# MyPhotomath

Kristijan Palić
kristijan.palic.96@gmail.com

This document serves as the report for the *Machine Learning Intern* selection process task. I will more thoroughly describe how I implemented some of the subtasks, why it works/doesn't work that well, and how would I improve it if I had been given more time/resources.

## 1. Implement a handwritten character detector

As it was stated in the task description, I used the OpenCV library. For starters, I didn't bother with extracting the expression from the noisy data, I'm expecting that the user will input the image that is already cropped and with a white background. I think that extracting the unnecessary things from the image would take a huge effort for something that is not a crucial part of this task.

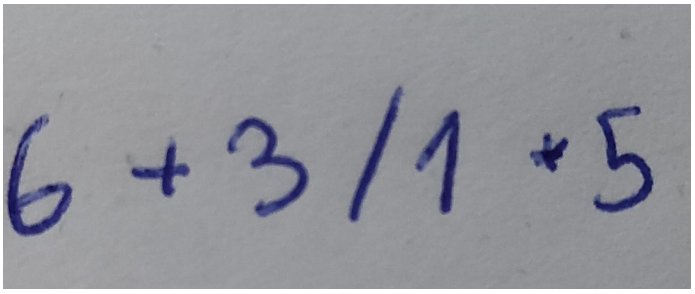

*Figure 1: Example of an input*

After the image is loaded, I applied the OpenCV's *threshold* function to turn on/of the pixels based on its value. After I had a black-and-white image from the original, I applied *findContour* function. It took me a while to extract only the contours I need in order to crop only the characters, without unnecessary data. And this I would stress as the first major problem I didn't succeed to solve. My solution was:

1. Remove all the contours that don't have the contour with id 0 as a parent. That way, I removed all the contours that were part of the bigger one.
2. Remove all contours that have a width less than 10. I assumed that those are the ones we don't want because it's the noise in the data.

It works very well for the very pretty input, but as soon as it's not that nice, it fails for a bit.

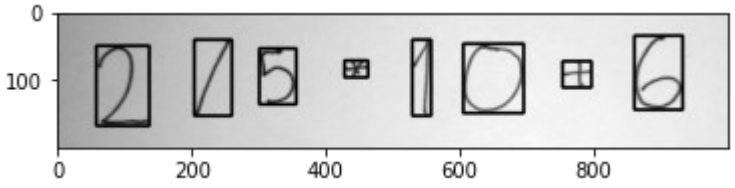


*Figure 2: Example of a successfull character wrapping*

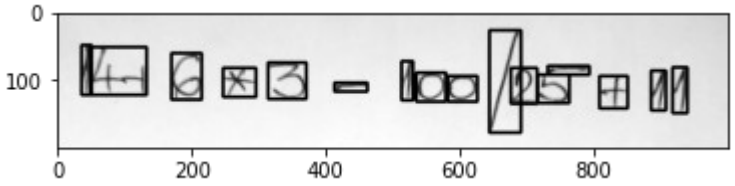


*Figure 3: Example of an unsuccessfull character wrapping*

After the characters are recognized, they are sorted by the x axis, cropped and moved to a folder *recognized_data/** as an input for the next step.

# 2. Implement a handwritten character classifier

After extracting each character from the image I had to make a classifier that will map each character with some label. I prepared the train and test data in a way that filename was like '*characterlabel_exampleNumber*' where character labels are (e.g. 9_1, 15_3):

- 0-9 for numbers
- 10-15 for ( ) * / + -

The first problem I occurred was that not all the extracted characters were of the same length. My first approach was just to put them to a larger frame, and fill the unused pixels with the white pixels. But that would result in sparse data. What I thought would be a better approach is to scale every character to some specified frame, in my case, it was a box of 32x32 pixels.

After the preprocessing was done, it was time to put them into the neural network and play with the parameters to get the best result. I decided to go with the simplest possible NN, with only one hidden layer. With my dataset that consists of around 7 examples for each character, on the training set I achieved an accuracy score of 95%, but could barely hit the 45% accuracy on the test set. I blame it on the really small dataset for this type of problem and on a really simple neural network.

# 3. Implement a solver

After I received a string expression from the classifier, I had to implement a solver. I decided to use a Shunting-yard algorithm[1] that transforms the expression to a postfix notation[2] that follows the importance of the operators and works with paratheses.

# 4. Bonus points

Although I do have experience with both Docker and creating web apps, unfortunately, I didn't have the time to implement those two things if I wanted to send the task solution by the deadline.

1 https://en.wikipedia.org/wiki/Shunting-yard_algorithm
2 https://en.wikipedia.org/wiki/Reverse_Polish_notation

# 5. Conclusion

After finishing the task I was slightly disappointed with the results, since the model didn’t work very well for even the simplest examples. But at the end of the day, I don’t really think it matters how it performs, because I think what is more important is to comment on how I built the model, address the biggest problems, and to show that I know why it doesn’t perform that well.

The biggest weakness of this kind of approach is eliminating the noise in the received images. Because when you have correct data, from that point on, it’s really straight-forward. You collect a huge amount of data, build a complex NN and wait for the results. So when the more complex examples has to be solved, not only including a few operators like here, I think that extracting data from the image is the hardest problem.