Final AI Project

Project creators: Daniel, Andrew and Abel

Work distribution:

Daniel:

- Research
- Data cropping + Data editing
- Final report
- Data set formatting and preparation

Andrew:

- Research
- Work with symbol recognition
- Work with data processing and data segmentation
- Work with data training and data recognition

Abel:

- Research
- Work with data segmentation
- Work with equation reconstruction

Dependencies: pickle, numpy, PIL, Pillow, Opencv-python, scikit-image, tensorflow and scipy.

Introduction

My teammates and I were tasked to create a project that can detect handwritten equations. This was a very interesting project however we didn't know where to start. We started by having a group meeting and a group discussion after which we all went to the internet and started researching. We realized that the task was vast because a program that does handwriting recognition needs to recognize a lot of mathematical symbols, writing styles, the combination of rules, etc.. . After a lot of research, we understood the concept of how to make such a project work. What we need to do was as follows:

1. Partition an expression into separate symbols
2. Use a trained convolutional neural network model to recognize each symbol
3. Use algorithm/s to reconstruct equations

After having a rough sketch of what to do we came up with a more detailed plan of action. Firstly we needed to get the data format to the correct size and prepare it to be used in the project. Secondly, we needed to separate the symbols into different parts. Thirdly we need to recognize the symbols that we have. And fourthly we need to reconstruct the equation.

(1) Data Formatting and preparation

We were given 860 images on Latte. However, with those images, there was a big problem. The problem consisted of not having individual symbols like the division symbol, identification of letters, identification of numbers, etc. What we have done was use code to break the images into sub-images ex: suppose we have an image that has the equation “x + y” then the code breaks the images into 3 separate images: “x”, “y”, and “+”. Via the usage of the code to break the 860 images we got around 3 to four thousand sub-images. These sub-images consisted of symbols like: “a”, “b”, …, “1”, “2”, …, “/”, “+”, “-”, etc.. . We then had to manually sort the 3-4 thousand images into folders ex: suppose we say an image of “a” then we put it in the folder named “a”...

After this was done we had some problems. The problems were as follows: the “i” looked like the “l” and the “l” looked like a “1”, the “c” looked like a “(”, the “0” looked like “o”, etc.. . What we have done was create around 100-200 images for each symbol so that the program could recognize between the different symbols better. It was a very long and tiring job but in the end, it paid off because the accuracy was good. The next step in the process was to prepare the images. We used a library called Pillow in Python to help us do that. Some of the things that were done in the preparation stage were as follows: resizing to 28x28, putting the images on a black canvas, etc… On Latte, we were given 860 images. We have used 90% of the images (774 images) to train our program and the rest 10% (86 images) we have used to test the program.

(2) Symbol separation in different parts

To do symbol separation we have used a python library called OpenCV. The way that it works is via the application of DFS (depth for search) in a 2-dimensional array. In other words, when the code registers a different color pixel from the background (In our case suppose the code recognizes the color white) it will read all the neighbors of the pixels around it. In addition to that, it will record the top left corner and the bottom right corner and return a bounding box. The next problem that we faced is recognizing symbols that contain two parts in it ex: the letter “i”, if we don't have the little dot on the top then we might mistake it for a “1” or an “l”. For that, we used a method called the “Parsing Method”. What this means is before we recognize a symbol like an “i” or “=” we can use the bounding boxes to guess what the symbol is and based on that we can make decisions based on combinations. Here is the process: step 1, guess the symbols based on the bounding boxes. Step 2, make the needed combinations. There are basic symbols that need to be identified before going on to the combination stage like lines, dots, etc… for combinations like: “i”, etc.

(3) Symbol Recognition

For our symbol recognition part, we have trained our program with 5000 iterations. It took a long time to train and we had a very nice accuracy result of 99.7%. When we made fewer iterations ex: 1000, 2000, 3000, etc... The accuracy wasn't as good but if we went over 5000 it took too long to calculate. We were very satisfied with the result thus we continued from there. In addition to that, we have 3 convolutional layers instead of 2.

(4) Equation Reconstruction

After we recognized the symbol we want to build the equation based on the value and position of each segmented symbol from the equation. The way that we examine the creation of the needed equation is by foreseeing the value and the position information of the symbol. Firstly we Understand the individual symbols in context. The way that we understand individual symbols in context is by determining the value of the symbol. We need to convert the symbol into which it should be by analyzing the context.
Secondly, we need to analyze the connection between symbols and decide the expression of the equation. My group and I implemented 3 types of working out the relative position relationship. The first type is the Square, to deal with square roots we need to use our toLatex method. Basically what we do here is divide the equation into 2 parts: the a part

and the b part. We do that because we want to determine the list of symbols that belong to the square root. We use the toLatex function to return recursively a completed equation. The second type is the Fraction, to recognize a fraction we need to understand the symbols on the top of the fraction and the symbols at the bottom of the fraction. We can also break it into 2 parts. The first part which we call a will represent all the symbols on the top of the fraction and the second part, part b will represent all the symbols at the bottom of the fraction. Here we of course also use the toLatex function recursively to output the equation out. The third type is the Superscript, here we are trying to recognize an equation like x^2. We can break it into 2 parts as well part a and part b. Suppose part a in our example will be the x and part b will be the 2 . If we see that the centers of a are higher than the center of b then we know that they are not on the same level thus the b is elevated.