Design Document

Handwritten Mathematical Expressions

Team: Runting Shao, Ningji Shen, Chang Xu,Summer Ai, Zihui Zhang | DATA 515 - Wi 21

1. Overview

Handwritten Mathematical Expression is a full stack web application system that takes in user's handwritten image (.png format) of a mathematical expression as input and uses machine learning and computer vision to recognize the numbers and the operators in the expression, performs the calculation in the background and displays both the expression using LaTex and the calculated result on the web page.

There are three major pieces in this system: a web page powered by Python Flask, HTML and JavaScrip that takes input from and deliver results to the user in print-friendly LaTex format; an image processing module that takes user's input image and saves them into multiple images that consists the corresponding character; and a machine learning model that takes in the segments and recognize the corresponding characters.

Our program intends to relive people's pain in typing and performing mathematically heavy calculations, and our main user cases are STEM students of all grades as well as faculty members in the education industry. It is easy to use as it can quickly convert the handwritten expressions to machine readable results.

2. Data And Data Preprocessing

2.1 Introduction of Dataset

Source: https://www.isical.ac.in/~crohme/CROHME_data.html

Our machine learning model uses data from competition CROHME 2012 and 2013 for training and testing. This dataset aims at bringing the researchers under a common platform so that they share the same dataset of handwritten mathematical expressions for their respective research and report performance of their systems on a common test data.

The datasets includes the following items:

CROHME2012_data : all data from the CROHME 2012 competition:

- testData : inkml test files without ground truth
- testDataGT : inkml test files with ground truth
- trainData : inkml train files with ground truth
- gram : xml grammars and symbol lists for parts I, II and III
- lists : lists of inkml files and latex expressions for parts I, II and III

CROHME2013_data : all data from the CROHME 2013 competition:

- TrainINKML : all training inkml files sorted by origin
- TestINKML : inkml test files without ground-truth, used to run the participants' systems.
- TestINKMLGT : inkml test files with ground-truth, used to evaluate the participants systems with the evalinkml tool.
- Test_LG/Test2012LG Test_LG/Test2013LG: label graph version of the test files for 2012 and 2013 dataset, using inherited edges (so the graphs are DAGs).
- Test_LG/Test2012LG_TREE Test_LG/Test2013LG_TREE: label graph version of the test files for 2012 and 2013 data set, without inherited edges (so the graphs are trees).

2.2 Data Preprocessing

All the data files in CROHME 2012 and 2013 contain full mathematical expressions and were stored with inkml format, whereas our machine learning model intakes one character in PNG image format. Therefore, we first use an open source tool (RobinXL) to convert all the inkml files to png files, and then perform image preprocessing that separates an expression into segments. We will discuss the image preprocessing functionality later in the Project Components section.

For training dataset, after the steps of segmentation and labeled according to the given ground truth, all the training data has been kept in ‘data/trainData’ folder with labels. Since my segmentation method can not allow all images to be cropped perfectly, I exclude those labels which do not have enough cropped images to train the model. Each cropped image after mapping with the ground truth label will be automatically inferred to the subdirectory ‘ under ‘data/trainData’, where each subdirectory contains the images for a specific symbol of integer.

The testing dataset we used to test the accuracy of our model is divided into two parts: 1. I extracted 30 testing images from CROHME2012_data and CROHME201_data which concludes only the integers and symbols that our model can recognize. All the testing mathematical expressions are divided into easy, median or hard groups based on their length and writing format. 2. We also prepared 32 images which include expressions that are written by ourselves. Each testing image is cropped and stored in a separate folder along with a pickle file, which contains the position information of each character in the image.

3. Project Architecture

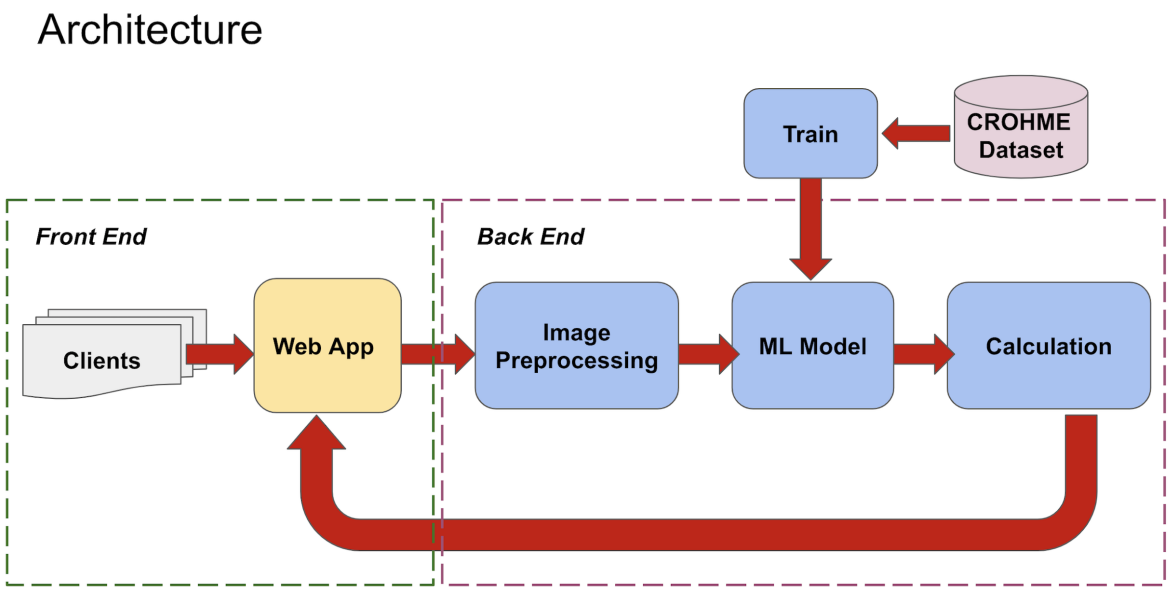

Figure 1: Project Architecture

3.1 Front End

The front end consists of three HTML templates (an index page, a waiting page and a result page) that use jinja2 and a python script, which uses Flask, a micro web framework, to serve the web application. Pages redirect as the user clicks buttons or automatically as the background calculation is done. The logic between pages is controlled by JavaScript code. The user interface is able to accept valid user input images in .png format and render the calculated results in latex and plain text format on the result page once the calculation process is complete. The principle is to do an ajax call to trigger our main python script in the background and run a callback function when it is returned. Everytime after the results are shown, the user can click the “Start Over” button to perform the task repeatedly.

3.2 Back End

3.2.1 Image Preprocessing

For each original PNG file, we convert the image into a binary image in order to reduce the noises with GaussianBlur and opening operation. Figure 1 and 2 is an example of the image before and after this step. We then define the border of each character by applying the projection algorithm to compute the contour of each character in the image. This allows us to crop each

character according to its contour. Figure 4 is an example of defining the contour of ‘c’ over Figure 3.

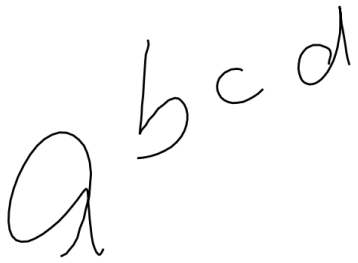

Figure 2: Original Image

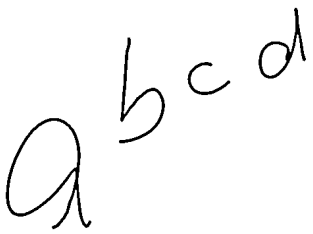

Figure 3: Binary Image

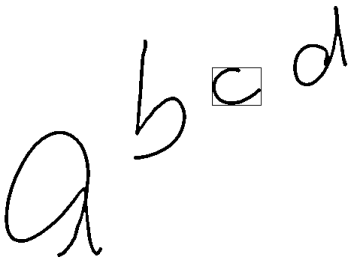

Figure 4: Contour of “c”

Figure 5: cropped “c”

Figure 6: “c” after skeletonize

After separating the cropped character from the original image (Figure 5), we resize the segment image to 32 * 32 pixels and skeletonize it (Figure 6). After performing image preprocessing, the original image (Figure 2) will be processed into the following four PNG images (Figure 7) and saved to a local folder.

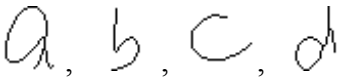

Figure 7: Image Preprocessing Outputs

3.2.2 Machine Learning Model

We build our model based on Yann LeCun's LeNet-5 convolutional neural networks. The LeNet-5 architecture consists of two sets of convolutional and max pooling layers, followed by a flattening convolutional layer, then two fully-connected layers and finally a softmax classifier. In order to build this network, we utilize Preprocessing, Models, Layers and Callbacks APIs from Keras, which runs on top of the machine learning platform

TensorFlow. With the functions from the Preprocessing API, we are able to read the folder names as the list of classes that need to be trained on, and with the functions from the Models, Layers and Callbacks APIs along with our CROHME datasets, we are able to compile, fit and evaluate the LeNet-5 convolutional neural networks. We saved our trained model into a single h5 file, which will be loaded into by our main module. Our model achieved a metrics of:

- Test Loss: 0.43
- Test accuracy: 0.86

With our model's weights and configuration saved locally, we can easily load it every time as the user calls the module and do the prediction efficiently with low cost. We also save our training module in a notebook along with its checkpoints and outputs for visibility and future improvements.

3.2.2 Post-processing

The post-processing module intakes a list of labels generated by the model and a list of coordinates of the corresponding character's position, and correlates the two into strings which correspond to the original expression. According to the positions of the characters, this module determines the spatial position relationship between two adjacent characters as power, division or parallel relationship, and returns two strings according to the order of operations. One of the strings is returned to the calculation module to perform the calculation, and the other string is fed into a function to generate a LaTex version of the expression, which will be written into a TXT file.

3.2.3 Calculation

The calculation module intakes a string, processes the string into a binary tree, and outputs the results into a TXT file. We referenced the structure built by Huzaifa-Imran that splits a string into a list of floats and operators, converts the list of Infix notations into Postfix notations, and then uses a binary tree to perform the calculation. In this project, we only support basic arithmetic such as plus, minus, times, divide, and power. Our calculation module does not support anything with an unknown variable such as solving for x.

4. Future Implementation

For the front end processes, one possible improvement is that the user interface currently displays the recognized formula in latex as well as the calculated result as an integer

on the webpage, both items are view-only. For the future, we want to make the two results editable to the users, so maybe displaying both results in text boxes. Another thing is that we can make the user interface client-side and use an object tree to append new nodes to the tree and thus show the newly generated results on the same web page without redirecting to a different page. This can be done with the introduction of frameworks such as React.js or Angular.js integrated with the current JavaScript. We can also make the waiting process smoother by adding a web loader. The third improvement we are able to make is that right now, we assume our users are good and follow our instructions on the webpage - they only upload images in .png format, but in the future, we want to add error handling messages if the user does not follow our instructions and upload images of invalid formats.

For the back end, what we can do to improve our system is to modify the image preprocessing module so that we could improve the accuracy of image segmentation. A possible solution to this problem is to compute the average weight of characters to determine superscripts, exponents, or written characters that are too separated since everyone has his own writing habits and it is mostly likely that each person's writing characters are similar in length. Another thing is to improve the accuracy of our Lenet model. Possible approaches include adjusting the kernels or the strips of convolutional layers or getting more training data for the model.

5. Project Plan

Assignee List & Progress Update :

Task	Assignee	Status	Date
Proposal & Technology Review	As a group	Completed	1/23/2021
Web Page Frame	Chang	Completed	2/28/2021
Call Back Functions	Chang	Completed	3/10/2021
Data Processing	Runting	Completed	3/1/2021
Image Preprocessing	Runting	Completed	3/1/2021
Lenet-5 Model	Ningji	Completed	3/2/2021
Model Training & Evaluation	Ningji & Runting	Completed	3/5/2021
Main.py	Ningji	Completed	3/10/2021
Post-processing	Runting	Completed	3/9/2021
Calculation	Ningji/Zihui/Summer	Completed	3/5/2021
Convert to LaTex	Ningji	Completed	3/5/2021

Unit tests for back end modules	Runting/Ningji	Completed	Upon completion of each module
Setup Travis CI	Summer	Completed	3/10/2021
setup.py & requirements	Zihui	Completed	3/10/2021
User Examples	Chang	Completed	3/12/2021
Presentation Slides	Summer	Completed	3/13/2021

6. References

_____"CROHME: Competition on Recognition of Online Handwritten Mathematical Expressions - TC11." *Iapr-Tc11.org*, 2012, www.iapr-tc11.org/mediawiki/index.php/CROHME:_Competition_on_Recognition_of_Online_Handwritten_Mathematical_Expressions.

RobinXL. "RobinXL/Inkml2img." *GitHub*, Oct. 2020, github.com/RobinXL/inkml2img.

Huzaifa-Imran. "Huzaifa-Imran/BinaryTreeCreator." *GitHub*, 8 Feb. 2021, github.com/Huzaifa-Imran/binaryTreeCreator/tree/5cb2c0b969d516948e7cf5c008965324a99ed670.