга\подвыборочный слой) целью которого является уплотнение карты признаков. Наиболее часто для этого применяется функция максимума. Обычно группы 2x2 пикселя уплотняются в один пиксель. Благодаря данному слою, существенно уменьшается размер изображения, но сохраняются наиболее важные признаки.

В конце сети находится полносвязный слой (персептрон). Он обрабатывает данные из предыдущего слоя и выдаёт вектор размерности N (количество классов). Элементы этого вектора выражают вероятность принадлежности к тому или иному классу. То есть полносвязный слой выполняет задачу классификации, используя для этого высокоуровневые признаки из предыдущих слоёв.

Таким образом, свёрточные нейронные сети обычно включают три типа слоёв: свёрточный, субдискретизирующий и полносвязный. Различные последовательности из этих слоёв порождают разные архитектуры сети, пригодные для определённых целей [21].

В проекте за основу топологии нейронной сети была выбрана модель LeNet-5 [22], которая была модифицирована для работы с изображениями размера 64x64 и большей точности распознавания.

Итоговая модель свёрточной сети состоит из трёх слоёв свёртки чередующимися с двумя слоями подвыборки. В конце расположены три полносвязных слоя.

Описанная модель представлена на рисунке:

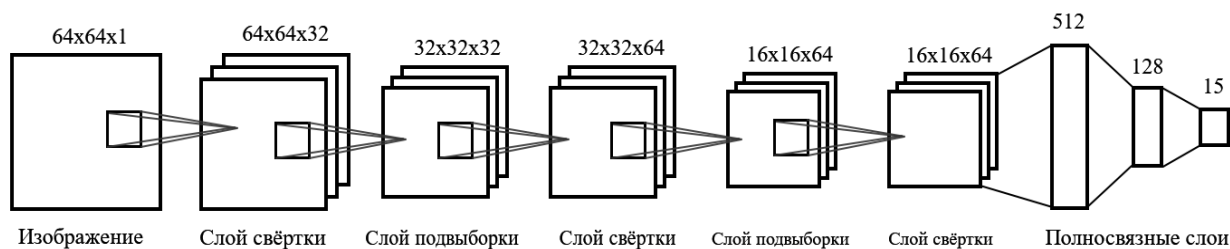


Рисунок 25. Топология свёрточной нейронной сети.

2.7. ФОРМИРОВАНИЕ ОБУЧАЮЩЕЙ ВЫБОРКИ

Для обучения свёрточной нейронной сети будет использоваться метод обучения с учителем. Суть которого заключается в том, что обучающая выборка – набор пар («объект», «ответ») подаётся на вход нейронной сети, задача которой найти зависимость между объектами и ответами. Для определения точности ответов применяется функция потерь, характеризующая насколько ответ свёрточной сети отклонён от правильного ответа для объекта x [23].

Анализ уже готовых обучающих выборок с математическими символами, доступных на сайтах Kaggle [24], ScienceDirect [25] и других, показал, что ни одна из них не включает нужный набор символов целиком. Также каждая выборка имеет свои особенности, в том числе масштаб картинок, поэтому совместить их не получится.

В связи с этим, было решено составить собственную базу из рукописных символов. Для удобства формирования выборки, символы, относящиеся к одному классу, изображаются в одном графическом файле, либо в нескольких файлах в одном каталоге. Разработанный для этих целей алгоритм, сканирует файловую систему, находит графические файлы и фрагментирует изображения на отдельные символы. Далее полученный массив данных сохраняется в один файл формата idx.

Обучающая выборка включает около 200 объектов каждого класса, изображённых в разных стилях. Этого достаточно для тестирования проекта и демонстрации возможностей. Но не хватает для охвата всевозможных написаний символов.

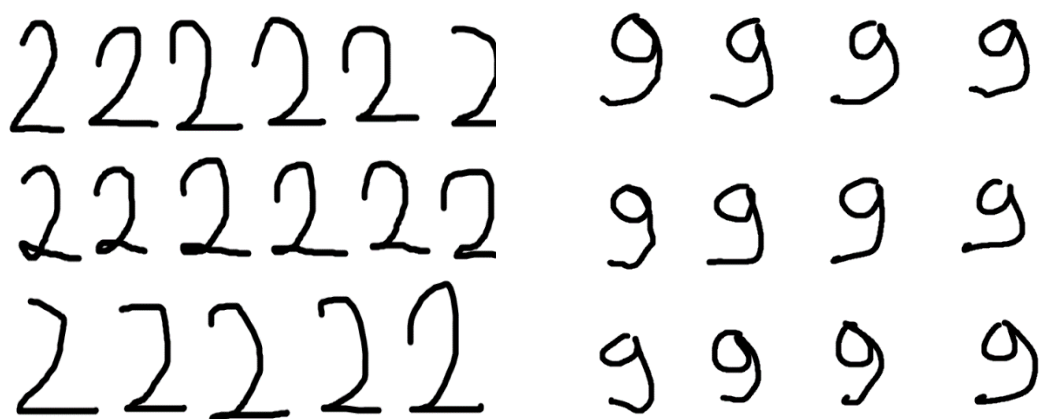


Рисунок 26. Пример символов «2» и «9» из составленного датасета.

Все классы символов обучающего набора представлены на рисунке:

0 1 2 3 4 5 6 7 8 9 + - () .

Рисунок 27.

Некоторые символы из датасета, в зависимости от контекста, могут иметь разное значение в математическом тексте. Например, символ «минус» может играть роль непосредственно минуса или роль дроби в выражении.

2.8. ВЫВОД ПО ГЛАВЕ

Данная глава посвящена проектированию и разработке сервиса. В ней описывается общая структура сервиса и требования к нему. Согласно проекту, сервис должен уметь обрабатывать изображение с одним математическим выражением и по нему строить дерево выражения. На текущем этапе разработки было решено включить для распознавания такие символы как цифры, основные математические операции, скобки и дроби.

Также в главе раскрыты теоретические детали реализации: алгоритм разбиения на символы, разбора выражения с использованием регионов символов, обработки дерева выражения. Подробно рассмотрена структура свёрточной нейронной сети и процесс формирования обучающей выборки.

ГЛАВА 3. ДОПОЛНИТЕЛЬНАЯ РАЗРАБОТКА И ТЕСТИРОВАНИЕ

3.2. ИНТЕГРАЦИЯ С WOLFRAM ENGINE

Задача разрабатываемого сервиса – генерирование дерева из символов выражения на основе картинке с математическим текстом. Но для демонстрации работоспособности проекта было решено подключить стороннее программное обеспечение – Wolfram Engine, представляющее из себя комплексную систему для высокоуровневых вычислений.

В нашем случае Wolfram Engine будет использоваться для нахождения значения выражения, которое распознал сервис. Но так как сервис формирует дерево выражения, а Wolfram работает с другими структурами данных, то требуется разработать алгоритм, преобразующий Baselines Tree в понятный для Wolfram вид – строку. Этим будет заниматься модуль Tree adapter.

Tree adapter содержит в себе основную функцию *adapt_to_solver(s:Symbol)*, которая рекурсивно обходит всё ВТ и формирует список из символов, который в последствии легко преобразовать в строку.

Как и в случае с Transform pass, данный алгоритм опирается на набор правил преобразования. Правила делятся на простые и комплексные. К простым относятся всего два правила – это правила обработки степени и индекса. Если у символа имеется, например, степень, то сначала добавляется к выходному списку “^”, затем рекурсивно обходятся символы степени и, наконец, добавляется закрывающаяся скобка. Тоже самое и для индексов.

Все остальные правила комплексные и обрабатываются отдельными функциями. Такое разделение на комплексные и простые правила было сделано в связи с особенностями синтаксиса и семантики Wolfram Language. В комплексных правилах можно оценивать символы выражения наперёд, чего нельзя сделать в простых. Это нужно в ситуациях, например, с выражением $\sum_{i=1}^{100} i^2$, которое в терминах Wolfram Language записывается так: *Sum[i^2, {i, 1, 100}]*. Это означает, что алгоритму нужно найти в Baselines Tree последний символ, относящийся к сумме, а после него вывести {i, 1, 100} и квадратную скобку.

Интеграция с Wolfram Engine

Чтобы иметь возможность взаимодействовать с Wolfram, был установлен пакет “wolframclient” для Python. Он предоставляет разработчику интерфейс для работы с локально установленным Wolfram Engine или с Wolfram Cloud.

Строка с выражением, полученная от Tree adapter, передаётся экземпляру класса Solver. Задача Solver – инициализировать сессию для работы и через функцию `solve(self, expr)` передавать строку с математическим текстом в Wolfram Engine.

Сессию можно инициализировать либо локальную (для этого нужно указать путь к ядру Wolfram), либо удалённую (потребуется указать ключ, полученный при регистрации). Было решено для тестирования проекта использовать удалённую сессию, так как в этом случае сервис легче разворачивать на новой машине (не требуется установка Wolfram Engine). Но удалённая сессия имеет ограничение на количество запросов.

Результат вычислений, получаемый от Wolfram, представлен либо строкой, либо комплексным объектом из библиотеки wolframclient, либо исключением. Функция `solver_output_to_str(data)` обрабатывает его, переводя любой вывод в строку.

3.3. РАЗРАБОТКА ИНТЕРФЕЙСА ДЛЯ ТЕСТИРОВАНИЯ

Чтобы иметь возможность тестировать работоспособность проекта, разрабатываемого как подключаемый сервис, необходимо дополнительно реализовать клиента – приложение, которое будет обращаться к сервису для распознавания рукописного выражения.

Приложение для тестирования было решено реализовать в виде простого графического редактора в котором можно изобразить математический текст. Для создания окна, в котором можно рисовать, применяется библиотека openCV.

Основной код программы включает в себя цикл и функцию для обработки событий от мышки (нажатие кнопок, передвижение).

В цикле происходит перехват нажатия кнопок клавиатуры. Всего «отлавливаются» три кнопки: «Escape» – для завершения работы программы, «c» – для очистки экрана и «r» – для распознавания нарисованного.

Функция `paint_draw` регистрируется как callback для событий, генерируемых мышкой. То есть вызывается при движении мышки или при нажатии на её кнопки. В `paint_draw` обрабатывается нажатие левой кнопки, чтобы пользователь имел возможность рисовать; правой кнопки – для стирания; передвижение мышки.

```
def paint_draw(event, former_x, former_y, flags, param):
    global current_former_x, current_former_y, mouse1_hold, mouse2_hold
    if event == cv2.EVENT_LBUTTONDOWN:
        mouse1_hold = True
    elif event == cv2.EVENT_RBUTTONDOWN:
        mouse2_hold = True
    elif event == cv2.EVENT_MOUSEMOVE:
        if mouse1_hold:
            cv2.line(image, (current_former_x, current_former_y), (former_x, former_y), (0, 0, 0), 5)
        if mouse2_hold:
            cv2.line(image, (current_former_x, current_former_y), (former_x, former_y), (255, 255, 255), 35)
            current_former_x = former_x
            current_former_y = former_y
    elif event == cv2.EVENT_LBUTTONUP:
        mouse1_hold = False
    elif event == cv2.EVENT_RBUTTONUP:
        mouse2_hold = False

    return former_x, former_y

image = 255 * np.ones((500, 800, 3), dtype=np.uint8)
cv2.namedWindow('Math-r Test')
cv2.setMouseCallback('Math-r Test', paint_draw)
while True:
    cv2.imshow('Math-r Test', image)
    k = cv2.waitKey(1) & 0xFF
    if k == 27: # Escape KEY
        break
    elif k == ord('c'):
        image[:] = 255
    elif k == ord('r'):
        # cv2.imwrite("tmp_expression.jpg", image)

    result = parse_image(image)
    result = image_parser_data_converter(result)
```

```
x = do_layout_pass(result)
do_transform(x)
str_simple = ''.join(layout_pass_to_list(x))
str_adapted = adapt_to_solver(x)
print(str_simple)
print(str_adapted)
print('result: ', solver_output_to_str(solver.solve(str_adapted)))
```

```
cv2.destroyAllWindows()
```

Алгоритмом вызываются сторонние функции: `parse_image`, `image_parser_data_converter`, `do_layout_pass`, `do_transform`. Эти функции предназначены для прямого взаимодействия с сервисом. Каждая из них представляет один соответствующий этап обработки.

3.4. РАЗРАБОТКА АЛГОРИТМА ПО АВТОМАТИЗАЦИИ ТЕСТИРОВАНИЯ

На разных этапах разработки любой программы всегда уделяется большое внимание её тестированию. При внесении серьёзных изменений в алгоритмах, желательно тестировать не только то, на что были направлены эти изменения, но и остальной функционал приложения. Потому что в частных случаях может поменяться поведение компонентов программы, опирающихся на код, который был подвергнут изменению.

Чтобы при каждой внесённой правке, при каждой модификации сервиса не производить вручную (при помощи описанного в пункте 3.2 инструмента) тестирование всех его возможностей, было решено написать модуль и подготовить материал для автоматического тестирования.

Задача разработанного модуля по автоматическому тестированию – проверить сервис на заранее заданных данных. Для этого были подготовлены изображения, содержащие по одному математическому выражению. Каждое выражение тестирует одну или несколько возможностей сервиса.

В коде данные для тестирования представляются так:

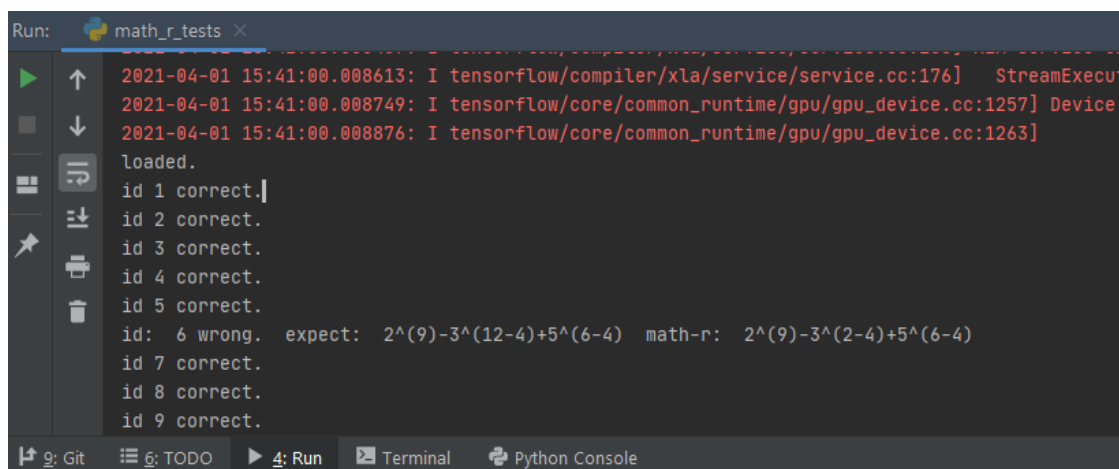
```
tests_data = [  
    {'id': 1, 'answer': '1234567890', 'expression_path': 'test_images/1.png'},  
    {'id': 2, 'answer': '3-4+8-0', 'expression_path': 'test_images/2.png'},  
    {'id': 3, 'answer': '(8)/(2)', 'expression_path': 'test_images/3.png'},  
    # И т.д.  
]
```

Сам алгоритм тестирования заключается в инициализации сервиса, а затем в последовательной подаче изображений сервису. Сервис после обработки возвращает результат, сравниваемых с заранее готовым ответом. Результат каждого теста выводится на экран.

```
if __name__ == "__main__":  
    prepare_network()  
  
    for test in tests_data:  
        math_r_answer = adapt_to_solver(math_recognition_image_by_path(test['expression_path']))  
        if math_r_answer == test['answer']:  
            print('id', test['id'], 'correct.')  
        else:  
            print('id: ', test['id'], 'wrong. ', 'expect: ', test['answer'], ' math-r: ', math_r_answer)
```

3.5. ТЕСТИРОВАНИЕ ПРОЕКТА

При разработке сервиса, производилось его поэтапное тестирование сначала при помощи графического интерфейса, описанного в пункте 3.3, а затем с помощью автоматических тестов. После каждого тестирования, в случае необходимости, вносились коррективы в код проекта.



```
Run: math_r_tests x  
2021-04-01 15:41:00.008613: I tensorflow/compiler/xla/service/service.cc:176] StreamExecu  
2021-04-01 15:41:00.008749: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1257] Device  
2021-04-01 15:41:00.008876: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1263]  
loaded.  
id 1 correct.  
id 2 correct.  
id 3 correct.  
id 4 correct.  
id 5 correct.  
id: 6 wrong. expect: 2^(9)-3^(12-4)+5^(6-4) math-r: 2^(9)-3^(2-4)+5^(6-4)  
id 7 correct.  
id 8 correct.  
id 9 correct.
```

Рисунок 28. Скриншот результатов автоматического тестирования

Финальная версия сервиса показала хорошие результаты на составленных тестах. А также она была протестирована несколькими пользователями.

Результаты тестирования показали, что в большинстве случаев при аккуратном написании выражения, оно правильно распознаётся и обрабатывается. Но при небрежном вводе символов, допускаются ошибки, в основном, в распознавании символов. Это означает, что в будущем следует поработать над распознаванием символов: составить более обширный датасет, охватывающий как можно больше стилей написания; попробовать изменить модель свёрточной сети.

3.6. ВЫВОД ПО ГЛАВЕ

В данной главе описывается процесс интеграции проекта с Wolfram Engine, а также его тестирование.

Так как, согласно проекту, разрабатываемый сервис не имеет пользовательского интерфейса и предоставляет результаты в виде дерева выражения, то для проверки работоспособности в связке со сторонними продуктами было подключено программное обеспечение Wolfram Engine.

Для тестирования непосредственно самого сервиса, был разработан пользовательский интерфейс для графического ввода выражений. А также придумано и реализовано средство автоматического тестирования. Благодаря тестированию, были обнаружены баги в коде и в логике проекта, незамеченные в процессе разработки. Их выявление позволило переработать программный код и исправить проблемы.

ЗАКЛЮЧЕНИЕ

В результате выполнения дипломной работы был спроектирован и разработан сервис по обработке рукописных математических выражений, работающий по следующему алгоритму:

1. Пользователь передаёт изображение с математическим текстом с помощью API.
2. Изображение обрабатывается, анализируется и разбивается на отдельные символы
3. Символы распознаются с помощью свёрточной нейронной сети.
4. Алгоритм Layout pass собирает символы в соответствии с их значением и положением в структуру данных – дерево выражения.
5. Производится пост-обработка дерева.
6. Сервис возвращает пользователю структуру – дерево, описывающее входное выражение.

При работе над проектом, я изучила теорию и принцип работы нейронных сетей, в частности, свёрточных; получила опыт в разработке проектов-сервисов; изучила новый язык программирования – Python; разобралась с готовыми решениями для создания нейронных сетей.

Несмотря на то, что все поставленные задачи были выполнены, у сервиса есть масса возможных путей развития, которые можно реализовать на следующих итерациях разработки. Основное направление развития – добавление новых математических символов для распознавания, например, латинских букв для обозначения переменных, оператора суммирования, интегрирования и т.д. Добавлять новые символы несложно благодаря простой и продуманной структуре проекта.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Лутц М. Изучаем Python. 2019. 125-165, 343-410 pp.
2. Хайкин С. Нейронные сети. Полный курс. 2006. 31, 172 pp.
3. Optical character recognition. [Электронный ресурс] URL: https://en.wikipedia.org/wiki/Optical_character_recognition
4. Сайт проекта InftyReader. [Электронный ресурс] URL: <https://www.inftyreader.org/>
5. Сайт Mathematical Expression Recognition. [Электронный ресурс] URL: <http://cat.prhlt.upv.es/mer/>
6. Photomath. Официальный сайт. [Электронный ресурс] URL: <https://photomath.com/ru/>
7. Сайт проекта MyScript. [Электронный ресурс] URL: <https://www.myscript.com/ru/>
8. Сайт приложения Mathpix. [Электронный ресурс] URL: <https://mathpix.com/>
9. Официальный сайт Python. [Электронный ресурс] URL: <https://www.python.org/>
10. Python documentation. [Электронный ресурс] URL: <https://docs.python.org/3/>
11. Официальный сайт PyCharm. [Электронный ресурс] URL: <https://www.jetbrains.com/ru-ru/pycharm/>
12. Официальный сайт проекта Keras. [Электронный ресурс] URL: <https://keras.io/>
13. Официальный сайт Git. [Электронный ресурс] URL: <https://git-scm.com/>
14. Официальный сайт OpenCV. [Электронный ресурс] URL: <https://opencv.org/>
15. Официальный сайт Wolfram Engine. [Электронный ресурс] URL: <https://>

www.wolfram.com/engine/

16. Richard Zanibbi, Dorothea Blostein, James R. Cordy. Recognizing Mathematical Expressions Using Tree Transformation. // IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 24, No. 11, Ноябрь 2002.
17. Gerald Friedland, Lars Knipping, Ernesto Tapia. Web Based Lectures Produced By AI Supported Classroom Teaching. // International Journal on Artificial Intelligence Tools, 2004.
18. Ernesto Tapia. Understanding Mathematics: A System for the Recognition of On-Line Handwritten Mathematical Expressions. Berlin. 2004.
19. Convolutional neural network. [Электронный ресурс] URL: https://en.wikipedia.org/wiki/Convolutional_neural_network
20. Что такое свёрточная нейронная сеть. [Электронный ресурс] URL: <https://habr.com/ru/post/309508/>
21. Свёрточные нейронные сети для распознавания образов. [Электронный ресурс] URL: <https://habr.com/ru/post/456186/>
22. Описание свёрточной нейронной сети LeNet-5. [Электронный ресурс] URL: <https://en.wikipedia.org/wiki/LeNet>
23. Supervised learning. [Электронный ресурс] URL: https://en.wikipedia.org/wiki/Supervised_learning
24. Kaggle. [Электронный ресурс] URL: <https://www.kaggle.com/>
25. ScienceDirect. [Электронный ресурс] URL: <https://www.sciencedirect.com/>