Computer And System Engineering Department
Faculty Of Engineering
Alexandria University

Handwritten Arabic Math Solver

Abobakr Abdelaziz Ahmed

Omar Shawky Abdelsalam

Elsayed Akram Elsayed

Abdelrhman Ibrahim Mostafa

Fares Medhat Helmy

Muhammed Yaser Mahmoud

Supervised by: Dr. Marwan Torki

Contents

Abstract

The recent educational trends in Egypt towards digitization have brought to the sight the usage of tablets with a stylus as an essential educational tool, emphasizing the importance of new technologies in the modern learning era.

Arabic mathematical notation is the method that is widely used in the Middle East; however, it's hardly implemented in the software domain and there is almost no practical implementation for it.

Here we found that it would be beneficial to build a comprehensive software that recognizes, analyses, and solves mathematics written in the Arabic mathematical notation.

The expected outcome from this is not only helping Egyptian students but also any learner or teacher of mathematics in the Middle Eastern region. Who still learn/teach mathematics in Arabic i.e primary stage, El-Azhar, and so on.

Acknowledgement

We are sincerely grateful to our supervisor Dr. Marwan Torki, who gave us faithful encouragement, technical support, and guidance, particularly in the field of computer vision, and helped us with many technical problems in our project.
Also, he guided us to focus on the big picture of the problem and not on tiny details that Could stop our progress, which in turn pushed us to complete this problem at the stage.

Chapter 1

Introduction

1.1 General

Throughout the school grades, the mathematical tools and applications were very essential to all students to help them to solve all kinds of problems starting with simple calculations such as addition, subtraction, multiplication, and division till higher-level problems like integration and differentiation.
In our period we simply used calculators to solve those simple kinds of calculations. But nowadays the recent educational trends in Egypt and most Arabic countries go towards the usage of tablets with a stylus as an essential educational tool. We have searched for any mobile application that provides the feature of writing numbers, symbols, and equations by hand then classifying them and showing the answer with steps and analysis of the final answer with graphs and so on. We didn't find any application for Arabic handwritten mathematics.

1.2 Motivation

Since there are no applications that provide solutions for handwritten Arabic equations, this motivated us to begin this project from scratch nevertheless there is no open-source projects before us, no dataset and no previous graduation projects talk about this particular problem.

We searched for English applications that are similar to what we need to implement; we found out that Microsoft has such an application -Microsoft Math Solver- but unfortunately, this application deals only with English characters and English numbers, also this application is closed-source.
Their application is very good, usable, and very friendly, it recognizes a character once you wrote it, concatenates this character to the previously written equation and if you undo or clear the character, this character will pop out from the written equation. After you solve the problem the application will show you a graph of the equation and some fancy things.
Microsoft math solver inspired us to do something close to it, at least we hoped so, this

application had many features such as draw, undo, redo, pen, eraser, delete, taking photos from the camera as input, taking keyboard typed equation as input and provides links related to the problem such as videos from khan academy and so on...

We also have found some open-source implementations that were similar to MS's application but were also for English handwritten problems, we spent time analyzing those open-source projects to benefit from some things that were common like their models that were responsible for classifying characters, how their segmentation works, how they solve the equation after converting it to string and which libraries they've used to accomplish this task.

1.3 Goals

- Real-time recognition of handwritten mathematical expressions in Arabic notation.
- Having an application that is able to solve many kinds of problems, such as :
    1. Simple calculation (addition , subtraction , multiplication, division).
    2. Finding roots of polynomial equations.
    3. Differentiation.
    4. Integration.
    5. Solving complex mathmatical functions (Logarithms, Trigonometry, Limits, etc..)
- Having an application that achieves high usability.

1.4 Scope

Our scope of users that will benefit from this problem is very wide since this application is considered an educational tool for more than one grade, the following represents the scope of who will benefit from this application:

1. Primary grades: nowadays the Egyptian government decided that all grades will use tablets in the schools so this application will be a good idea for young students to train themselves by learning math using this application.
2. Teachers: it is very essential that the teachers have the knowledge of how to deal with online applications like our application to solve problems using tablets rather than traditional ways.
3. Out of Egyptian scope: any student in the world who need to learn mathematics in the Arabic notation for any reason.
4. Schools: general schools can use our product for learning purposes if this application can be improved and some extra features added to it, the version can be premium and schools can buy it.

1.5 Difficulties

Since there are no applications that provide solutions for handwritten Arabic equations, this made us face many difficulties, such as:

1. Arabic math letters dataset: the dataset was limited, this forced us to make the dataset by ourselves and this took a long time because we wrote different handwritten shapes for each letter.
2. It was hard to find someone who works on Arabic handwritten equations' recognition, also what was found, in the Arabic handwritten recognition field of research, was working on only simple operations, like adding and subtracting two numbers.
3. We thought that we would find a wide previous work with English equations and we will use what we found to improve and convert it to solve our problem, unfortunately, there were few ones who work on that problem.

Chapter 2

Background and Related Work

2.1 Bangla Handwritten Math Recognition and Simplification Using CNN[1]

This paper represents a system which takes an input image of Bangla handwritten mathematical expression, simplifies the problem and generates the answer as an output.
This was done mainly using 4 steps :

1. **Image Preprocessing**
2. **Line Segmentation**
3. **Character Segmentation**
4. **Classification and Recognition**.

(1) Image Preprocessing: this step's aim was to preprocess the RGB image into a gray-scale image with text holding value 255, and empty space holding value 0, using the **BINARY_INV** method, this helps in eliminating noise and helps in a better segmentation process.

(2) Line Segmentation: this step's aim was to segment a multi-line image into only one line that holds the equation to be solved, this was done by iterating horizontally in the image, if the pixel value is 255 (white pixel), it means it is a text. The White pixel value of 255 is considered text. If the sum of the horizontal row is 0, it means it is a black row, and it is considered a gap between two lines.

(3) Character Segmentation: the aim of this step was to segment the one-line equation into single characters by summing the vertical pixels in a column, if the accumulated sum was 0, then it is called a gap and suspect as a character.

(4) Classification and Recognition: the aim of this step was to classify single characters and concatenate classified characters into one string to be solved as a mathematical

expression.

The model they used was **MathNet**.

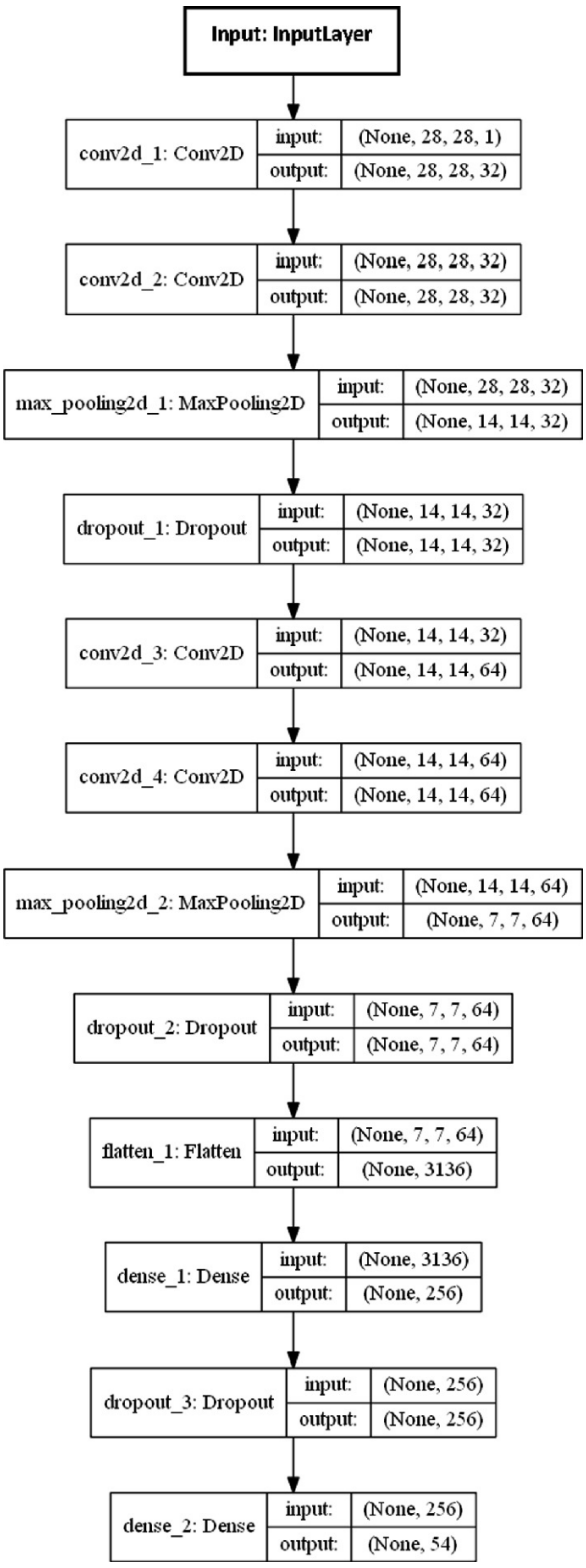


Figure 2.1: Model architecture from paper

In this paper, **the recognition accuracy of Bangla characters is 96.5%**.

While this process was very similar to our steps, it had some limits that they couldn't solve:

1. They could only process equations if they were written horizontally in one single line, if the written equation was multi-line (contains powers, Logarithms, etc....), they wouldn't be able to process this correctly.
2. If two symbols were attached (not separated by space), they would be classified as one symbol instead of 2 separated symbols.

2.1.1 What we learnt

- We used a smaller derivative of their classification model (**MathNet**), but we used less layers, as our training data was limited and the original model was over-fitting it.
- We began to consider how to circumvent their first limitation, which prompted us to look into how to solve multi-line mathematical equations. As a result, we were able to solve equations containing powers, logarithms, and other complex structures.

2.2 A Syntax Directed System for the Recognition of Printed Arabic Mathematical Formulas[2]

The paper discussed the problem of recognizing Arabic mathematical formulas.

For **symbol recognition**, we learned that we may use a recognition method based on K* with a combination of different statistical features.

For **formulas analysis**, we learned that we may use a top-down and a bottom-up parsing algorithm based on symbol dominance and we saw the ability of the proposed system to handle complex mathematical formulas containing implicit multiplication, subscripts and superscripts.

The **bottom-up parser** begins by looking for the dominant operator, as explained above. Then, it chooses the corresponding rule in the grammar, considering the operator contexts.

This rule provides instructions to the **top-down parser** to partition the formula into sub-formulas which are analyzed by the same way and so on until analyzing the whole of the formula.

We also saw how the system offers the possibility to detect and correct some symbol recognition errors during the different levels of formulas analysis process.

In this paper the **recognition accuracy is 89.8%**.

2.3 Dynamic Random Forest for the Recognition of Arabic Handwritten Mathematical Symbols with A Novel Set of Features[3]

This paper was one of the few first papers that addressed our problem specifically, and it contained the core components for solving it which are:

1. segmentation of the mathematical expression into isolated symbols.
2. recognition and classification of these symbols.
3. structural analysis that determines the relationships between the symbols.

The paper focused on the second component which is the recognition of the Arabic handwritten mathematical symbols. The difficulties of this step as introduced in the paper were:

- The variety of the symbols like Arabic and Latin numerals, Arabic alphabet, arithmetic symbols, etc.
- The similarity between different symbols.
- The similarity between Arabic alphabets differs in the position and number of dots.
- The ambiguity in human handwriting and how this affects the above 2 points.
- Some operators are stretched in Arabic notation like square root.

Then the paper discussed the related work to the problem which was extensively done on the Latin mathematical expressions rather than Arabic ones, for example; Adaboost, SVM, and Random Forest classifier were employed by Davila et al.[4], a Hidden Markov Models (HMM), and Bidirectional Long Short Term Memory (BLSTM) for online handwritten mathematical symbols were proposed by Álvaro et al.[5].

Their proposed solution is **Dynamic Random Forest for Arabic Mathematical Symbols Recognition with Hybrid Features** consists of a normal preprocessing step that includes noise removal, grayscale normalization, etc. Then the extracting features step and finally training using random forests which is an ensemble algorithm that uses multiple decision trees, where each tree contributes with a single vote for the assignment of the most frequent class.

In the paper, HAMF[6] dataset is introduced which is a dataset that is freely available that consists of 4238 images of handwritten Arabic mathematical formulas and 20300 isolated symbols images. This is the dataset that is used for the evaluation in the paper and it was one of the datasets that helped us during our project.

2.3.1 What we learnt

This paper reminded us how complex is the problem we are solving and how each component alone is complex enough to be a project of its own, nonetheless, we want to apply an operation on the recognized expression and get a result out of it on an android phone, which seems infeasible given the limited time and the scale of the project. As a result, there are compromises to be made for each component and to work on them in a balanced fashion such that it would be a cornerstone for anyone to build on our work.

These compromises include but are not limited to:

- using Arabic alphabet and function names without dots.
- using a CNN model that's easy to be trained which has been possible by the above point.
- using HTML rather than LATEX or MathML as an encoding form due to the limitations of Android and the lack of support for Arabic mathematical expressions on it.

2.4 Offline Handwritten Mathematical Symbol Recognition Utilizing Deep Learning[7]

The paper discussed how to do symbol recognition in English handwritten mathematics. We used this paper to help us to see other ways to recognize Arabic handwritten mathematical symbols.

In this paper, symbol recognition is divided into four parts: limitations of symbol recognition, segmentation, symbol categorization, classifiers based on HOG, LBP, salient, and intensity).

(A) **Limitations of symbol recognition**: Symbol recognition process, regardless of being online or offline, has some limitations that do not allow achieving high accuracy.

(B) **Segmentation**: In order to obtain accurate results from mathematical symbol classification, a multi-step segmentation module is required.
These steps include block segmentation, line segmentation, line classification as three classes of text only, embedded math and math-only classes, word segmentation, and symbol segmentation. Line classification is to classify mathematical multi-dimensional lines, linear text lines, and mixed lines.

(C) **Symbol categorisation**: Symbol categorisation has been performed based on the symbol's aspect ratio. A symbol aspect ratio is the ratio of height to width.

(D) **Classifiers based on HOG, LBP, Salient, and Intensity**: Although this method has reasonable accuracy for a small number of classes, it can be expected to keep the accuracy high when the number of classes is more than 300 due to the limitations of symbol recognition. To obtain the most discriminator features, HOG or LBP can be

used. During the implementation of feature extraction in this classifier, a restriction was observed due to the inability to extract such features from symbols. To address this issue, a morphology technique was performed to change the boldness of symbols.

2.4.1 Proposed Method

A- **Segmentation with SLIC**: Simple Linear Iterative Clustering (SLIC). Superpixel is a group of connected pixels with similar colors or gray levels.
Superpixel segmentation divides an image into hundreds of non-overlapping superpixels. SLIC is an approach that utilizes clustering for superpixel segmentation by combining five-dimensional colors and image plane space.
It is a method to replace multistep pre-processing for segmentation. SLIC can solely perform the role of all of the steps stated above to obtain segmented symbols.

B- **Classification**: There are two different architectures described as approaches for mathematical handwritten symbol recognition.

(a) **Data augmentation**: means increasing the size of the training dataset by changing color channels, adding noise, applying random regional sharpening filters, and adding mean-images based on clustering techniques efficiently.
Most recent augmentations are centered on using the Generative Adversarial Network (GAN) model, sometimes swapping the generator network with a genetic algorithm. However, to perform augmentation on a dataset of handwritten mathematical symbols, it must be considered that augmentation does not change the symbol meaning, for example, $\cup$ when is horizontally flipped, it is converted to $\cap$. Therefore, augmentation techniques cannot be run on all symbols.

(b) **classification**: The most prominent challenge for classification of the handwritten mathematical symbols is the real world nature of the training samples.
In some cases, the extracted descriptor from the last layer of the network proves inner class discrimination more than the outer class.

Making the decision to use one of these two approaches fairly depends upon the nature of the training sample.

Chapter 3

Software Requirements Specifications

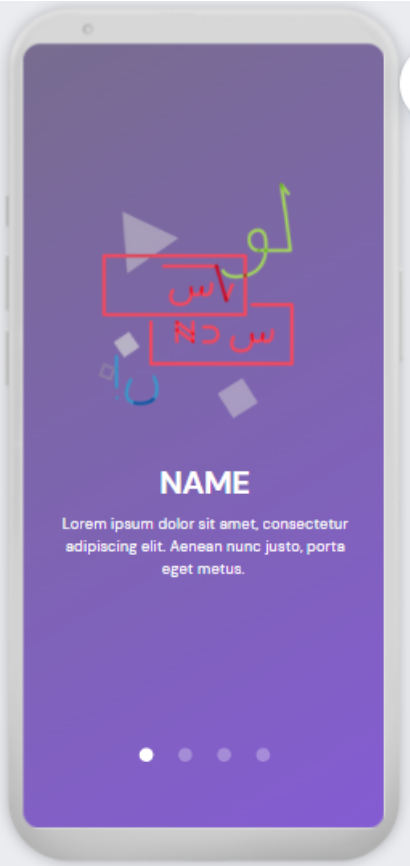
3.1 User Story

Arabic Math Solver Story
Click on the app icon to open the Arabic math solver application, you will see a slider for important notes; please read it carefully. Write the equation you want and then choose which operation you want to apply to it. If you want to erase a specific symbol you can use the eraser. You can undo or redo what you have written. You can move what you have written with "تحريك". If you choose the "ايجاد الحدود" operation then you want to get the roots of your polynomial equation. If you choose the "التكامل" operation then you want to integrate your input. If you choose the "التفاضل" operation, then you want to differentiate your input. Click on send button . Equation and the solution will be sent to the application to be shown to the user.

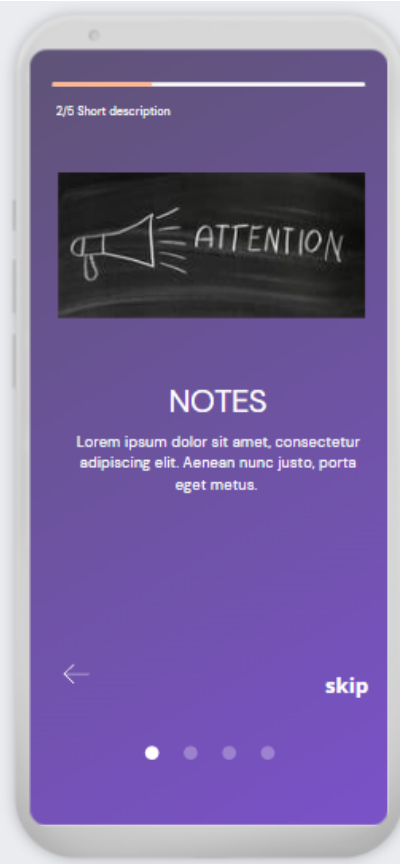
3.2 Requirement Elicitation

3.2.1 Used Techniques

- **Brainstorming technique**: this technique is used to generate new ideas and find a solution for a specific issue. The members included for brainstorming can be domain experts or subject matter experts.
- **Prototyping technique**:

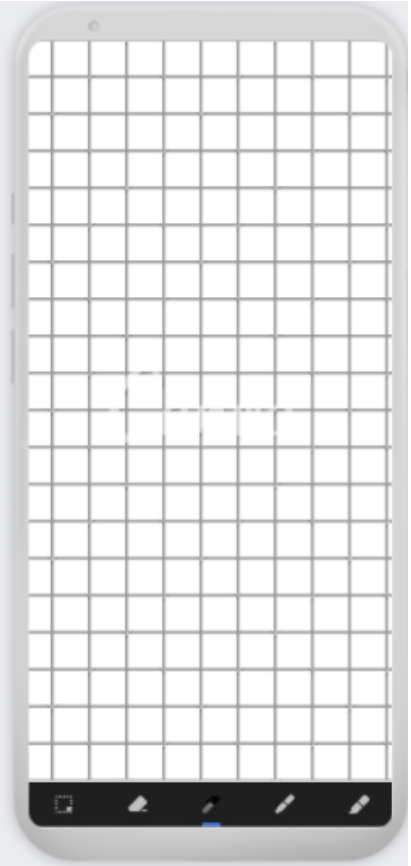


(a) Page 1



(b) Page 2

(c) Page 3



(d) Page 4

Figure 3.1: First pages appear to the user in the app

3.3 Requirement specification

1. Ability to recognize handwritten equations in Arabic notation with good accuracy.
2. The application provides support to solve a wide range of mathematical problems e.g. (Simplifying Solving Polynomials - Integration - Differentiation)
3. The application provides a variety of tools to provide drawing usability to the user e.g. (Pen-Eraser-Move-Zoom).
4. Length of the drawn equation is unlimited.
5. Application Usability.

Chapter 4

System design and modelling

4.1 System Use case diagram

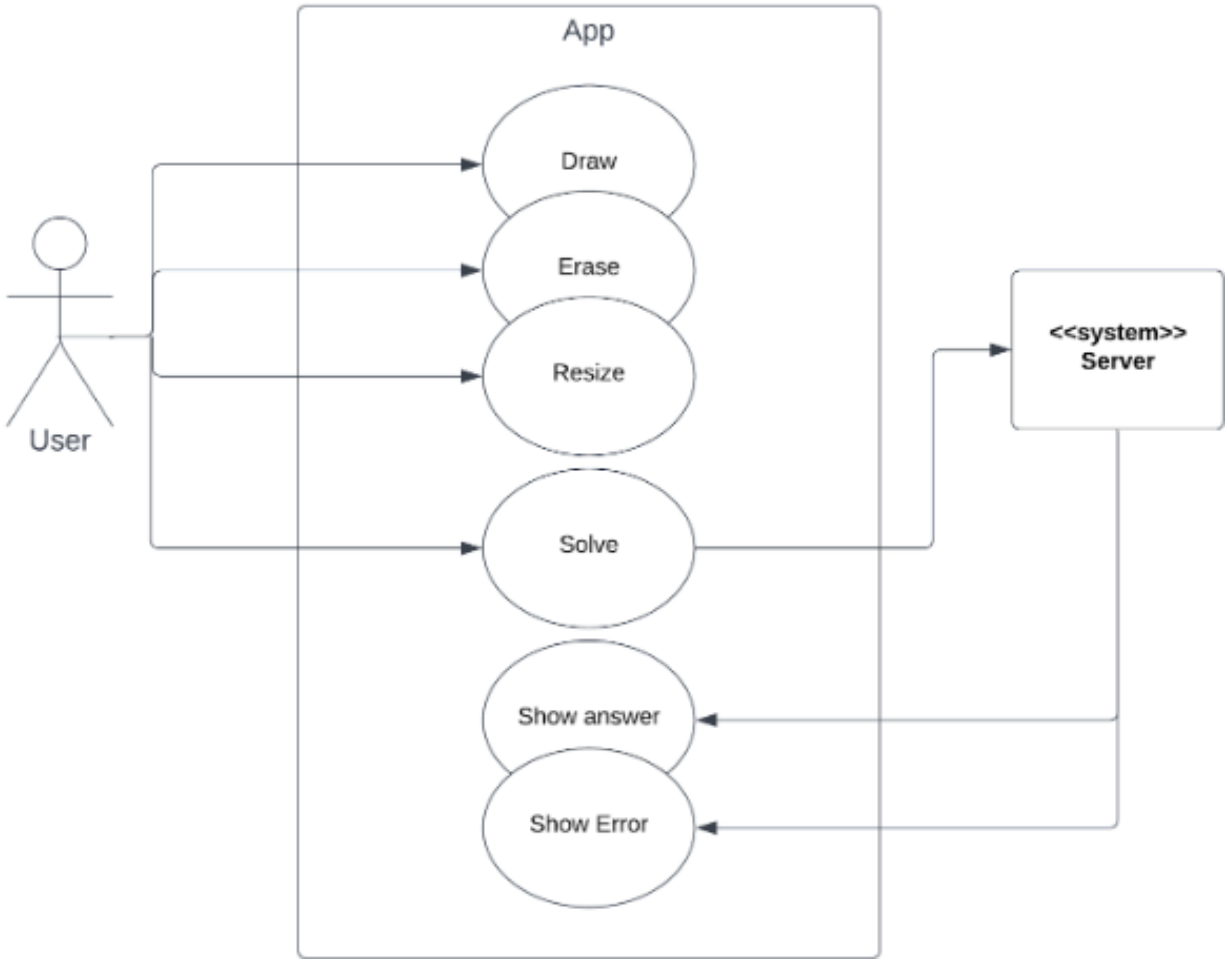


Figure 4.1: Use case diagram

4.2 System Sequence diagram

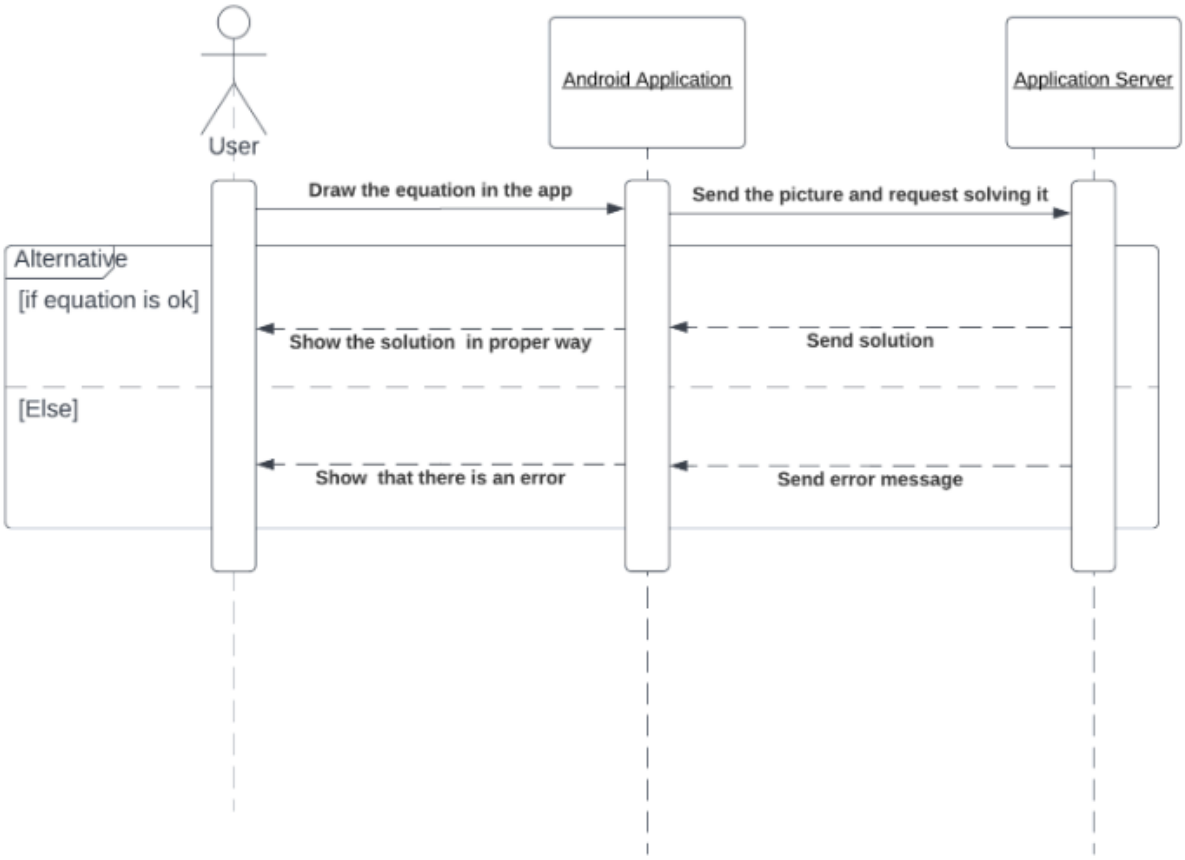


Figure 4.2: System Sequence diagram

4.3 System Activity diagram

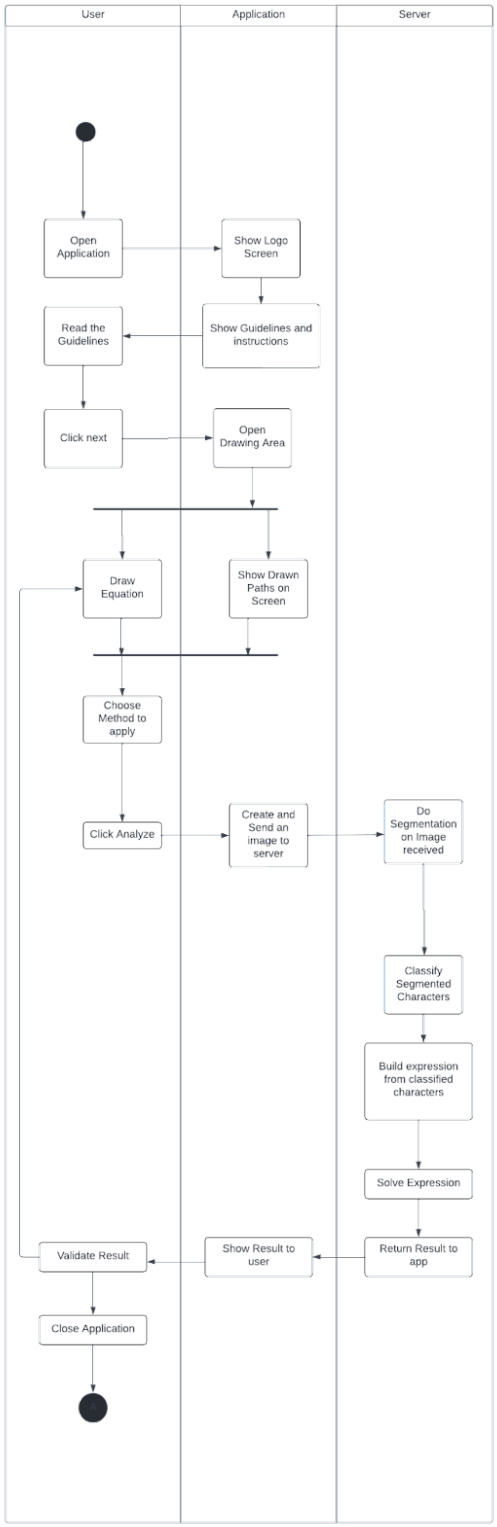


Figure 4.3: System Activity diagram

4.4 Android class diagram

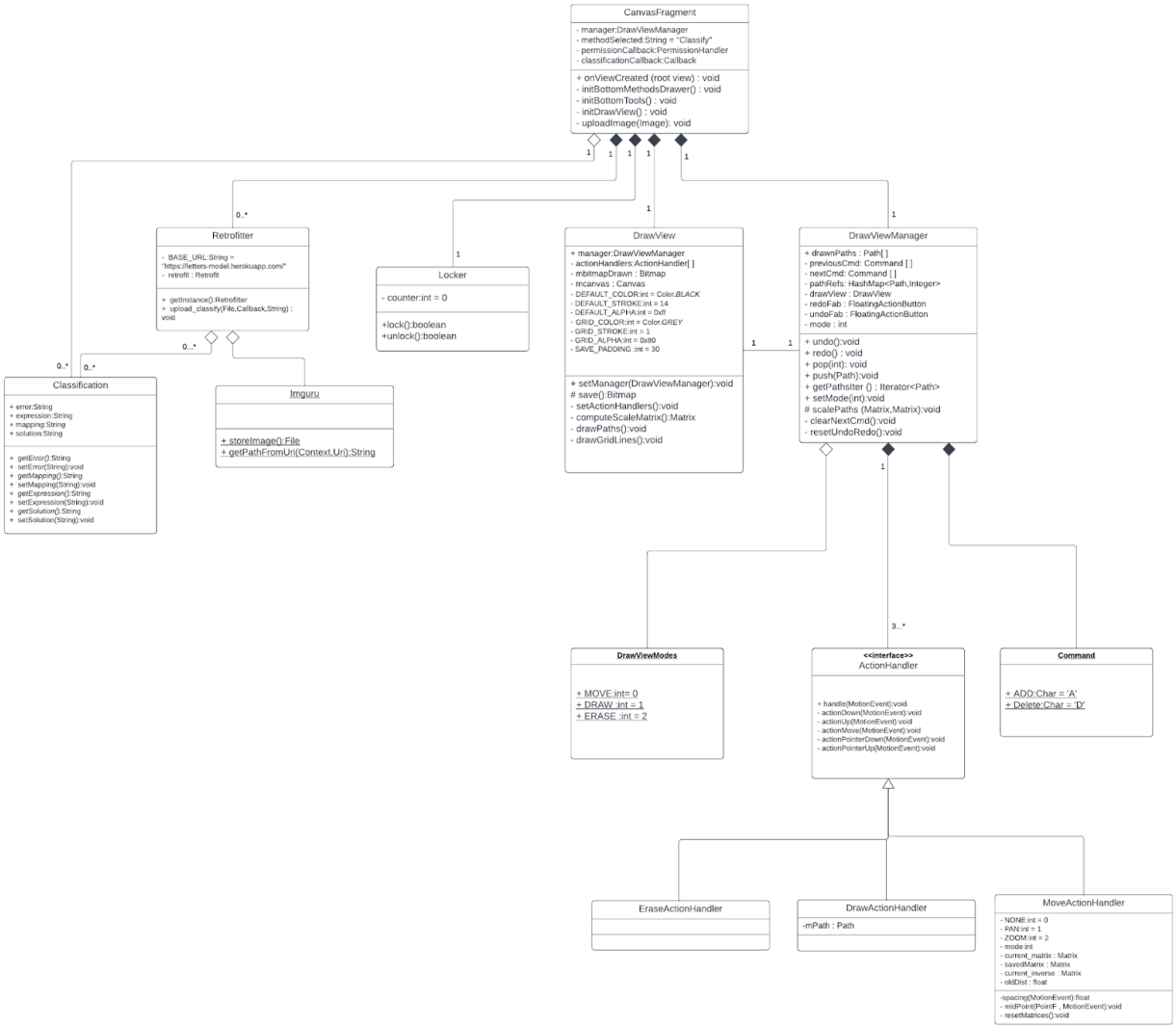


Figure 4.4: Android class diagram

4.5 Server class diagram

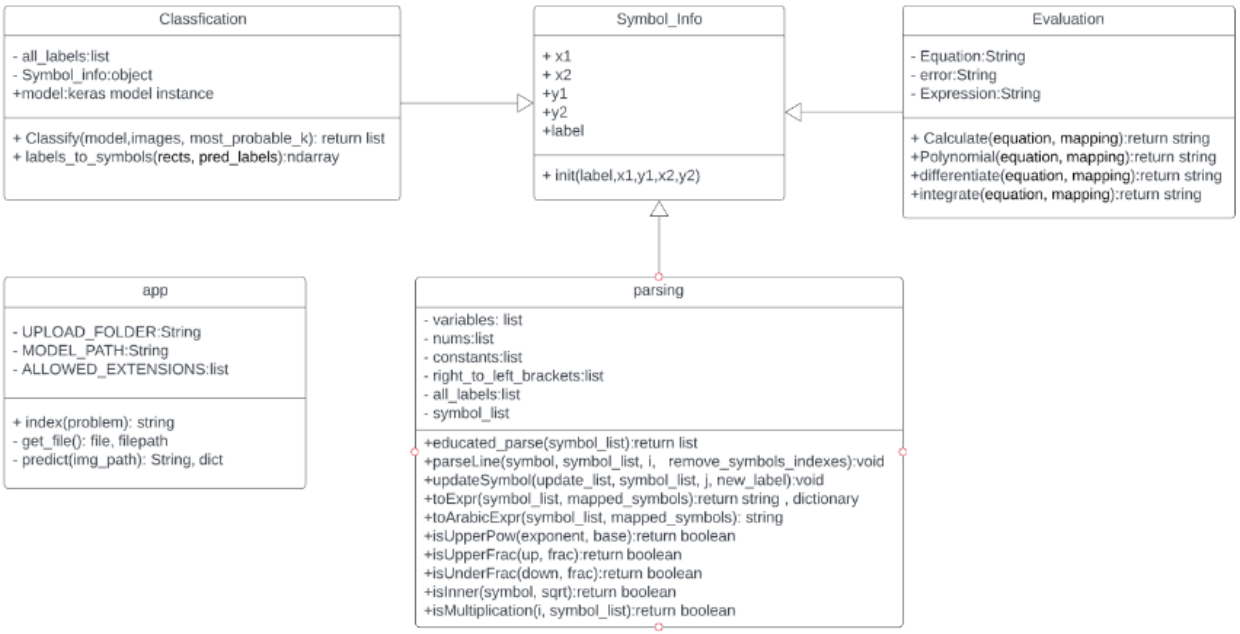


Figure 4.5: Server class diagram

Chapter 5

Architecture

5.1 Client-Server Architecture

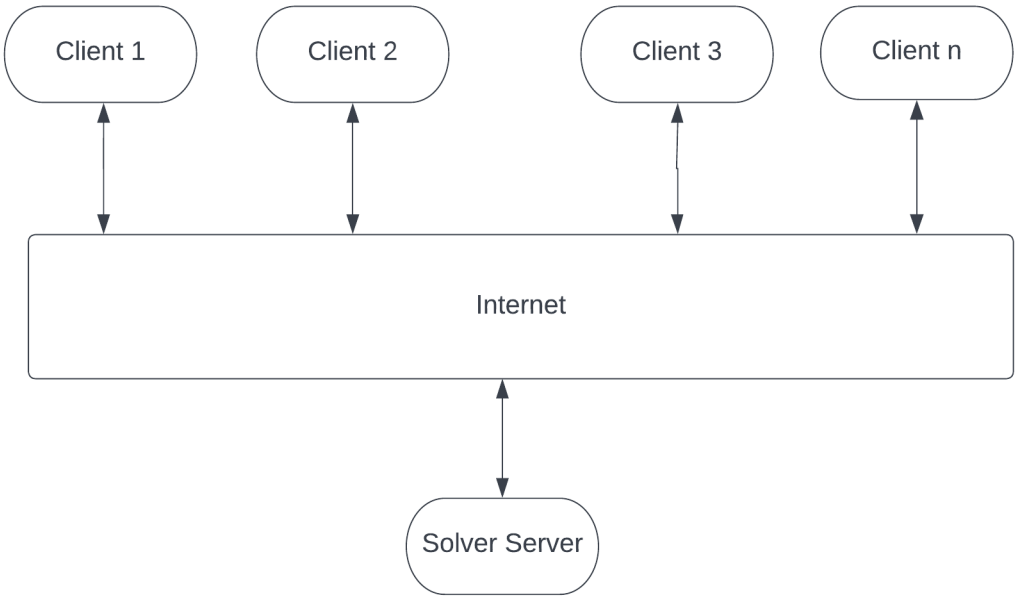


Figure 5.1: Client-server architecture of our app

5.1.1 Description

- In a client–server architecture, the functionality of the system is organized into services. Clients are users of these services and access servers to make use of them.
- The functionality of the solver server → segmentation, classification, parsing, solving, and response with an answer.

5.1.2 Why we chose this architecture

- Servers can be accessed from a range of locations.
- Servers can be replicated.
- Servers can be used when the load on a system is variable.

5.1.3 Advantages

The principal advantage of this model is that servers can be distributed across a network. General functionality could be available to all clients and does not need to be implemented by all services.

5.1.4 Disadvantages

Each service is a single point of failure so susceptible to denial of service attacks or server failure. Performance may be unpredictable because it depends on the network as well as the system.

Chapter 6

General Flow

6.1 Summary

After segmentation, classification, and parsing for example if the input is ٧- = ٣ + س

- The segmentation step extracts each character (٧, -, =, ٣, +, س)
- The classification step recognizes each character and outputs its label
- The parsing step creates the equation in English by the following steps:
    1. Storing all the labels in a map so that the result of the classification (the label) linked with its analogous in English letters
    2. Creating 5 lists for variables, numbers, constant, functions, and brackets
    3. Convert the Arabic equation to English mathematical expression
    4. After converting the Arabic equation to English equation we use **SymPy** library to solve the English mathematical expression based on the type of the problem (simplification, roots of polynomial, differentiation, or integration)

Through this step we loop from right to left (because Arabic in default from right left) and add to the expression from left to right (English expression).
In this step we handle as possible as we can all corner cases:

- Handle power in middle of the expression as illustrated in converting to expression section as logic and here is the implementation
- Add multiplication sign
- Handle fraction
- Handle square root

- Handle log
- Simplification, ex: 1+1 or log(100)
- Polynomial, ex: x+1 = 0
- Differentiation ex: $\frac{\partial}{\partial x}$ (x)
- Integration ex: $\int x dx$

Chapter 7

External Libraries, and Tools

7.1 External Libraries

7.1.1 SymPy[8]

Why use SymPy?

SymPy is a Python library for symbolic mathematics. It aims to become a full-featured computer algebra system (CAS) while keeping the code as simple as possible in order to be comprehensible and easily extensible. SymPy is written entirely in Python.

SymPy is:

- **Free:** Licensed under BSD, SymPy is free both as in speech and as in beer
- **Python-based:** SymPy is written entirely in Python and uses Python for its language.
- **Lightweight:** SymPy only depends on mpmath, a pure Python library for arbitrary floating point arithmetic, making it easy to use.
- **A flexible library:** Beyond use as an interactive tool, SymPy can be embedded in other applications and extended with custom functions.

Here are some projects using SymPy that are similar to our project

- *pyneqsys*: Solve symbolically defined systems of non-linear equations numerically.
- *pyodesys*: Straightforward numerical integration of ODE systems from Python.
- *galgebra*: Geometric algebra (previously sympy.galgebra).

- *SageMath*: Open source SageMath is a computer algebra system with features covering many aspects of mathematics, including algebra, combinatorics, graph theory, numerical analysis, number theory, calculus and statistics, all that is done using SymPy.

7.1.2 OpenCV[9]

Why use OpenCV?

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products.

OpenCV is:

- **Free:** OpenCV is BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.
- **multi-platform & -Language library:** OpenCV has C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS.
- **CPP-based:** OpenCV is written natively in C++ and has a templated interface that works seamlessly with STL containers.
- **Optimized:** OpenCV has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms.

Most commonly used OpenCV methods in our project

- **resize:** This function resizes the image down to or up to the specified size.
- **threshold:** This function applies fixed-level thresholding to a multiple-channel array. The function is typically used to get a bi-level (binary) image out of a grayscale image (compare could be also used for this purpose) or for removing a noise, that is, filtering out pixels with too small or too large values..
- **drawContours:** This function draws & retrieves contour outlines from the binary image using the algorithm[10]. The contours are a useful tool for shape analysis and object detection and recognition.
- **boundingRect:** This function of OpenCV is used to draw an approximate rectangle around the binary image. This function is used mainly to highlight the region of interest after obtaining contours from an image

7.2 Tools

7.2.1 Android Studio IDE

Introduction

Android Studio provides the fastest tools for building apps on every type of Android device.

Features

- **Visual layout editor:** Create complex layouts Then preview the layout on any screen size by selecting one of various device configurations or by simply resizing the preview window.
- **Visual layout editor:** Find opportunities to reduce your Android app size by inspecting the contents of your app APK file, even if it wasn't built with Android Studio. Inspect the manifest file, resources, and DEX files. Compare two APKs to see how your app size changed between app versions.
- **Fast emulator:** Install and run apps faster than with a physical device and simulate different configurations and features, including ARCore, Google's platform for building augmented reality experiences.
- **Intelligent code editor:** Write better code, work faster, and be more productive with an intelligent code editor that provides code completion for Kotlin, Java, and C/C++ languages.
- **Flexible build system:** Powered by Gradle, Android Studio's build system allows you to customize your build to generate multiple build variants for different devices from a single project.
- **Realtime profilers:** The built-in profiling tools provide realtime statistics for your app's CPU, memory, and network activity. Identify performance bottlenecks by recording method traces, inspecting the heap and allocations, and see incoming and outgoing network payloads.

Chapter 8

Symbols Segmentation

The first stage at which the image of an expression goes through is the segmentation stage which:

- preprocesses an image
- find symbols' contours
- for each contour, it crops this part of the image with its bounding rectangle

and then sends these crops and bounding rectangles which corresponds to each symbol in the expression to the next stage which is the symbols recognition.

Most functionalities are supported by **OpenCV** which is a library of programming functions mainly aimed at real-time computer vision.

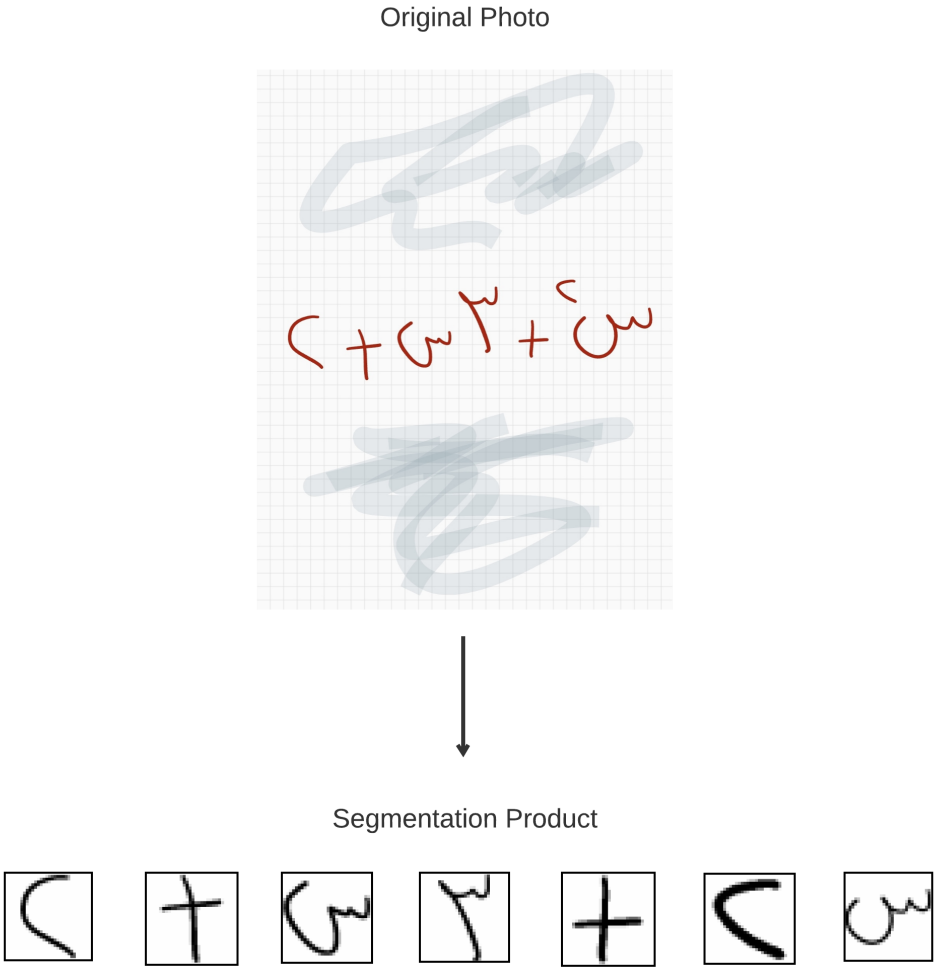


Figure 8.1: Overview of the final result of the segmentation stage

8.1 What are Contours?[11]

Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity. The contours are a useful tool for shape analysis and object detection and recognition.

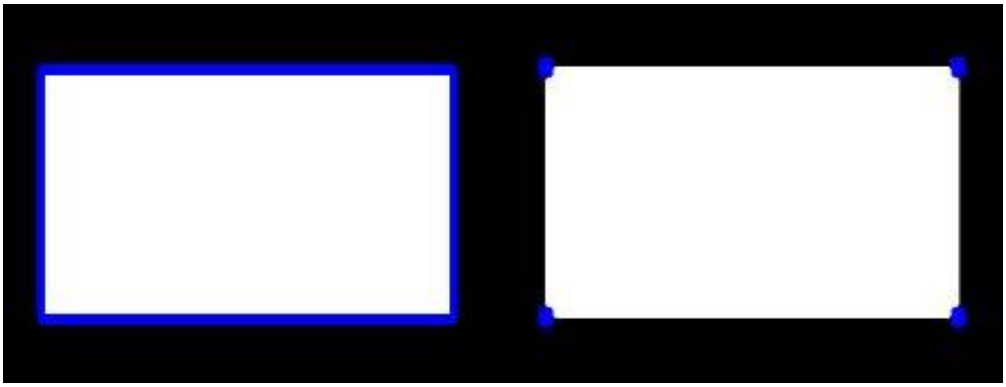


Figure 8.2: Image of contours of a rectangle

8.2 Image Preprocessing

This a crucial step in image segmentation where filtering the image eases the tasks of other stages and increases the overall accuracy of the final expression.

First, the image is converted to gray scale using `cv2.cvtColor()` to simplify it and reduce the overall computation requirements.

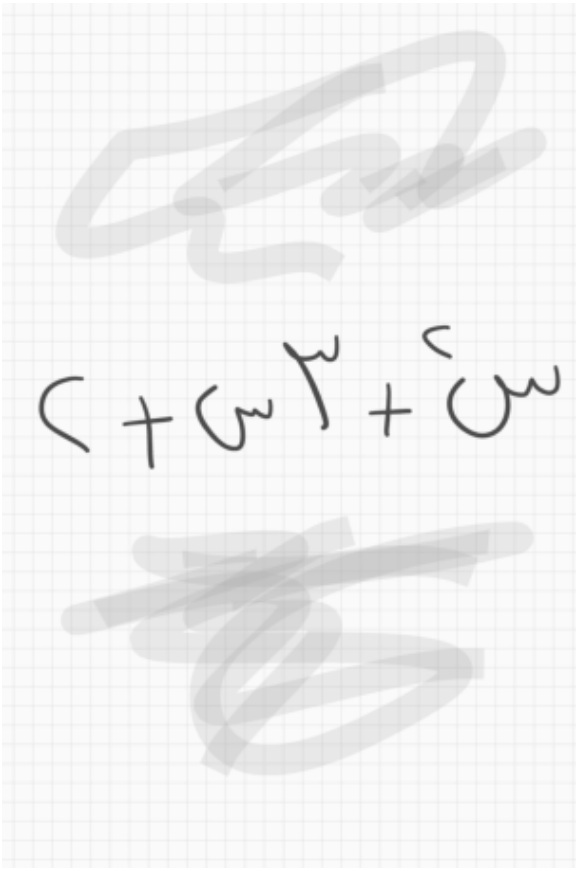


Figure 8.3: Converting to grayscale

Secondly, thresholding is applied to the image using `cv2.threshold()` where each pixel if it is smaller than the threshold, it is set to 0, otherwise it is set to a maximum value. This step clears the image from any non-related page lines or dust or blur and makes the important parts of the image much more clear.

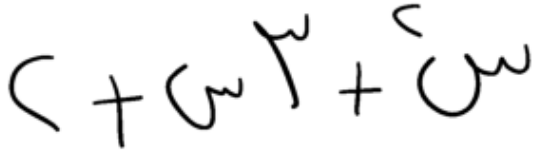


Figure 8.4: Apply thresholding

Thirdly, the image's white and black pixels are flipped because finding the contours requires the target object to be white on a black background.

Figure 8.5: Flipping black and white pixels

8.3 Contours Finding

In this step, the image is passed to OpenCV's `findContours()` where it finds the contours of each symbol and their hierarchy (whether a contour is part or inside another contour).

These contours are then sorted in descending order of **x2**, which is the most right point in the **bounding rectangle** around the contour.

The contours could be drawn using `drawContours()`.

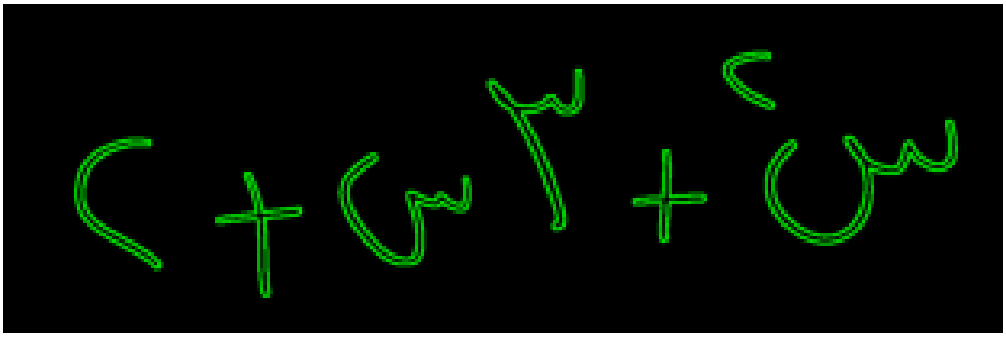


Figure 8.6: Example of drawn contours on the image

8.4 Masking the Contours

Each symbol is required on its own to be passed to the recognition stage, so the contours are looped through where each contour is

1. masked by 0 value on a matrix filled with 1s using `drawContours()` with thick line to cover all the symbol
2. then this mask is applied on a white blank image where these masked pixels are converted to black to represent the symbol
3. while the symbol is the only thing in the image, it's cropped out of the image
4. the cropped image is padded to be square image then resized to (32x32) pixels

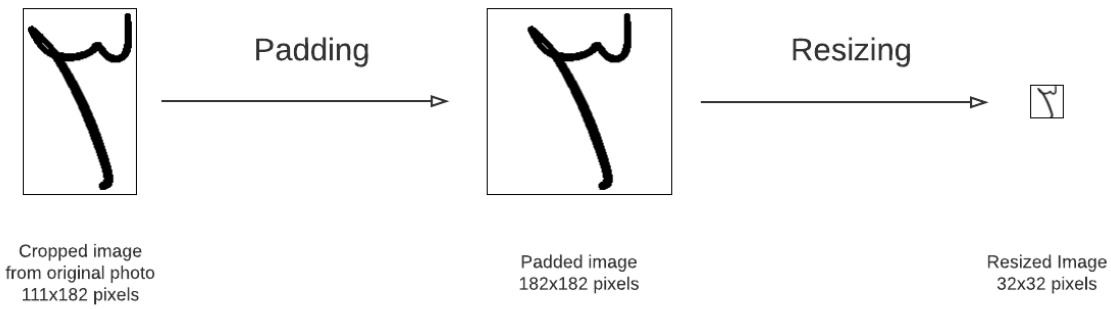


Figure 8.7: Example of cropped image, padded, then resized

These small images with their 4 coordinates **(x1, y1, x2, y2)** sorted in a descending order according to x2 are passed to the recognition stage.

Chapter 9

Symbols Recognition

The aim of symbols recognition stage is to recognize Arabic letters and symbols from untitled segmented images into annotated string characters that can be fitted into a string equation to be solved later.

This was done mainly using a trained convolutional neural network and we have used *Math-Net* as our base so the model takes as input segmented gray-scaled images of dimensions (32x32x1), and outputs one of 43 different labels to each image using a softmax layer, where each label represents either an Arabic letter, mathematical symbol or a brackets symbol.

For symbols recognition we tried mainly 3 models, from which we chose one to continue with, our choice was based on whether the model is overfitting or underfitting the data, following is the architecture of the different model and their training/test accuracy:

9.1 Models

9.1.1 First Model

Layer (type)	Output Shape	Param #
(BatchNormalization)	(None, 32, 32, 1)	4
(Conv2D)	(None, 32, 32, 64)	1664
(Conv2D)	(None, 32, 32, 64)	102464
(MaxPooling2D)	(None, 16, 16, 64)	0
(Dropout)	(None, 16, 16, 64)	0

```
(Conv2D)                (None, 16, 16, 128)     73856

(Conv2D)                (None, 16, 16, 128)     147584

(MaxPooling2D)          (None, 8, 8, 128)       0

(Dropout)               (None, 8, 8, 128)       0

(Flatten)               (None, 8192)            0

(Dropout)               (None, 8192)            0

(Dense)                 (None, 256)             2097408

(Dense)                 (None, 43)              11051

=================================================================
Total params: 2,434,031
Trainable params: 2,434,029
Non-trainable params: 2

training evaluation
loss: 0.0218 - accuracy: 0.9934 - recall: 0.9933 - precision: 0.9937 -
f1_score: 0.9935
test evaluation
loss: 0.0526 - accuracy: 0.9875 - recall: 0.9873 - precision: 0.9878 -
f1_score: 0.9875
```

9.1.2 Second Model

```
Layer (type)                Output Shape        Param #
=================================================================
(BatchNormalization)        (None, 32, 32, 1)   4

(Conv2D)                    (None, 32, 32, 64)  1664

(MaxPooling 2D)             (None, 16, 16, 64)  0

(Dropout)                   (None, 16, 16, 64)  0

(Conv2D)                    (None, 16, 16, 128) 73856

(MaxPooling2D)              (None, 8, 8, 128)   0

(Dropout)                   (None, 8, 8, 128)   0
```

```
(Flatten)                    (None, 8192)        0

(Dropout)                    (None, 8192)        0

(Dense)                      (None, 256)         2097408

(Dense)                      (None, 43)          11051

=================================================================
Total params: 2,183,983
Trainable params: 2,183,981
Non-trainable params: 2
training evaluation
loss: 0.0181 - accuracy: 0.9948 - recall_2: 0.9943 - precision_2: 0.9953 -
f1_score: 0.9948
test evaluation
loss: 0.0428 - accuracy: 0.9880 - recall_2: 0.9873 - precision_2: 0.9898 -
f1_score: 0.9886
```

9.1.3 Third Model

```
Layer (type)                 Output Shape              Param #
=================================================================
(BatchNormalization)         (None, 32, 32, 1)           4

(Conv2D)                     (None, 32, 32, 64)          1664

(Conv2D)                     (None, 32, 32, 64)          102464

(MaxPooling 2D)              (None, 16, 16, 64)          0

(Dropout)                    (None, 16, 16, 64)          0

(Conv2D)                     (None, 16, 16, 128)         73856

(Conv2D)                     (None, 16, 16, 128)         147584

(MaxPooling                  (None, 8, 8, 128)           0 2D)

(Dropout)                    (None, 8, 8, 128)           0

(Flatten)                    (None, 8192)                0

(Dropout)                    (None, 8192)                0
```

```
(Dense)                      (None, 43)                352299

=================================================================
Total params: 677,871
Trainable params: 677,869
Non-trainable params: 2

training evaluation
loss: 0.0354 - accuracy: 0.9901 - recall_6: 0.9887 - precision_6: 0.9915 -
f1_score: 0.9901
test evaluation
loss: 0.0579 - accuracy: 0.9830 - recall_6: 0.9818 - precision_6: 0.9855 -
f1_score: 0.9836
```

9.1.4 Forth Model

```
Layer (type)                 Output Shape              Param #
=================================================================
(BatchNormalization)         (None, 32, 32, 1)         4

(Conv2D)                     (None, 32, 32, 64)        1664

(MaxPooling2D)               (None, 16, 16, 64)        0

(Dropout)                    (None, 16, 16, 64)        0

(Conv2D)                     (None, 16, 16, 128)       73856

(MaxPooling2D)               (None, 8, 8, 128)         0

(Dropout)                    (None, 8, 8, 128)         0

(Flatten)                    (None, 8192)              0

(Dropout)                    (None, 8192)              0

(Dense)                      (None, 43)                352299

=================================================================
Total params: 427,823
Trainable params: 427,821
Non-trainable params: 2

training evaluation
```

```
loss: 0.0569 - accuracy: 0.9860 - recall_4: 0.9829 - precision_4: 0.9890 -
f1_score: 0.9860
test evaluation
loss: 0.0732 - accuracy: 0.9785 - recall_4: 0.9750 - precision_4: 0.9839 -
f1_score: 0.9794
```

9.2 Conclusion

The models have high values for accuracy and F score and this refers to that the models have learned the patterns at our dataset well , but our dataset didn't cover much patterns so we depend on the model with the better results from test cases (Junit tests) of some written equations with the same writing patterns of our dataset to check which is the better model.

9.3 Results

The first model is the one which passed the tests so we chose it.

In chapter 13, we made an evaluation for this model with images of symbols and equations from other people that didn't contribute to our dataset to check how our model can recognize their writing.

Chapter 10

Symbols Parsing

After classifying the symbols, they are passed to the parsing component with their coordinates **(x1, y1, x2, y2)**, where some symbols prediction is updated first which is what we call **educated parsing**, then the symbols are collected together to form an expression according to the current part of the equation, for ex: power, or fraction, or square root, etc.

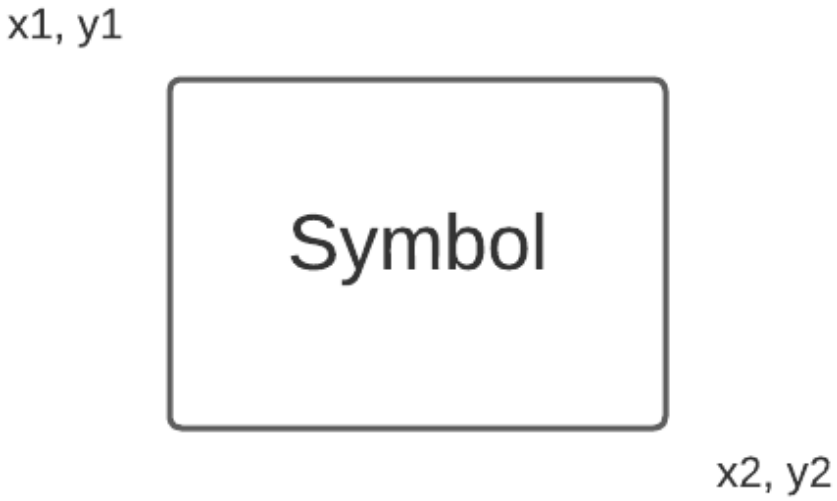


Figure 10.1: A general symbol with its coordinates

10.1 Order Assumption

After segmentation we will have a list of symbols that are ordered by the most right point ***x2***. This order was found to be natural and useful in many specific situations like fraction, and logarithm.

10.2 Educated Parsing

The prediction of a symbol doesn't take into consideration that this symbol might be part of a letter or a sign. For example, a line might be part of an equal sign, or division, or if it's too long it might be a fraction, or it's just a minus sign.

The educated parsing targets this problem by looping the symbols and updates each symbol according to its relative symbols.

If there is a dash or line, then it could be one of four symbols: equal sign, division sign, fraction line, or a minus sign. This is handled by **parseLine** function.

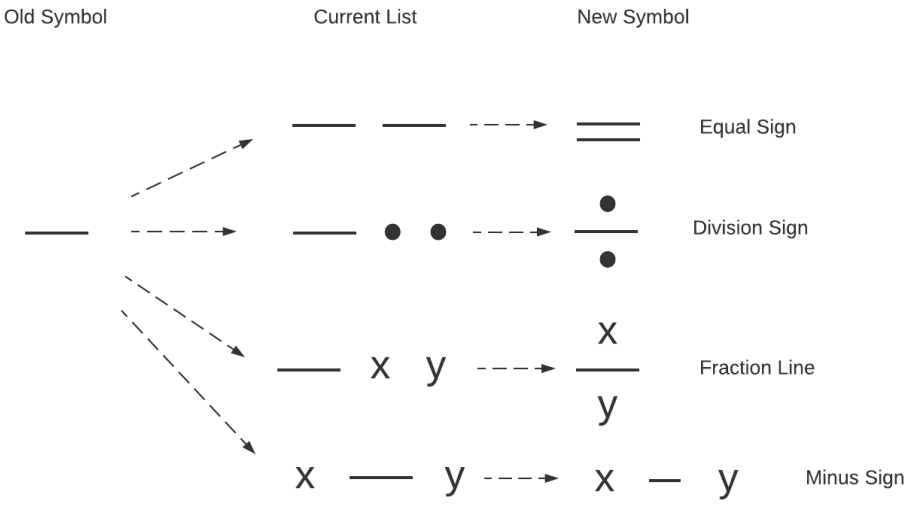


Figure 10.2: Simple dash different outputs depending on the current list

There was supposed to be a function **parseDot** where it handles dot where it could be zero, or part of a letter like in shen (ش) or baa (ب) but it would complicate the problem so **we relaxed it by the following assumption.**

10.3 Arabic Letters Assumptions

We have noticed from article in Wikipedia: Modern Arabic mathematical notation, from the example section that shape of letters that used in Arabic mathematics are:

Latin	Arabic		Notes
a	أ	ا	From the Arabic letter ا *ʾalif*; *a* and ا *ʾalif* are the first letters of the Latin alphabet and the Arabic alphabet's ʾabjadī sequence respectively
b	ب	ب	A dotless ب *bāʾ*; *b* and ب *bāʾ* are the second letters of the Latin alphabet and the ʾabjadī sequence respectively
c	حـ	ـحـ	From the initial form of ح *ḥāʾ*, or that of a dotless ج *jīm*; *c* and ج *jīm* are the third letters of the Latin alphabet and the ʾabjadī sequence respectively
d	ئ	د	From the Arabic letter د *dāl*; *d* and د *dāl* are the fourth letters of the Latin alphabet and the ʾabjadī sequence respectively
x	س	س	From the Arabic letter س *sīn*. It is contested that the usage of Latin *x* in maths is derived from the first letter ش *šīn* (without its dots) of the Arabic word شيء *šayʾ (un)* [ʃajʔ(un)], meaning *thing*.[1] (*X* was used in old Spanish for the sound /ʃ/). However, according to others there is no historical evidence for this.[2][3]
y	ص	ص	From the Arabic letter ص *ṣād*
z	ع	ع	From the Arabic letter ع *ʿayn*

Figure 10.3: Shapes of Arabic letters for variables' names

so two of these letters are without dots: ب → ب, جـ → حـ

For trigonometric functions here is the table:

Description	Latin	Arabic		Notes
Sine	sin	جا	حا	from حاء *ḥāʾ* (i.e. dotless ج *jīm*)-ʾ*alif*; also **جب** (جب *jīm-bāʾ*) is used in some regions (e.g. Syria); Arabic for "sine" is جيب *jayb*
Cosine	cos	حتا	حتا	from حتا *ḥāʾ* (i.e. dotless ج *jīm*)-*tāʾ*-ʾ*alif*; also **تجب** (تَجب *tāʾ*-*jīm-bāʾ*) is used in some regions (e.g. Syria); Arabic for "cosine" is جيب تمام
Tangent	tan	طا	طا	from طا *ṭāʾ* (i.e. dotless ظ *ẓāʾ*)-ʾ*alif*; also **ظل** (ظل *ẓāʾ*-*lām*) is used in some regions (e.g. Syria); Arabic for "tangent" is ظل *ẓill*
Cotangent	cot	طتا	طتا	from طتا *ṭāʾ* (i.e. dotless ظ *ẓāʾ*)-*tāʾ*-ʾ*alif*; also **تظل** (تظل *tāʾ*-*ẓāʾ*-*lām*) is used in some regions (e.g. Syria); Arabic for "cotangent" is ظل تمام
Secant	sec	قا	قا	from قا dotless ق *qāf*-ʾ*alif*; Arabic for "secant" is قاطع
Cosecant	csc	قتا	قتا	from قتا dotless ق *qāf-tāʾ*-ʾ*alif*; Arabic for "cosecant" is قاطع تمام

Figure 10.4: Arabic functions' names

There are trigonometric functions without dots and others with dots so we assumed that all don't have dots.

10.4 Conversion to Expression

After educated parsing the updated symbol list is passed to **to_expr** where the symbols are collected as an expression.

10.4.1 Fraction

When a fraction line is found, the subsequent symbols are split into upper symbols (numerator) and lower symbols (denominator) according to their bounding boxes' spatial relation to that of the fraction line.
Assumption: the fraction line always comes first in order before its numerator and denominator due to the sorting by *x2*, so it's the first point from the right.

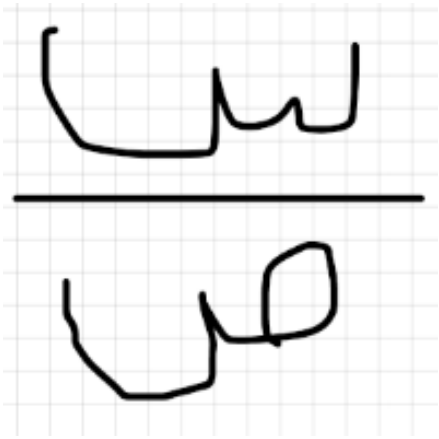
The symbols are then checked to be put in the upper (numerator) list or lower (denominator) list.

- The symbol is considered above the fraction line if the symbol center is between the

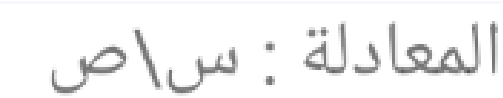
fraction line's ends and the symbol's lowest point is above the fraction line's lowest point.

- The symbol is considered below the fraction line if the symbol center is between the fraction line's ends and the symbol's highest point is below the fraction line's highest point.

Assumption: for the above two conditions to hold, the fraction line must be mostly horizontal, and not inclined or near diagonal.



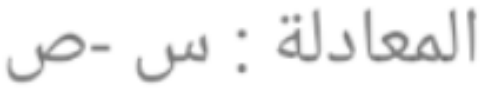
(a) Example



(b) Expression

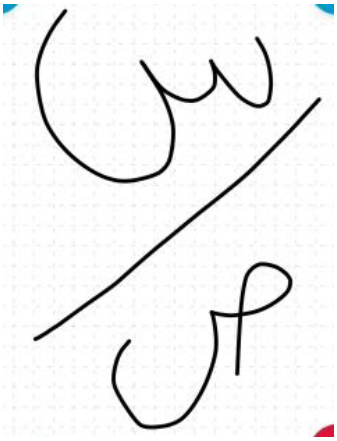
Figure 10.5: Fraction line is the first point to the right

(a) Example



(b) Expression

Figure 10.6: Violated the assumption, the numerator comes first from the right



(a) Example

(b) Expression

Figure 10.7: Violated the assumption, the fraction line is inclined

10.4.2 Power

Every two subsequent symbols are checked from a spatial perspective if they are a pair of base and exponent, this is done by checking the following:

- The exponent's **x center** should be to the left of **0.6** of the base rectangle width
- The exponent's lowest point **y2** should be above **0.4** of the base rectangle height
- The exponent's highest point **y1** should be above the base's highest point **y1**

We chose these two ratios **0.6** and **0.4** based on the different handwritings we examined other than ours, and to make sure that the exponent is captured specifically for letters that are inherently lower to the left half like **sen (س)**.

If this check passed, then the base's label is checked to be one of the following:

- A variable
- A number
- A constant
- A left bracket **(**

and for the exponent, it must be one of the following:

- A variable
- A number
- A constant

- A mathematical function (sin, cos, log, etc..)
- A right bracket) or a negative sign -

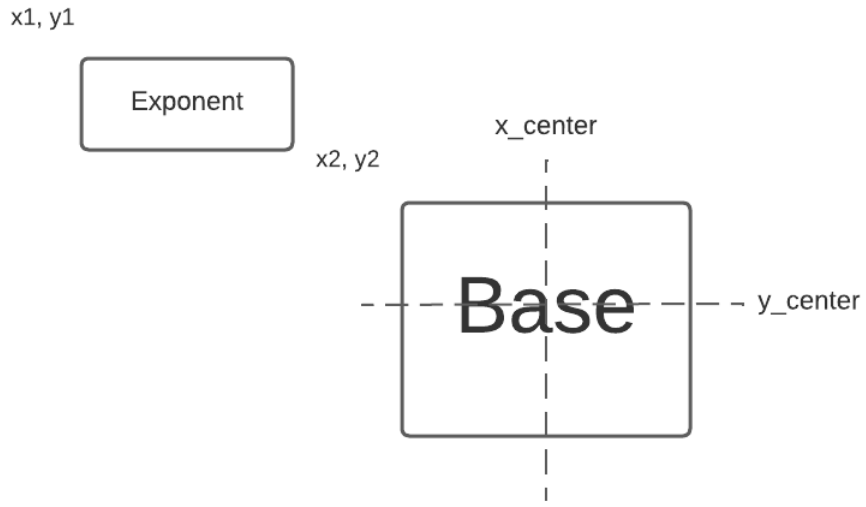
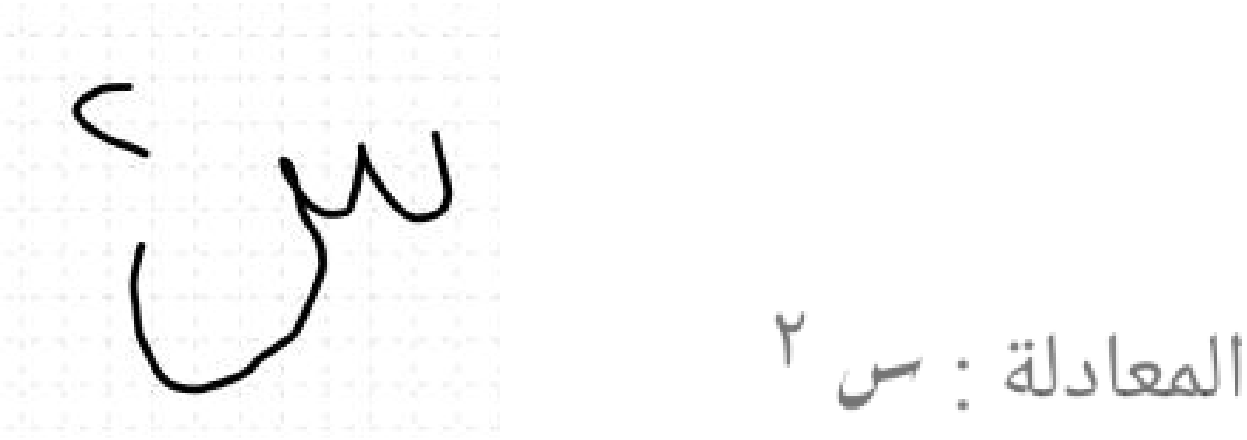


Figure 10.8: Power symbol coordinates



(a) Example **(b)** Expression

Figure 10.9: Power example

10.4.3 Square Root

- The symbol **x center** point must be within the square root's width.
- The symbol's highest point **y1** must lie between the square root's **y1** and **y2**.

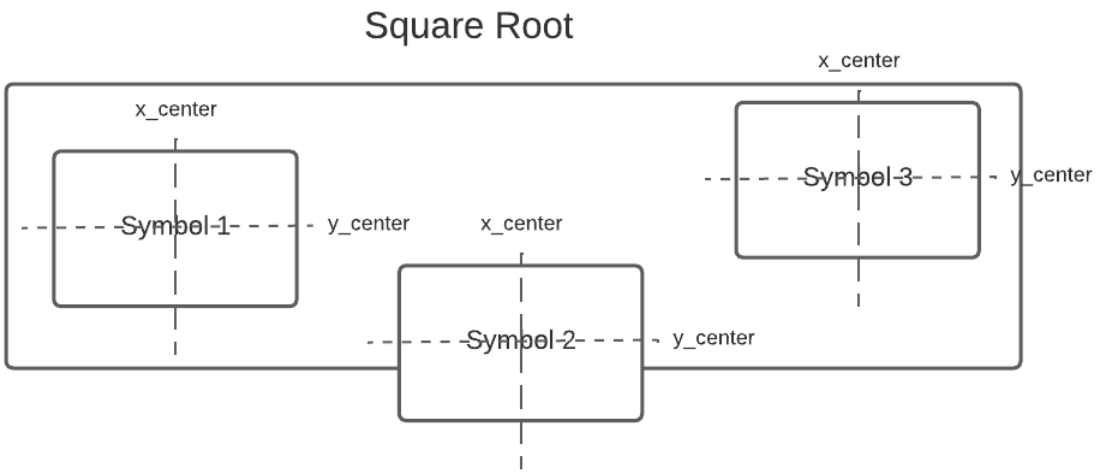


Figure 10.10: Square root symbol

This example would fail because **sad's x_center** is beyond the square root limits:

(a) Example **(b)** Expression

Figure 10.11: Square root violation

10.4.4 Logarithm

The log base and argument both depend on the relative ordering.

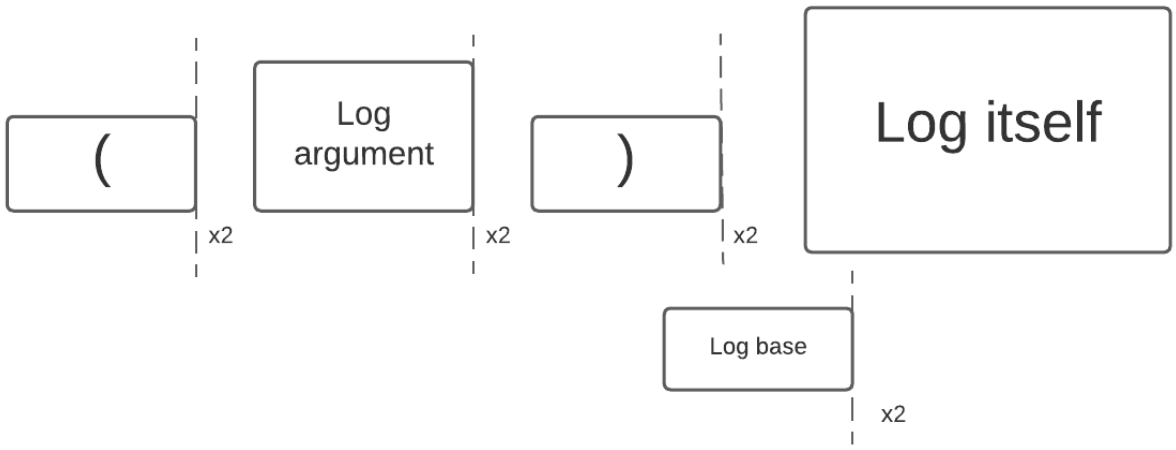


Figure 10.12: Logarithm symbol coordinates

10.4.5 Comment on Assumptions

There are assumptions due to implementation issues that we faced and they are because of:

1. Difficulty of handling things like segmentation and that due to Arabic letters that have dots.
2. Counting on ordering of symbols to use it in parsing .
3. Parsing to reach final mathematical expression.

Chapter 11

Dataset

As mentioned in previous sections, finding a dataset, to train a model on, was a very challenging step that faced us at the beginning of our project. We needed to start considering and building a model to start with, but we didn't find neither open-source database nor previous work that contain data for this particular problem.

We decided to start searching in the most popular communities and sites, such as Kaggle, Research Gate, Springer etc..., which might contain different parts of datasets that we might add up to create a new dataset that we can use a database to start creating a model with.

11.1 Data Sources

Thus, we created our database of data from the following sources:

- Arabic Handwritten Characters Dataset[12].
- Database of handwritten Arabic mathematical formulas images[6].
- Window-Based Descriptors for Arabic Handwritten Alphabet Recognition: A Comparative Study on a Novel Dataset[13].
- Our colleagues & our team members have written large amount of the data.

11.2 Data issues and cleaning

- **Issue #1:** After the first time we trained the model with the data we got from Kaggle, we discovered some issues regarding the model's accuracy. After some debugging, we found a fatal problem within the dataset that there were many duplicates of the same images having different labels within each class. This resulted in having overlapping training and test sets (training and test data containing the same images but with

different labels), and an over-fitting model which provides fake high training and test accuracy but misclassifies any new data provided during the runtime of our application.

Solution #1: We coded a method to iterate in every folder in the dataset, check if there are two or more images that have the same pixel-map and if any were found, remove this duplicate image. That reduced the size of our data set significantly by a ratio of 2/3 of its original size.

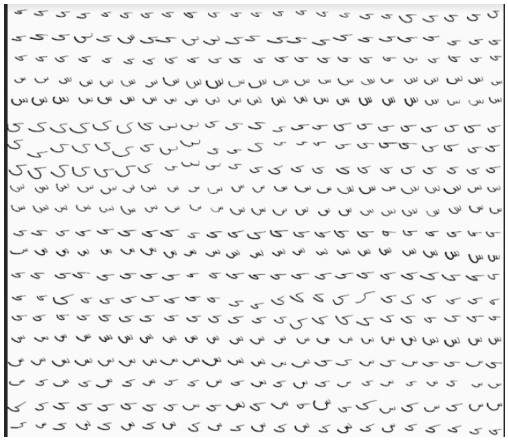
- **Issue #2:** We emailed some Tunisian researchers to ask for their dataset on which they used in their paper[6], and they have responded very late. In addition, we figured out that their dataset was small to some extent and their data was unlabelled and unsorted into classes.

- **Issue #3:** As a result of the first 2 issues, even by adding up data from different sources to create a dataset, we were still in need of more data to support our model's training.

  Solution #3: We started asking our colleagues to write data for us. As their response was timid, we were forced to spend time writing data that we need ourselves.

  Writing the data by ourselves was very challenging because it was tiresome to write every single character alone, so we came up with an idea to write as many similar characters as possible on a single white page, figure 6.1, then execute a segmentation step and extract each character as a solo labeled image.

(a) Example 1



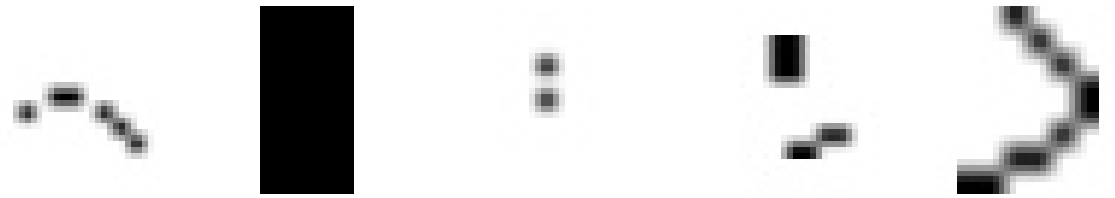
(b) Example 2

Figure 11.1: Writing similar characters in a white page

- **Issue #4:** Sometimes segmentation step (mentioned in Issue #3) would produce undefined images that results from separating the same character into more than one piece or reading noise in the white page as probable characters to be segmented, as in figure 6.3.

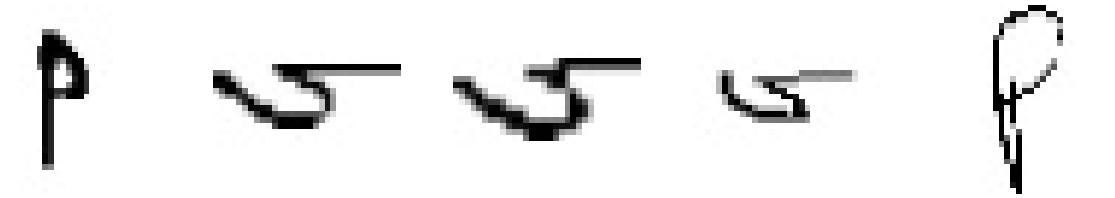
  Solution #4: Every time, We must do a manual cleaning process on these segmented characters to ensure that they were extracted and looked at correctly for the classi-

fication process, as in figure 6.2, to avoid having an ill-trained model on some noisy characters.



(a) Example 1 **(b)** Example 2 **(c)** Example 3 **(d)** Example 4 **(e)** Example 5

Figure 11.2: incorrectly segmented 32X32 characters



(a) Example 1 **(b)** Example 2 **(c)** Example 3 **(d)** Example 4 **(e)** Example 5

Figure 11.3: correctly segmented 32X32 characters

11.3 Dataset statistics

At the end, we reached a total of **110,292 different images** in our dataset, separated into **53 different classes**.

Here's a statistic on how many images in each class :

Class	#Images	Class	#Images	Class	#Images
Alif - الالف	2,482	Sad - الصاد	4,491	Mim - الميم	3,037
Baa - الباء	4,569	Taa - الطاء	2,877	Noon - النون	3,327
Jeem - الجيم	4,862	Ayn - العين	3,474	Haa - الهاء	2,385
Dal - الدال	7,040	Qaf - القاف	2,278	Waw - الواو	2,732
Sen - السين	4,569	Lam - اللام	2,843	Yaa - الياء	1,566
Cosec - قتا	4,569	Int - التكامل	2,877	Cos - جتا	2,837
Log - لو	497	Sin - جا	3,726	Tan - ظا	3,336
Cot - ظتا	2,326	Sec - قا	2,897	Sqrt - الجذر	959
Decimal Dot ,	1,240	Subtract −	2,313	R.Bracket (	2,896
L. Bracket)	2,924	Add +	2,070	F.Slash /	157
Phi ϕ	83	Theta θ	543	Multiply *	1,711
Zero - صفر	1,516	One - واحد	1,516	Two - اثنين	3,407
Three - ثلاثة	3,006	Four - أربعة	2,835	Five - خمسة	1,714
Six - ستة	1,779	Seven - سبعة	1,823	Eight - ثمانية	2,046
Nine - تسعة	2,206				

11.4 Data Preprocessing

Before providing images to the model to be classfied, we first had to do some Data Preprocessing on them.

Here's our steps for the Data Preprocessing step :

1. Read images and convert them from **RGB** to **Grayscale** using **OpenCV threshold**.
2. Find contours using **Opencv findContours**, and sort them on x2.
3. For each contour, draw it on a white blank image, and collect images in an array.
4. Resize images into 32x32 pixels using **Opencv Resize**, and pass them to the model afterwards.

Chapter 12

Testing

Our project could be split into 3 parts in general:

- character recognition
- segmentation and parsing
- solving problems or evaluation of mathematical expressions

and we tested them by providing test cases or by simply evaluating the accuracy.

12.1 Character Recognition Testing

We tested the model by splitting the dataset into 80/10/10 for train/cross validation/testing and achieving a test accuracy **98.04%** was enough for its testing.

12.2 Segmentation and Parsing Testing

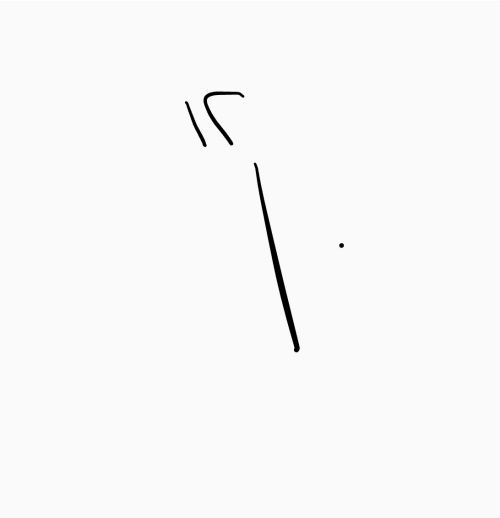
Knowing each symbol position relative to its predecessor is a crucial functionality in our project since it's used for recognizing exponents of powers, what symbols under the square root, and what is the base for logarithm.

So we tested each parsing case as if it's a unit test before moving to testing the results of solving polynomial equations for example.

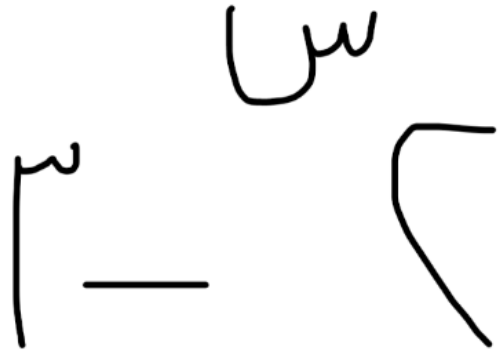
12.2.1 Power Test

In this test, we make sure that the exponent is parsed successfully whether it's one digit or more, a variable, or a long expression. Also making sure that the base is one digit or more,

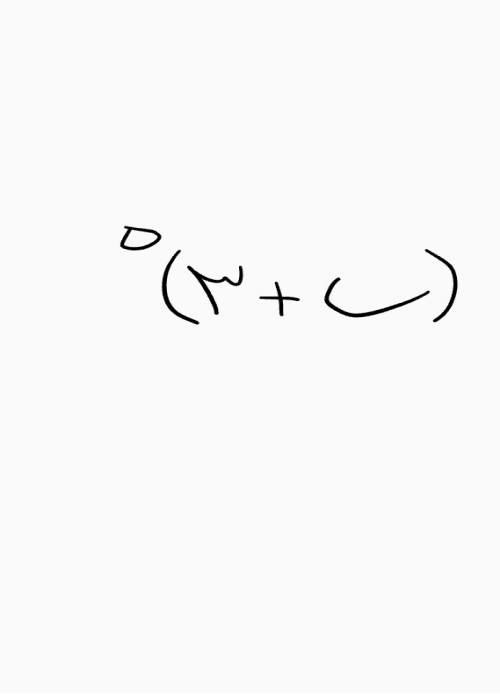
or an expression enclosed by brackets.



(a) Multiple digits exponent



(b) Variable as exponent



(c) Base enclosed by brackets

(d) Long expression as exponent

Figure 12.1: Power different cases

12.2.2 Fraction Test

In this test, we make sure that the fraction is handled successfully, whether it's a simple one digit numerator over one digit denominator, or a long expression with another fraction in the numerator over another long expression in the denominator.

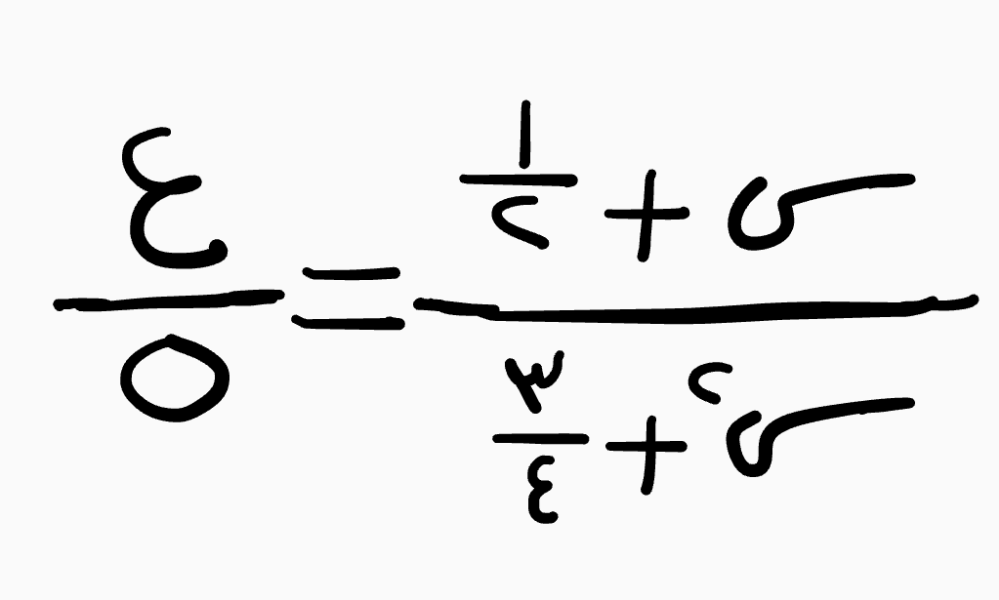


Figure 12.2: Fraction different cases

12.2.3 Square Root Test

In this test, we make sure that the expression under the square root is parsed successfully.

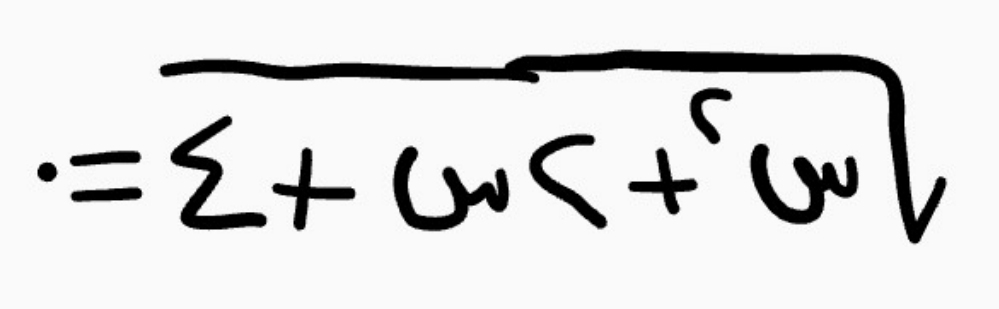


Figure 12.3: Square root test case

12.2.4 Decimal Point Test

In this test, we make sure that the number with decimal point is parsed successfully.

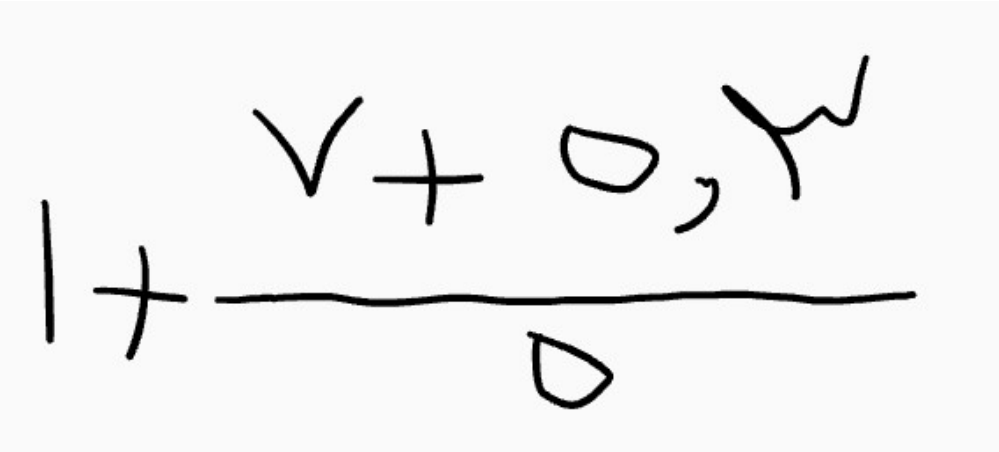
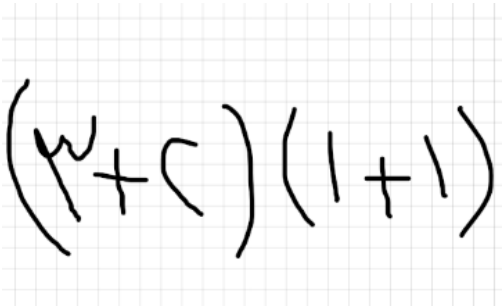


Figure 12.4: Decimal point test case

12.2.5 Multiplication Test

In this test, we make sure that the adjacent expressions whether they are numbers or variables or constants or long expressions in brackets are multiplied together successfully.

(a) Different symbols multiplied together



(b) 2 expressions in brackets multiplied together

Figure 12.5: Multiplication different cases

12.2.6 Trigonometric Functions Test

In this test, we make sure that the different trigonometric functions are parsed successfully, whether with long expressions as arguments or short ones, and whether there is more than on trigonometric function in the same expression or just one.

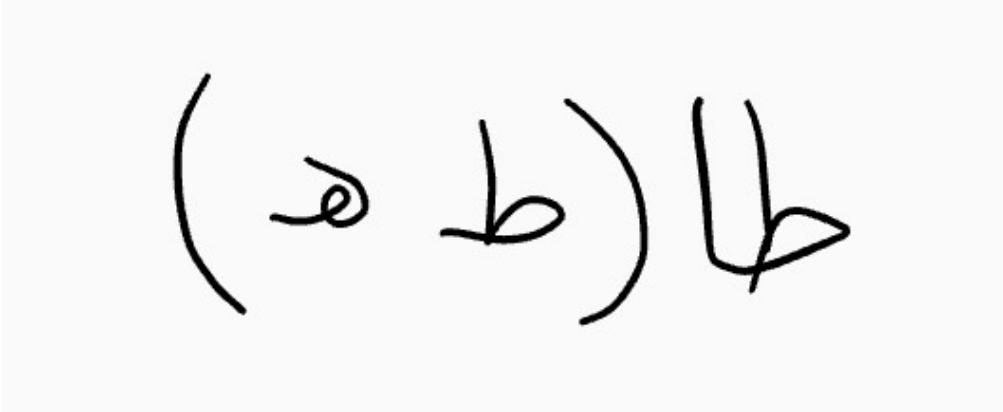
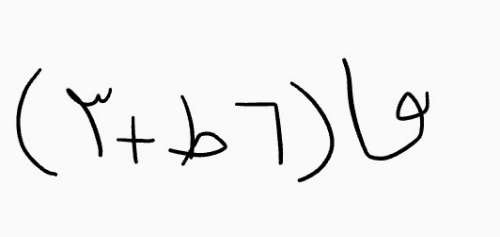
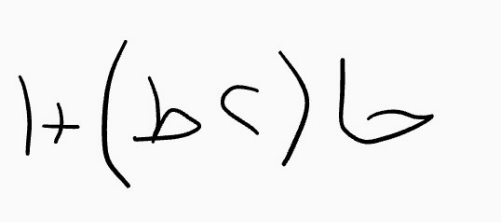


Figure 12.6: Trigonometric function with constants

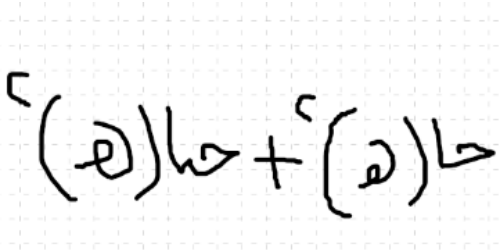


(a) Trigonometric function with long expression

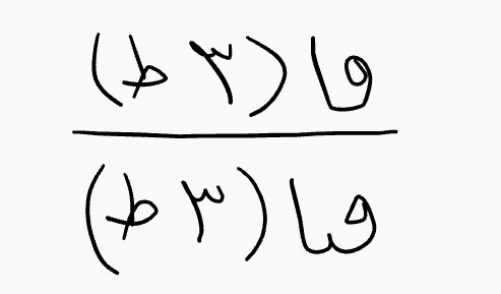


(b) Trigonometric function as part of an expression

Figure 12.7: Trigonometric functions in long expressions



(a) Multiple trigonometric functions with power

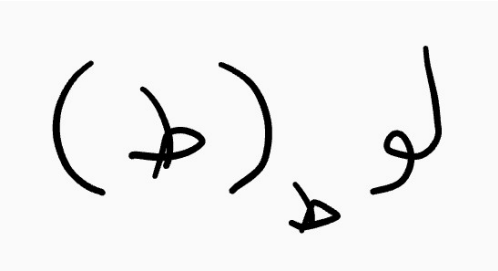


(b) Multiple trigonometric functions as a fraction

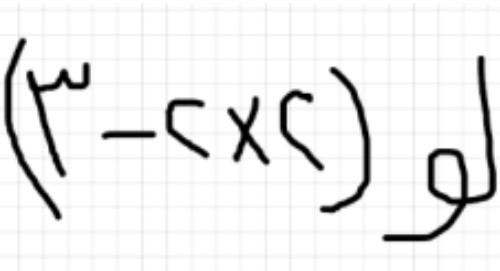
Figure 12.8: Multiple trigonometric functions in different cases

12.2.7 Logarithm Test

In this test, we make sure that the logarithm base and argument are parsed successfully, whether the base is found or not (default base = 10), and whether the argument is a long expression with another logarithm in it.



(a) Logarithm with base as a constant



(b) Logarithm without base and the argument is an expression

Figure 12.9: Multiple Logarithm cases with and without bases

Figure 12.10: Logarithm with multiple logarithms inside of it

12.2.8 Complex Expression Test

In this test, we make sure that a complex expression composed of square root, power, trigonometric functions, logarithm, constants, and fraction is parsed successfully.

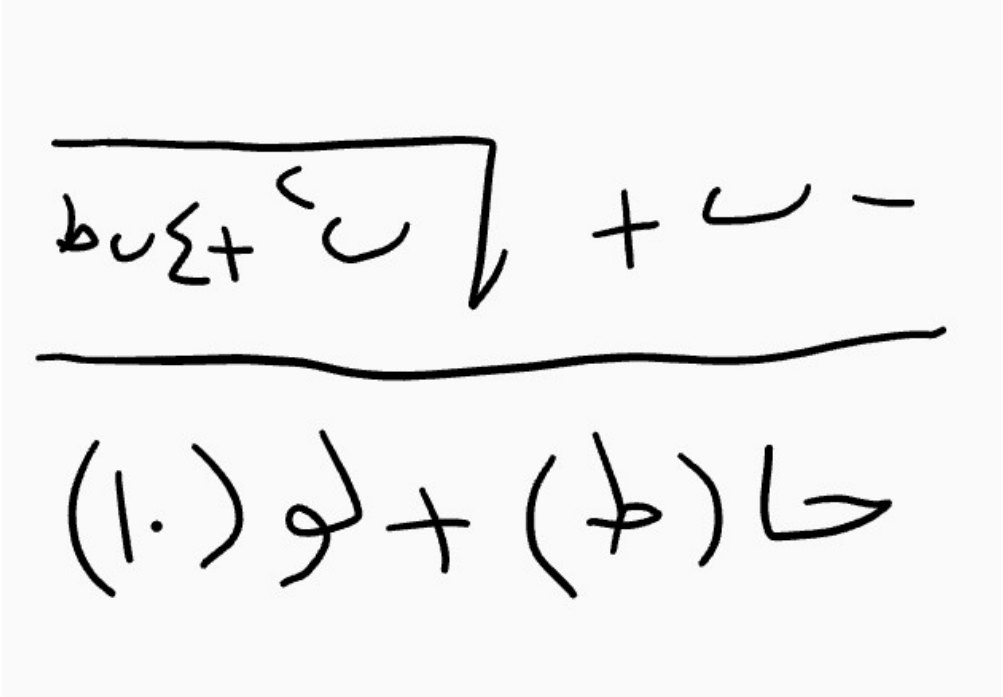


Figure 12.11: Complex expression case

12.3 Mathematical Problems Testing

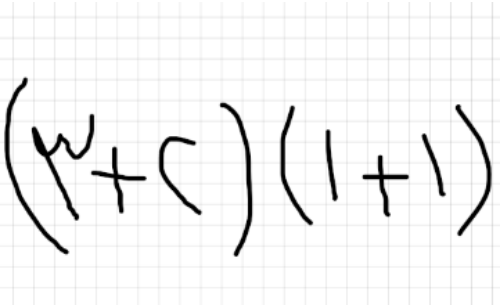
We added some tests before beginning the implementation following the **Tests First Approach**, since we dealt with mathematical equations and the results of the equations were known and deterministic, the expected output was clear and could be covered using some initial tests.

Test-first programming involves producing automated unit tests for production code, before you write that production code. Instead of writing tests afterward (or, more typically, not ever writing those tests), you always begin with a unit test.

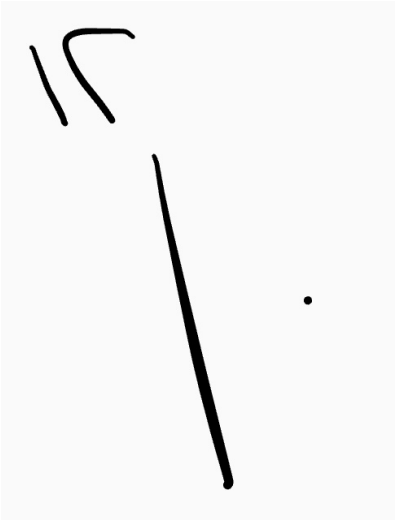
For example, if the user inserts (in Arabic notation) $x^2 - 1 = 0$ as input and requires finding the roots of polynomial, the output that we need to assert is (1,-1)

12.3.1 Simplification Problems Test

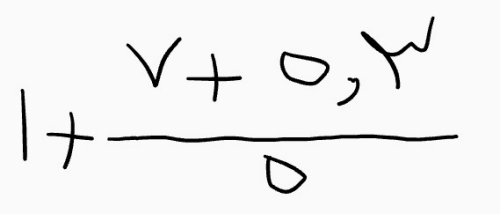
In this test, we make sure that the expression is simplified correctly, and it yields the correct numerical result.



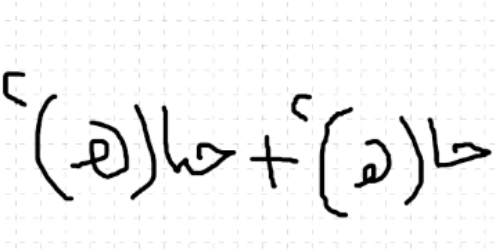
(a) Simple expression reduced to 10



(b) Expression with power reduced to a long numerical result



(c) Simple expression with decimal point and fraction reduced to 3.46

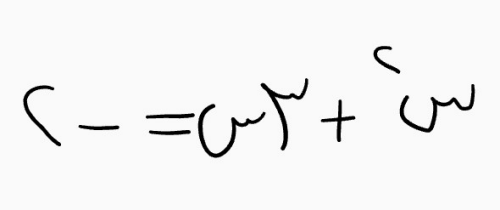


(d) Expression with trigonometric functions reduced to 1

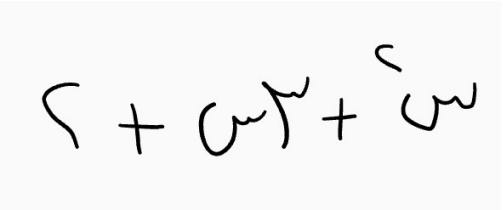
Figure 12.12: Multiple cases with simplification

12.3.2 Polynomial Equations Test

In this test, we make sure that the polynomial equations are parsed successfully regardless of the position of the equal sign, whether it's found or not, and the solution is correct.



(a) Polynomial equation with equal sign and part of the equation at the left hand side



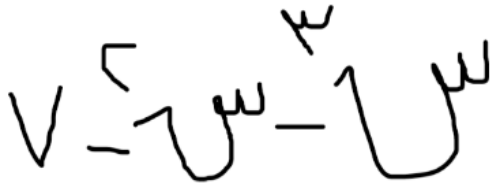
(b) Polynomial equation without an equal sign

Figure 12.13: Different cases of polynomial equations

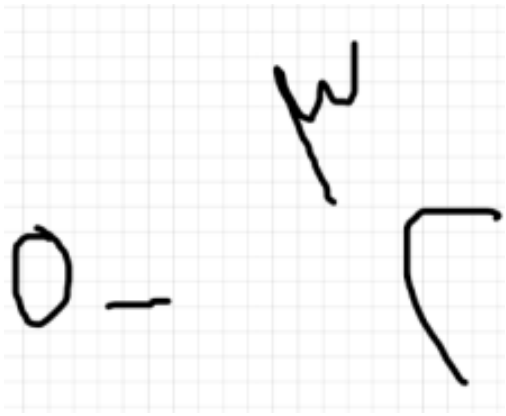
12.3.3 Differentiation Problems Test

In this test, we make sure that the result of differentiation is correct. This needs some testing as it's mainly handled by **SymPy**, and if the expression is mathematically correct and parsed correctly, then it's almost certain will be differentiated correctly.

We assume that the expression is inserted without $\frac{dy}{dx}$.



(a) Differentiation of this $= 3x^2 - 2x$



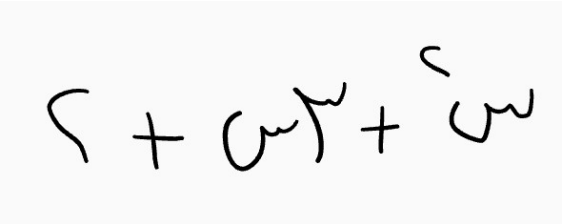
(b) Differentiation results in zero

Figure 12.14: Differentiation different cases

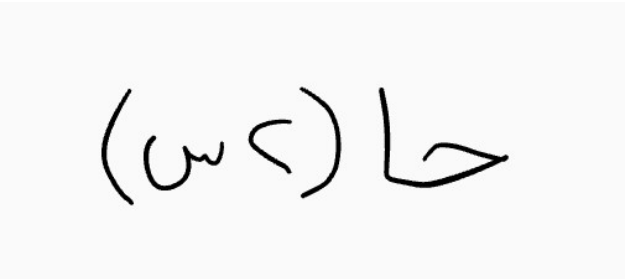
12.3.4 Integration Problems Test

In this test, we make sure that the result of integration is correct. This needs some testing as it's mainly handled by **SymPy**, and if the expression is mathematically correct and parsed correctly, then it's almost certain will be integrated correctly.

We assume that the expression is inserted without $\int$.



(a) Simple integration result $= \frac{1}{3}x^3 + \frac{3}{2}x^2 + 2x$



(b) Integration of trigonometric function result $= -\frac{1}{2}\cos 2x$

Figure 12.15: Integration different cases

Chapter 13

Evaluation

13.1 Introduction

Since the most of the dataset was written by us, we can't depend on the results of the model. So we asked different people to send some equations and symbols to us to test the model. More than 20 people thankfully have helped us and each one of them have sent images to us. We asked them to write symbols like that:

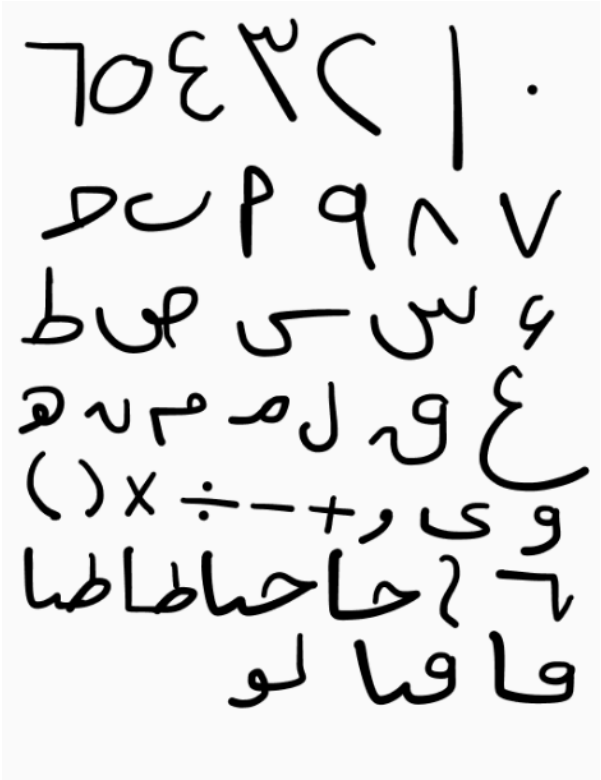
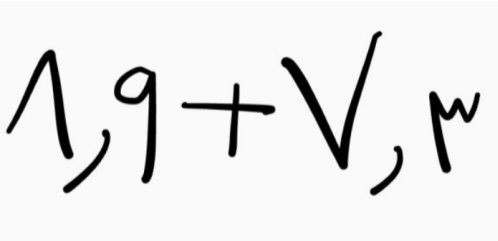
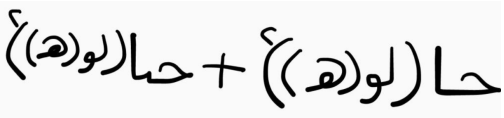


Figure 13.1: Evaluation symbols

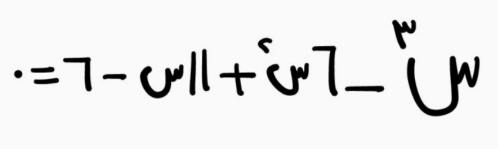
We also asked them to write equations and we gave them samples that cover multiple important things so that they may write look like them:



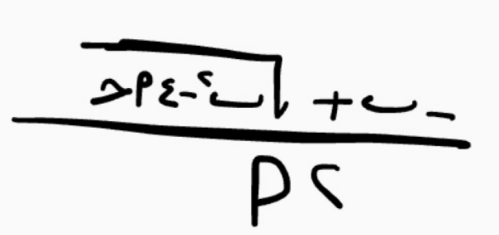
(a) Simple equation with decimal point



(b) Expression with trigonometric functions



(c) Simple polynomial equation



(d) Complex expression with square root and fraction

Figure 13.2: Evaluation expressions

13.2 Difficulties

Some volunteers don't know the assumptions, so they have violated those assumptions i.e

1. They wrote the symbols next to each other without gaps

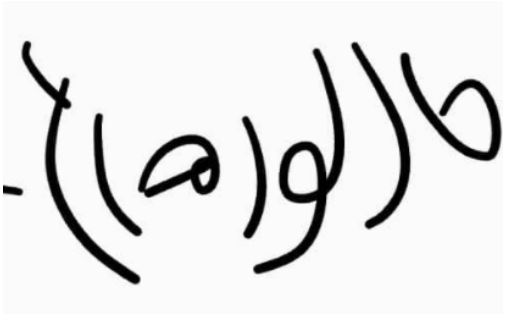


Figure 13.3: Example of expression without gaps between symbols

2. They wrote the exponent in wrong position

Figure 13.4: Example of expression with exponent in wrong position

3. They didn't write the fraction right with respect to our assumptions

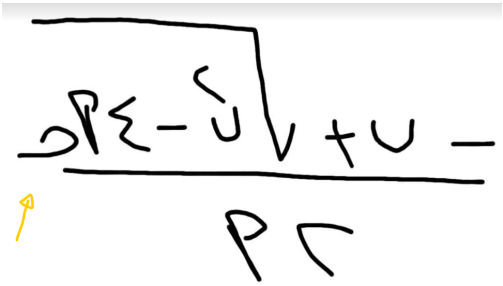


Figure 13.5: Example of expression with short fraction length violating the assumption

4. The "*fasla*" looks like the '*waw*' and this happened with most of the volunteers

Figure 13.6: Example of expression with wrong decimal point

5. The misclassifications were expected with some symbols but after testing other unexpected classification appeared
   Expected:
   - 4 and ayen
   - 2 and)
   - Fasla and waw

   Unexpected:
   - Haa and jeem
   - Baa and nun

- Baa and lam
- Jeem and dal
- Integration with Jeem
- Haa with five
- Sqrt with 6

Some of unexpected misclassifications were because of the bad handwriting way the volunteers had, i.e. The model has seen this baa as lam and honestly it looks like lam

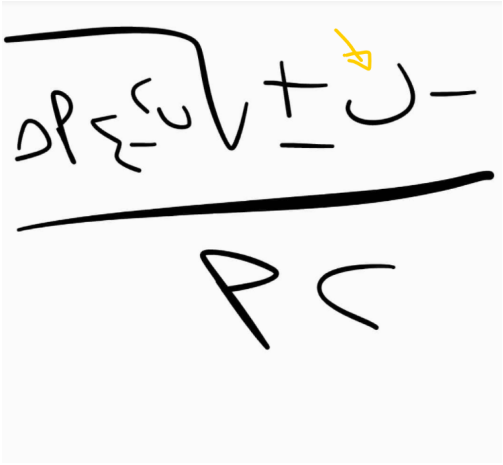


Figure 13.7: Example of expression with misclassification due to bad handwriting

13.3 Modifications

Our solution to those difficulties is to edit the images that the volunteers have sent to us but also to keep their writing pattern as much as possible.

Some of these modification are as follows:

1. If the equation is not in straight line we rotate it to be in straight line

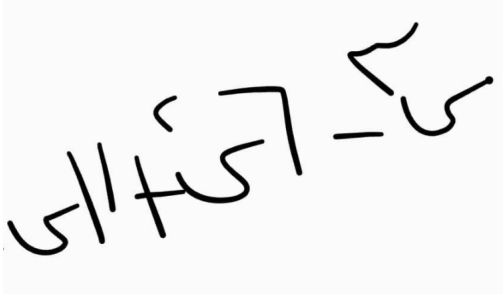
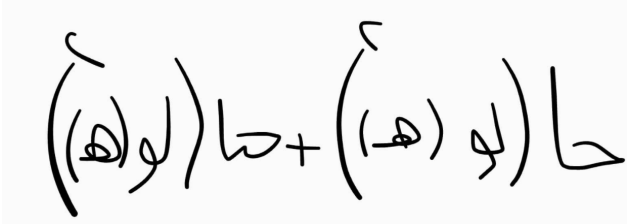
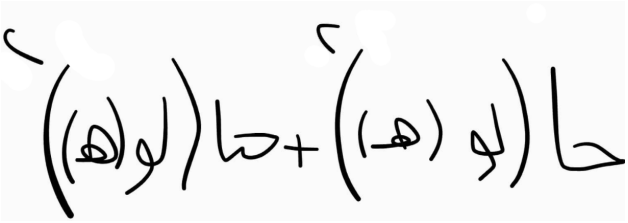


Figure 13.8: Rotated expression which needs to be horizontal

2. If the exponent is not in the right place we move it to be in the right place



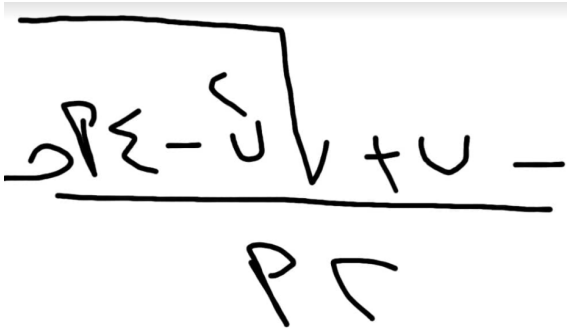
(a) Wrong exponent position



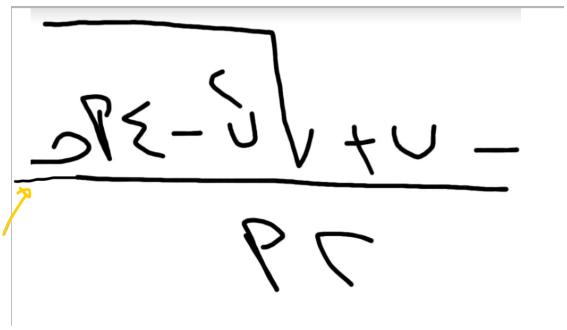
(b) Corrected exponent position

Figure 13.9: Example of wrong (a) and corrected (b) exponent position

3. If the volunteer writes the numerator and the denominator but with short fraction line so we extend this fraction line to cover both the numerator and the denominator



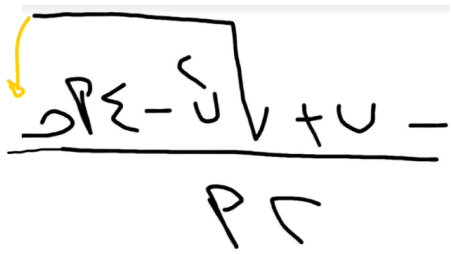
(a) Wrong fraction line length



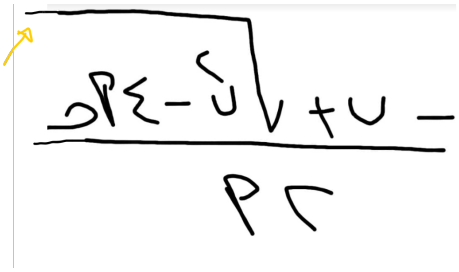
(b) Corrected fraction line length

Figure 13.10: Example of wrong (a) and corrected (b) fraction line length

4. Some volunteer writes short square root that doesn't cover all of the symbols inside the square root so we extend that square root to cover all symbols



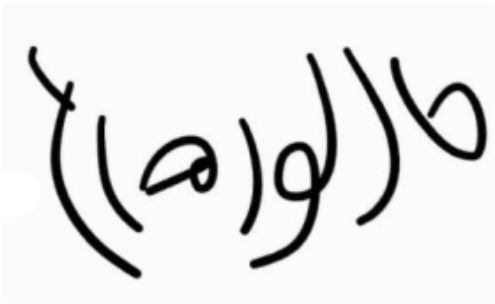
(a) Wrong square root



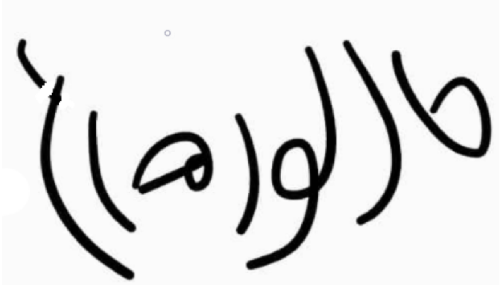
(b) Corrected square root

Figure 13.11: Example of wrong (a) and corrected (b) square root

5. If the symbols were not separated by a gap, we separate them



(a) Wrong gaps between



(b) Corrected gaps between symbols

Figure 13.12: Example of wrong (a) and corrected (b) gaps between symbols

13.4 Evaluation Results

- **Symbols**: We have 482 symbols and 44 of them get predicted wrong with **error rate = 9.13%** and **accuracy rate = 90.87%**
- **Equations**: Total equations = 79 equations.
  - Before editing: equations predicted right = 43, equation predicted wrong = 36, **accuracy = 54.43%**, **error rate = 45.56%**
  - After editing: equations predicted right = 55, equation predicted wrong = 24, **accuracy = 69.6%**, **error rate = 30.4%**

13.5 Conclusion

It is hard to cover all patterns using our dataset.

If all symbols in the equation was predicted right **except one symbol**, we consider this equation is wrong and this has been happening a lot with equations containing "fasla" and the equations that contain expected misclassified symbols i.e: 4 and ayn and so on.

And because of that, we provided **the erase feature** in our application so if the equation contains only a misclassified symbol, the user can easily erase this symbol and rewrite it again.

And with some practice by the users, they will learn the mistakes that the application might make besides the assumptions.

It is sure that the more this application is used, the fewer errors will occur.

Chapter 14

Application Architecture

14.1 MVC Architecture

14.1.1 Motivation

A typical Android app contains multiple app components, including activities, fragments, services, content providers, and broadcast receivers. Most of these app components are declared in the app manifest. The Android OS then uses this file to decide how to integrate the app into the device's overall user experience. Given that a typical Android app might contain multiple components and that users often interact with multiple apps in a short period of time, apps need to adapt to different kinds of user-driven workflows and tasks.

Mobile devices are also resource-constrained, so at any time, the operating system might kill some app processes to make room for new ones.

Given the conditions of this environment, it's possible for the app components to be launched individually and out-of-order, and the operating system or user can destroy them at any time. Because these events aren't under our control, it shouldn't store or keep in memory any application data or state in your app components, and your app components shouldn't depend on each other.

As Android apps grow in size, it's important to define an architecture that allows the app to scale, increases the app's robustness, and makes the app easier to test.

The most important Any android application should follow **separation of concerns**.

Separation of concerns (SoC): is a design principle for separating a computer program into distinct sections. Each section addresses a separate concern, a set of information that affects the code of a computer program. A concern can be as general as "the details of the hardware for an application", or as specific as "the name of which class to instantiate".

14.1.2 Model-View-Controller (MVC)

is a software architectural pattern, which divides the application logic into three interconnected elements, separating internal representations of information of the application; it specifies that an application consists of a data model, presentation information, and control information. The pattern requires that each of these be separated into different objects which are:

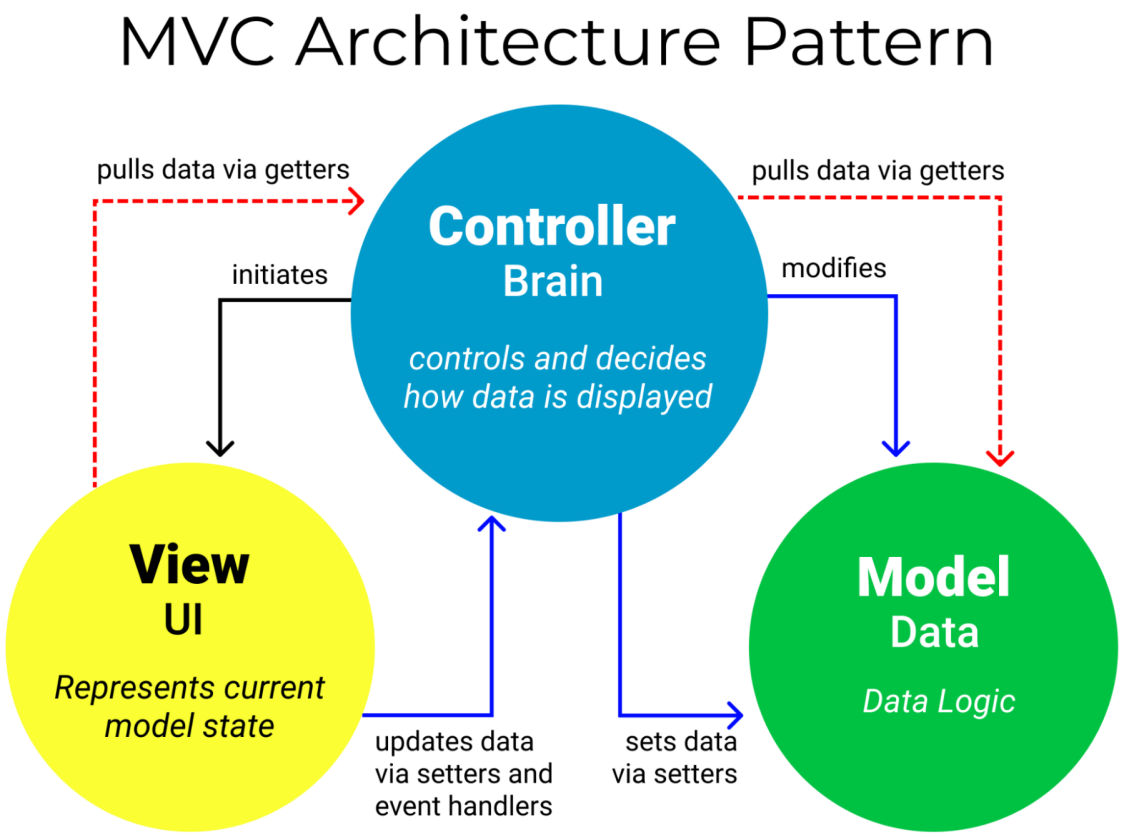


Figure 14.1: MVC Architecture Pattern

- **Model**: It is used as an application object of the application data administration. It responds to information requests about its status, which usually come from the view, as well as to statements for status changes, which are usually sent by the controller. In this way, only the model is used to process data internally, without making reference to the application and its user interface. There can be different views for a model, which can be implemented using different view pages.
- **View**: It handles the graphical and textual output at the interface and therefore represents the input and output data in each interface element, such as push-buttons, menus,

dialog boxes and so on. The view takes care of visualization. To visualize the status, the view queries the model, or the model informs the view about possible status changes.

- **Controller**: It interprets and monitors the data that is input by the user using the mouse and the keyboard, causing the model or the view later to change if necessary. Input data is forwarded and changes to the model data are initiated. The controller uses the model methods to change the internal status and then informs the view about this. This is how the controller determines reactions to the user input and controls processing.

The view and the controller together form the user interface.

Since the model does not recognize either views or the controller, internal data processing is detached from the user interface. Since the model does not recognize either views or the controller, internal data processing is detached from the user interface.

14.1.3 MVC Implementation

After giving an intuition of using MVC and illustrating how it works. In this section, we will illustrate how we implemented this architecture in our application, But before that let us mention the concept of **Path** which is the building block of Drawing.

The Path is a class that encapsulates compound (multiple contour) geometric paths consisting of straight line segments, quadratic curves, and cubic curves. It can be drawn with **canvas.drawPath(path, paint)**, either filled or stroked (based on the paint's Style), or it can be used for clipping or to draw text on a path.

- In our application, we treat The **DrawViewManager.java** as our **Model**, it's responsible for holding important data such as:
  - What and which paths and drawings are currently drawn on our View, how it's drawn and in which order.
  - Order of commands that were given by the user to our application e.g. (Draw-Erase).
  - As well as being responsible for tending to Undo and Redo operations that affect the Paths.

```
class DrawViewManager {
  List<Path> drawnPaths;
  List<Pair<Character, Path>> previousCmd, nextCmd;
  HashMap<Path, Integer> pathRefs;
  FloatingActionButton redoFab, undoFab;
  Button clearBtn;
  int mode;
  static DrawViewManager instance;
}
```

- We consider both **fragment_canvas.xml** and **DrawView.java** to be both our main **View** components.
- We consider the **CanvasFragment.java** to be our main **Controller** that's responsible for handling the user interactions with both views and changing the state of the model in response.

14.2 Implemented Features

14.2.1 Drawing Area

- It's our main application view component, it consists of a Canvas which shows Paths drawn on it, the Canvas at the start of application only shows an empty Gridded area.
- Whenever the user touches the screen, the Canvas transforms the user's touch event into a drawn pixel on the Canvas, as the user moves his fingers on the screen, it forms connected drawn pixel (which we later will call it as Path), It lets the user draw on the screen whatever s/he likes.
- Users should use this area to draw any equation it wants to apply an operation on.

14.2.2 Different Equation Operations

- The user is allowed to choose different operations to be applied to the drawn equation.
- We use a bottom navigation drawer that's opened using a FAB (Floating Action Button) to show the user which operations could be done and which operation is currently chosen.

14.2.3 Drawing Tools

- The user has different abilities to interact with what is currently drawn in the drawing area using different tools e.g. (Pen, Eraser, Move).
- We implemented this using a bottom navigation view in which the user can click or swipe on to change the tool that's currently selected.
- Following is the description of what each tool does when chosen:
  - Pen: while this tool is selected, any user interaction with the drawing area is translated into a drawn black-filled Path (white-filled if the current theme Dark).
  - Eraser: while this tool is selected, if any user interaction with the drawing area is intersecting with the boundary box of a drawn path, it gets deleted from the drawing area.
  - Move: while this tool is selected, any user swipe on the drawing area:

    * If done with only one finger, it does a movement with respect to the drawn paths in the same direction of user swipe.
    * If done with two fingers, the drawn paths get minimized or maximized based on the relative distance between the user's fingers before and after the interaction.

14.2.4 Undo & Redo operations

- The user has the ability to undo or redo any drawing or erasing operations he has done.
- This was presented to the user using 2 FABs in the left bottom corner with forward and backward arrows as their icons.
- Like any Undo/Redo options in other apps, if the user does some undo-s operations, he will have the ability to redo them, but if s/he then use the pen/eraser to draw or erase any drawing, the stored Redo-s are completely neglected.

14.2.5 Equation Image storing

- The user has the ability to store a static image of the current equation drawn, in the external storage.
- If the user wants to resolve an equation, this previously stored image can be resubmitted on the fly to the application to be solved again.

14.3 Application Design

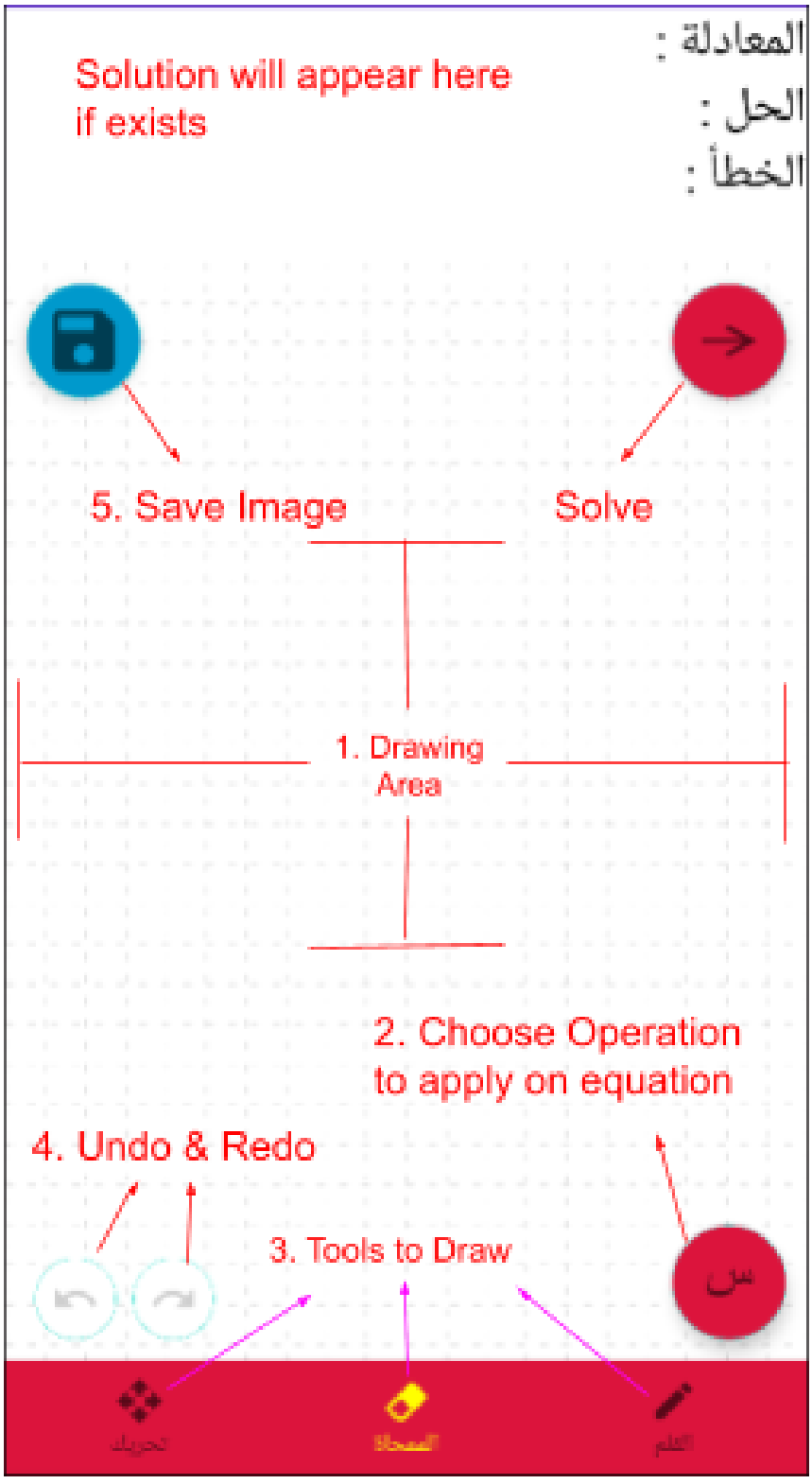


Figure 14.2: Application's User Interface

Chapter 15

Networking

15.1 Introduction

Networking is a crucial factor in mobile development. Most, if not all mobile applications incorporate networking on some level. Applications are either sending or receiving information. Initially, developers did networking on the main thread. This made applications less user-friendly since screens would "freeze".

15.2 Application Networking

Networking on the main thread stopped after the Honeycomb version was released. Google then developed Volley in 2013.

Volley offered something better: It was faster, provided better functionality, a simpler syntax, etc. Still, there was room for growth when it came to networking.

15.2.1 Volley

Advantages

- It saves time from writing the same network/cache code over and over.
- It can run many concurrent network connections.
- It provides tracing and debugging tools.
- It enables the automatic scheduling of network connections.
- It enables caching.
- It supports OkHttp.

Disadvantages

- It's not suitable for streaming and large downloads.
- It does not have proper documentation yet.
- It's slower compared to ***Retrofit***.
- It has a complex code structure.

15.2.2 Retrofit

Introduction

We decided to use retrofit and that because of:

- Retrofit is a type-safe HTTP networking library used for Android and Java.
- Retrofit was even better since it was super fast, offered better functionality, and even simpler syntax.
- Most developers since then have switched to using Retrofit to make API requests.

Uses

Retrofit is used to perform the following tasks:

- It manages the process of receiving, sending, and creating HTTP requests and responses.
- It alternates IP addresses if there is a connection to a web service failure.
- It caches responses to avoid sending duplicate requests.
- Retrofit pools connections to reduce latency.
- Retrofit resolves issues before sending an error and crashing the app.

Advantages

- It is very fast.
- It enables direct communication with the web service.
- It is easy to use and understand.
- It supports request cancellation.
- It supports post requests and multipart uploads.

- It supports both synchronous and asynchronous network requests.
- Supports dynamic URLs.
- Supports converters.

Disadvantages

- It does not support image loading. It requires other libraries such as Glide and Picasso.
- It does not support setting priorities.

Classes Used

- Model class - This class contains the objects to be obtained from the JSON file.
- Retrofit instance - This Java class is used to send requests to an API.
- Interface class - This Java class is used to define endpoints.

15.2.3 How do we send our data to the server ?

- After reading the advantages and disadvantages of Volley and Retrofit, we decided that it's better to use Retrofit than Volley as Retrofit offers more flexibility, robustness and simplicity in sending HTTP requests.
- At first, we used retrofit by
  - first storing the equation image as file in the user's external storage
  - then, hand retrofit a reference to the file to be sent in a multipart HTTP request

15.2.4 Challenges

This previous method was inappropriate when we started handling our application on higher Android APIs, specifically any API ¿ 29 (Android 9.0)

As stated here[14], On higher APIs, some third party libraries would be unable to access a file in the external storage of a user using what is called a "direct path" instead of the new "URI path" used in higher APIs.

This resulted in "*open failed: EACCESS (Permission denied)*" exception being thrown from within Retrofit.

- To overcome this previously stated challenge, instead of passing a File reference that contains a "direct path", we converted and sent our image files as a byte array from our android application to the server, which we converted to an image file back at the server.

15.3 Server Networking

For the server part of the networking, we would receive http requests of the images from the application using Retrofit, but we're introduced to the choice of which platform to deploy our server on.

There are many options including: Amazon AWS, Google cloud, MS Azure, and Heroku. We chose Heroku because of its ease of deployment, integration with Git, and its free plan was more than enough for our simple flask web server.

15.3.1 Heroku[15]

Introduction

Heroku is a cloud platform as a service (PaaS) supporting several programming languages. One of the first cloud platforms, Heroku has been in development since June 2007, when it supported only the Ruby programming language, but now supports Java, Node.js, Scala, Clojure, Python, PHP, and Go.

For this reason, Heroku is said to be a polyglot platform as it has features for a developer to build, run and scale applications in a similar manner across most languages..

History

Heroku was initially developed by James Lindenbaum, Adam Wiggins, and Orion Henry for supporting projects that were compatible with the Ruby programming platform known as Rack.The prototype development took around six months.

Later on, Heroku faced setbacks because of lack of proper market customers as many app developers used their own tools and environment.

In January 2009, a new platform was launched which was built almost from scratch after a three-month effort. In October 2009, Byron Sebastian joined Heroku as CEO. On December 8, 2010, Salesforce.com acquired Heroku as a wholly owned subsidiary of Salesforce.com.

On July 12, 2011, Yukihiro "Matz" Matsumoto, the chief designer of the Ruby programming language, joined the company as Chief Architect, Ruby. That same month, Heroku added support for Node.js and Clojure.

On September 15, 2011, Heroku and Facebook introduced Heroku for Facebook. At present Heroku supports Redis databases in addition to its standard PostgreSQL.

Etymology

The name "Heroku" is a portmanteau of "heroic" and "haiku".The Japanese theme is a nod to Matz for creating Ruby.

The name itself is pronounced similarly to the Japanese word meaning "widely" (hiroku), though the creators of Heroku did not want the name of their project to have a particular meaning, in Japanese or any other language, and so chose to invent a name.

Architecture

Applications that are run on Heroku typically have a unique domain used to route HTTP requests to the correct application container or dyno.

Each of the dynos are spread across a "dyno grid" which consists of several servers. Heroku's Git server handles application repository pushes from permitted users.

All Heroku services are hosted on Amazon's EC2 cloud-computing platform.

15.4 Using Heroku

15.4.1 Deploying with Git[16]

1. Prerequisites: git and Heroku CLI are installed.
2. Local git repo already exists whether using `git clone <repo> <directory>` or `git init` in the current project directory.
3. For creating the Heroku app, there is 2 ways
   (a) For a new app, use `heroku create -a example-app` and then `git remote -v` which confirms that a remote named heroku has been set for the app.
   (b) For an existing app, add a remote to your local repo with heroku `git:remote -a example-app` command.
4. Check the following required files, **requirements.txt**, **runtime.txt**, and **Procfil**e for running the web app. Also check that the repo size is not greater than **600 MB** which is heroku's limit.
5. Deploy your code using the command `git push heroku main`.

15.4.2 GitHub Integration (Automatic Deploys)[17]

1. Enable GitHub integration in the deploy tab of apps in the *Heroku Dashboard* and authenticate with GitHub.

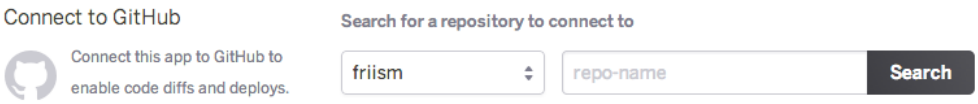


Figure 15.1: Enabling Git integration

2. After that, you can enable automatic deploys for a GitHub branch, Heroku builds and deploys all pushes to that branch.

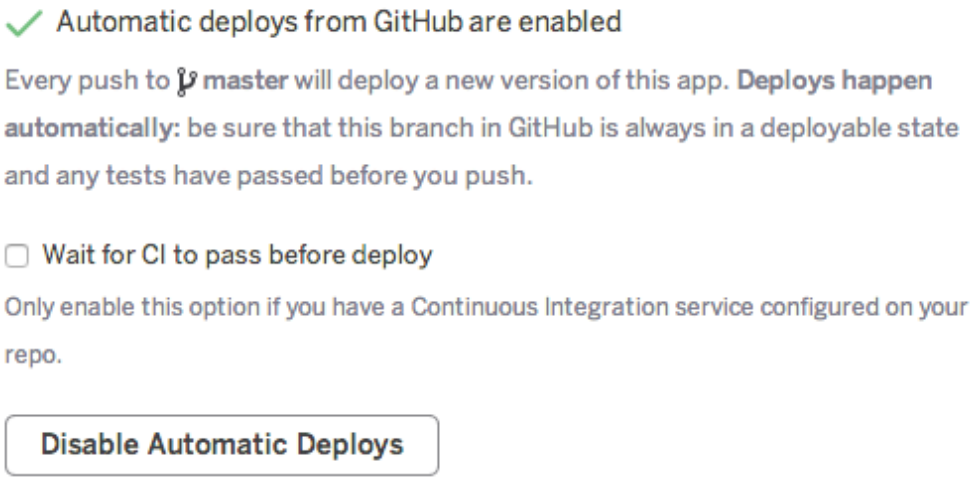


Figure 15.2: Automatic deployment enabling from Git

If you've configured your GitHub repo to use automated Continuous Integration, you can check the "Wait for CI to pass before deploy" checkbox. When enabled, Heroku will only auto-deploy after all the commit statuses of the relevant commit show *success*.

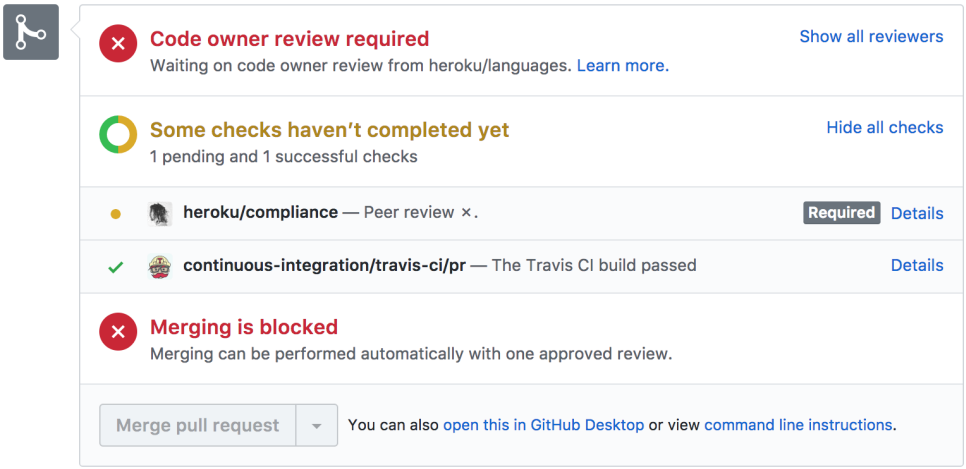


Figure 15.3: Wait for CI to pass

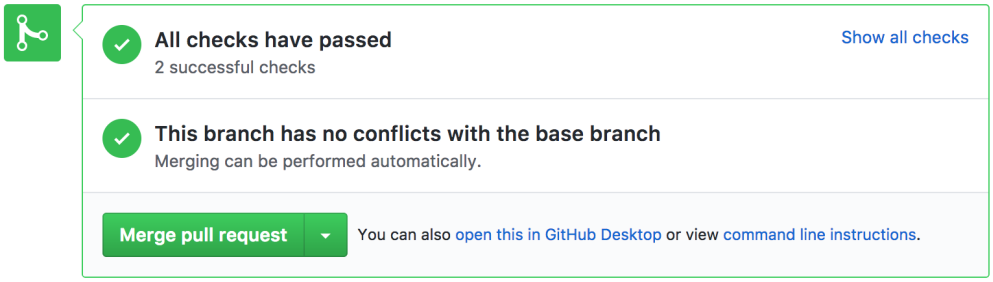


Figure 15.4: CI passed tests successfully

Chapter 16

Future Work

Instead of using CNN model to predict symbols only, sequence model could be used to predict the whole equation.

The assumptions that used to relax the problem could be considered and implemented .

The project is open to handle more problems and recognize more symbols to work with. This includes but not limited to:

- handling limits which requires the limit symbol and its arrow $\lim_{x \to \infty}$
- adding summation symbol $\sum$ and calculating summation of a series
- parsing integration sign $\int$ and differentiation sign $\frac{dy}{dx}$ and handling the different cases in which they appear
- adding partial integration $\int \int f(x, y)\, dx\, dy$ and partial derivative $\frac{\partial u}{\partial t}$
- adding complex numbers

The app is also open to more features like:

- drawing the graph of an equation in addition to presenting its solution
- adding the ability to get symbols from keyboard rather than handwriting or using a photograph

Bibliography

[1] F. Hasan, S. N. Shuvo, S. Abujar, and S. A. Hossain, "Bangla handwritten math recognition and simplification using convolutional neural network," in *Emerging Technologies in Data Mining and Information Security*, Springer, 2021, pp. 133–141.

[2] K. K. Ayeb, Echi, A. Kacem, and A. Belaïd, "A syntax directed system for the recognition of printed arabic mathematical formulas," in *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*, IEEE, 2015, pp. 186–190.

[3] I. H. Ali and M. A. Mahjoub, "Dynamic random forest for the recognition of arabic handwritten mathematical symbols with a novel set of features.," *Int. Arab J. Inf. Technol.*, vol. 15, no. 3A, pp. 565–575, 2018.

[4] K. Davila, S. Ludi, and R. Zanibbi, "Using off-line features and synthetic data for on-line handwritten math symbol recognition," in *2014 14th International Conference on Frontiers in Handwriting Recognition*, IEEE, 2014, pp. 323–328.

[5] F. Alvaro, J.-A. Sánchez, and J.-M. Benedí, "Classification of on-line mathematical symbols with hybrid features and recurrent neural networks," in *2013 12th International Conference on Document Analysis and Recognition*, IEEE, 2013, pp. 1012–1016.

[6] Ali, I. Hadj, Mahjoub, and M. Ali, "Database of handwritten arabic mathematical formula images," in *2016 13th International Conference on Computer Graphics, Imaging and Visualization (CGiV)*, 2016, pp. 145–149. DOI: 10.1109/CGiV.2016.36.

[7] A. Nazemi, N. Tavakolian, D. Fitzpatrick, C. Y. Suen, and et al., "Offline handwritten mathematical symbol recognition utilising deep learning," *arXiv preprint arXiv:1910.07395*, 2019.

[8] *SymPy Library*, https://www.sympy.org/.

[9] *OpenCV Library*, https://opencv.org/.

[10] S. Suzuki and K. Abe, "Topological structural analysis of digitized binary images by border following," *Comput. Vis. Graph. Image Process.*, vol. 30, pp. 32–46, 1985.

[11] *Contours tutorial*, https://docs.opencv.org/3.4/d4/d73/tutorial_py_contours_begin.html.

[12] A. El-Sawy, M. Loey, and H. EL-Bakry, "Cnn for handwritten arabic digits recognition based on lenet-5," in *AISI*, vol. 5, Springer International Publishing, 2016, pp. 566–575.

[13] M. Torki, M. E. Hussein, A. Elsallamy, M. Fayyaz, and S. Yaser, "Window-based descriptors for arabic handwritten alphabet recognition: A comparative study on a novel dataset," *arXiv preprint arXiv:1411.3519*, Nov. 2014.

[14] *Android Direct File Paths*, https://developer.android.com/training/data-storage/shared/media#direct-file-paths.

[15] *Heroku Wiki*, https://en.wikipedia.org/wiki/Heroku.

[16] *Heroku Git Deployment*, https://devcenter.heroku.com/articles/git.

[17] *Heroku Git Integration*, https://devcenter.heroku.com/articles/github-integration.