DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

EE492 PROJECT MIDTERM REPORT

Recognition and Mathematical Operation System for Handwritten Expressions

Ender Efe ERYEGÜL – 280206062

Bartu ERDAL - 280206061

Mahmut Cenk EFELER

DATE: 08/05/2025

I. PROJECT STATEMENT

The primary objective of the project is to execute mathematical operations on a given handwritten mathematical expression by recognizing the expression using a deep learning model. The main system of the project is designed as a pipeline of two main subsystems: The image processing and the mathematical processing system. These systems operate sequentially. The input of the entire system directly serves as the input for the image processing subsystem. After the processes in the first subsystem, the output of it enters the next subsystem, mathematical processing, as an input. Following that, the mathematical processing system gives an output as desired by the user that varies as symbolical and numerical differentiation, numerical integration and more. Afterwards, the subsystem's output serves as an output for the whole system.

In the following sections, the overall structure of the subsystems will be examined under several titles.

A. The Image Processing System

The system begins by taking a handwritten expression image as an input and applying image processing techniques such as contour detection based on OpenCV to segment the characters. These characters are then passed through a trained convolutional neural network using that is implemented in PyTorch to classify each character. After each character is classified, an algorithm takes a list of the classified characters as an input where each node contains the coordinates and scale information of the characters. This algorithm uses several methods (e.g X, Y and Z) to convert the list to a string format, which becoming the output of the system.

a. Character Segmentation

The first step of image processing system is to segmentate the input image into individual character images. In the image processing system, each character is classified individually, thus, reliable segmentation is crucial for this project. For this task, we took advantage of OpenCV library.

OpenCV (Open Source Computer Vision Library) is a widely used open-source library designed for computer vision applications such as our project. OpenCV offers comprehensive tools for image and video processing. Due to its high performance and flexibility, OpenCV has become a popular choice in both academic and industrial applications. Although the library was written in C/C++, it provides its features for other languages like Python and Java. For these reasons, we decided to use OpenCV library for most of our image operations in our project.

b. Character Classification

c. Output Representation

B. The Mathematical Processing System

The system takes the output of the previous subsystem as an input. The system has several of modules (e.g Tokenizer, PostFixExtractor, MathOps) and uses some data structures (e.g Binary tree, stack, queue). The postfix expression in a stack gets converted to a binary tree. The reason we chose binary tree is its capability of fragmenting and reconstructing its own structure. The leftchild and the rightchild of any node can be extracted, subjected to a mathematical operation and reinserted into the correct position within the structure.

a. **Tokenization**
b. **Postfix Conversion and Tree Construction**
c. **Mathematical Operations Module**

II. CURRENT STATUS

The base Token class and its three subclasses (FunctionToken, OperatorToken and NumberToken) have been successfully implemented. Each subclass contains specific attributes:

FunctionToken represents mathematical functions such as sin, cos.

OperatorToken includes metadata like operator type (prefix, suffix) and precedence level.

NumberToken stores numerical values and flags whether the value is constant or not.

Each of the three classes passed independent testing and verification to guarantee proper operation, including content search, operator type and precedence checks and constant identification. This object-oriented token structure serves as the basis for efficient mathematical statement processing and analysis.

The tokenizer module has been implemented in Python. It can successfully parse mathematical expressions into structured tokens such as numbers, operators and function identifiers. Common mathematical functions (such as sin, cos, tan, log), constants conventional arithmetic operations are all supported by the implementation, which also makes the difference between various operator types (such as prefix, suffix, left-associative and right-associative).

Mathematical operations, trigonometric functions, negative numbers, variables, factorials, percentages, square roots and power operations have all been tested using a variety of expressions. Every tested scenario shows the tokenizer acting as it should.

Work is now being done to improve error handling for incorrect inputs and further refine function parsing, particularly for multi-character functions and variable names.

Test Results

1. Fundamental Arithmetic Functions

"2 + 3 * 4" was successfully processed by the tokenizer, which correctly identified the integers and operators while preserving the correct precedence. Accurate operator hierarchy handling was demonstrated by processing addition with precedence 2 after multiplication with precedence 3.

2. Functions of Trigonometry

The system properly identified both trigonometric functions and their arguments in the tokenized expression "sin(30) + cos(60)". For the addition between functions, the tokenizer kept the correct operator precedence and handled the parenthesis correctly.

3. Negative Values

Excellent handling of negative numbers in the equation "-5 + 3" was displayed by the tokenizer. It processed the negative integer correctly while preserving the right operator precedence for the next addition, correctly identifying the minus sign as a prefix operator.

4. Complicated Words

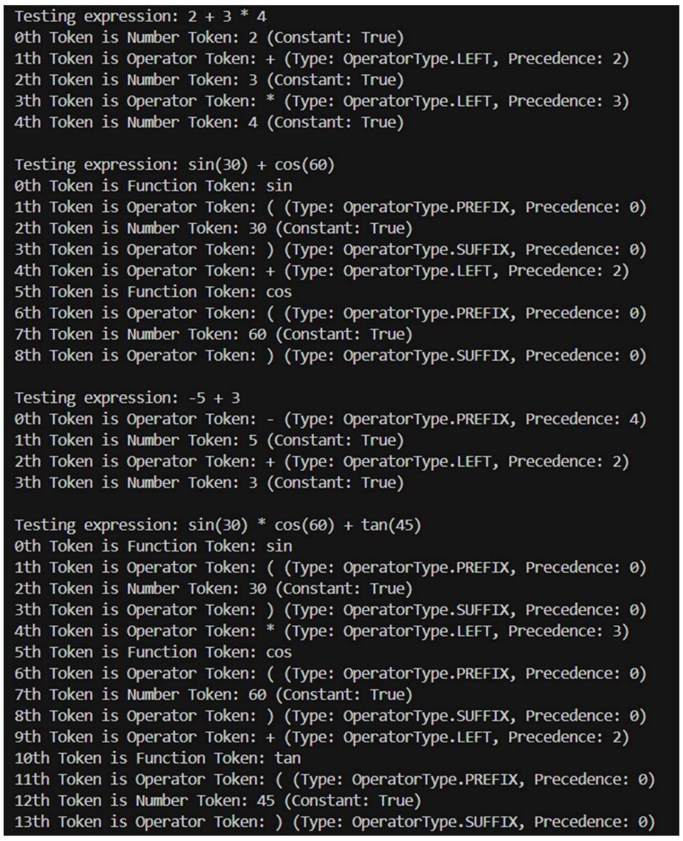
The complex equation "sin(30) * cos(60) + tan(45)" was successfully processed by the system, demonstrating its capacity to manage several trigonometric functions in a single expression. The tokenizer processed all function inputs correctly while maintaining the correct operator precedence between addition and multiplication.

5. Elements

The tokenization of the expression "x + y * z" was accurate, and the algorithm correctly recognized variables as non-constant values. The tokenizer proved its capacity to handle algebraic expressions by correctly differentiating between variables and operators while maintaining accurate operator precedence.

6. Special Operators

The tokenizer showed strong capabilities in handling special operators, successfully processing square roots (√(16) + 4), factorials (5! + 3), percentages (50% of 200), and power operations (2^3 + 4^2). Although the majority of special operators were handled successfully, there are a few small problems that require solving, such as the "of" keyword in percentage expressions and certain missing tokens in power operations.

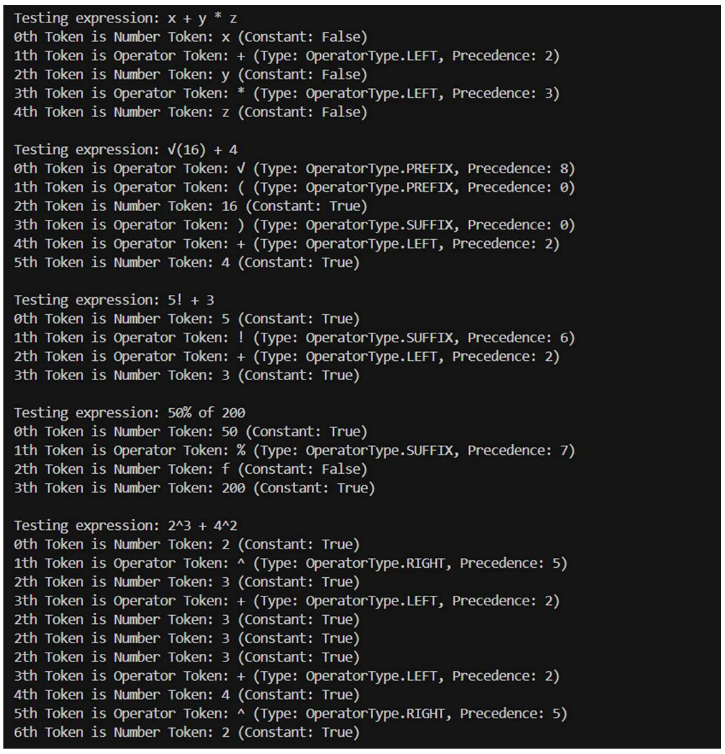


```
Testing expression: 2 + 3 * 4
0th Token is Number Token: 2 (Constant: True)
1th Token is Operator Token: + (Type: OperatorType.LEFT, Precedence: 2)
2th Token is Number Token: 3 (Constant: True)
3th Token is Operator Token: * (Type: OperatorType.LEFT, Precedence: 3)
4th Token is Number Token: 4 (Constant: True)

Testing expression: sin(30) + cos(60)
0th Token is Function Token: sin
1th Token is Operator Token: ( (Type: OperatorType.PREFIX, Precedence: 0)
2th Token is Number Token: 30 (Constant: True)
3th Token is Operator Token: ) (Type: OperatorType.SUFFIX, Precedence: 0)
4th Token is Operator Token: + (Type: OperatorType.LEFT, Precedence: 2)
5th Token is Function Token: cos
6th Token is Operator Token: ( (Type: OperatorType.PREFIX, Precedence: 0)
7th Token is Number Token: 60 (Constant: True)
8th Token is Operator Token: ) (Type: OperatorType.SUFFIX, Precedence: 0)

Testing expression: -5 + 3
0th Token is Operator Token: - (Type: OperatorType.PREFIX, Precedence: 4)
1th Token is Number Token: 5 (Constant: True)
2th Token is Operator Token: + (Type: OperatorType.LEFT, Precedence: 2)
3th Token is Number Token: 3 (Constant: True)

Testing expression: sin(30) * cos(60) + tan(45)
0th Token is Function Token: sin
1th Token is Operator Token: ( (Type: OperatorType.PREFIX, Precedence: 0)
2th Token is Number Token: 30 (Constant: True)
3th Token is Operator Token: ) (Type: OperatorType.SUFFIX, Precedence: 0)
4th Token is Operator Token: * (Type: OperatorType.LEFT, Precedence: 3)
5th Token is Function Token: cos
6th Token is Operator Token: ( (Type: OperatorType.PREFIX, Precedence: 0)
7th Token is Number Token: 60 (Constant: True)
8th Token is Operator Token: ) (Type: OperatorType.SUFFIX, Precedence: 0)
9th Token is Operator Token: + (Type: OperatorType.LEFT, Precedence: 2)
10th Token is Function Token: tan
11th Token is Operator Token: ( (Type: OperatorType.PREFIX, Precedence: 0)
12th Token is Number Token: 45 (Constant: True)
13th Token is Operator Token: ) (Type: OperatorType.SUFFIX, Precedence: 0)
```



```
Testing expression: x + y * z
0th Token is Number Token: x (Constant: False)
1th Token is Operator Token: + (Type: OperatorType.LEFT, Precedence: 2)
2th Token is Number Token: y (Constant: False)
3th Token is Operator Token: * (Type: OperatorType.LEFT, Precedence: 3)
4th Token is Number Token: z (Constant: False)

Testing expression: √(16) + 4
0th Token is Operator Token: √ (Type: OperatorType.PREFIX, Precedence: 8)
1th Token is Operator Token: ( (Type: OperatorType.PREFIX, Precedence: 0)
2th Token is Number Token: 16 (Constant: True)
3th Token is Operator Token: ) (Type: OperatorType.SUFFIX, Precedence: 0)
4th Token is Operator Token: + (Type: OperatorType.LEFT, Precedence: 2)
5th Token is Number Token: 4 (Constant: True)

Testing expression: 5! + 3
0th Token is Number Token: 5 (Constant: True)
1th Token is Operator Token: ! (Type: OperatorType.SUFFIX, Precedence: 6)
2th Token is Operator Token: + (Type: OperatorType.LEFT, Precedence: 2)
3th Token is Number Token: 3 (Constant: True)

Testing expression: 50% of 200
0th Token is Number Token: 50 (Constant: True)
1th Token is Operator Token: % (Type: OperatorType.SUFFIX, Precedence: 7)
2th Token is Number Token: f (Constant: False)
3th Token is Number Token: 200 (Constant: True)

Testing expression: 2^3 + 4^2
0th Token is Number Token: 2 (Constant: True)
1th Token is Operator Token: ^ (Type: OperatorType.RIGHT, Precedence: 5)
2th Token is Number Token: 3 (Constant: True)
3th Token is Operator Token: + (Type: OperatorType.LEFT, Precedence: 2)
2th Token is Number Token: 3 (Constant: True)
2th Token is Number Token: 3 (Constant: True)
2th Token is Number Token: 3 (Constant: True)
3th Token is Operator Token: + (Type: OperatorType.LEFT, Precedence: 2)
4th Token is Number Token: 4 (Constant: True)
5th Token is Operator Token: ^ (Type: OperatorType.RIGHT, Precedence: 5)
6th Token is Number Token: 2 (Constant: True)
```

III. PLANNED PROJECT TASKS

In this section the work which is expected to be done until the end of the project study should be written.

Also, you **must** *provide the work plan* by filling out the table below with itemized project activities and mark their duration on the timeline to the right by shading the corresponding weeks. Add rows as needed.*

Project work items						**Weeks**						
	3	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
… (Specific Aim 1)												
…												
…												
…												
…												
…												

*As of 25 April 2025, we are in the 9th week. Also, fill out the tasks that have been accomplished in the past weeks.

REFERENCES