Automatic score calculator for Mathable

1st Semester of 2024-2025

Popescu Andrei-George
georgepopescu318@gmail.com

Abstract

To start: Mathable is a board game similar to Scrabble, in which each player places pieces with numbers to make different mathematical equations with the other pieces already on the board to receive points. The code I will detail below can resolve three different tasks that are necessary to calculate the score for each player: determine the position of the newly placed piece, find the number found on the new piece, and finally calculate the score.

1 Introduction

Key aspects of the project:

- **Input**
  The input for this project was represented by images with moves in the game, each move indicates a new piece placed on the board, and a game is composed of fifty moves. Alongside each game, a turns file is found, a turn is a succession of moves that one of the two players performs consecutively, this is used to determine the score the player has at the end of his turn.
- **Output**
  The output is very much related to the input, meaning that for each image with a move, a text file describing the position and the number found on the piece found in the image has to be created. At the end of the game, another text file is created, with the score each player received after each of their turns.
- **The code**
  I wanted to structure the code in a way that makes it easy to test different approaches and easily modify different functions of the code. Many different Traditional Computer Vision Techniques are used throughout the project to solve the tasks, I will get into detail in the following fragment.

2 Approach

- I will describe below how I solved the tasks, the technical methods, and the optimizations used to get better results. Parts of the explication will be regarding a singular image and others will be about an entire game or set of images.
- **Source Code**

1. **Finding the piece**
   - In the original format, a big part of the image was represented by the table on which the board is placed, by using the **HSV** color space, I selected two values in the HSV space and performed color thresholding using *cv.inRange*, simply put the pixels outside the range are turned to zero.
   - Now that the table was black I still needed a way to extract only the game board, I needed to find its contour and extract it from the bigger image. I performed some **preprocessing** before I could search for contours, **erode** is a morphological operation that made the contour clearer, so I used *cv.erode*. Next, I used *cv.Canny* to get the edges in the image so that *cv.findContours* could identify the board and with *cv.warpPerspective* I transformed the contour found into a square so that I could better delimitate each tile on the board.
   - Lastly I figured that the best way of finding the position of the newly placed piece was to subtract one image from another, the next step was to search for the tile with the biggest amount of color, this search was made in a grid a determined by knowing the distance between the

blue border of the board and the position of the actual tiles.

2. **Identifying the number**

   - Now that the location of the new piece is detected, I performed template matching with a set of templates I extracted from the images. Template matching is done by sliding the template on the patch I extracted earlier, *cv.matchTemplate* does all that for you.
   - Just template matching with all the available templates didn't result in good results, so I wanted to find a way to improve it. The solution I came up with was to determine all the numbers that could be placed there and perform template matching with only those numbers.

3. **Calculating the score**

   - All that was left was to calculate the score based on the value of the piece and the modifiers from the special tiles or the number of equations that result in that number, I was able to determine this by using a matrix that contains all the pieces placed on the board up to that move.

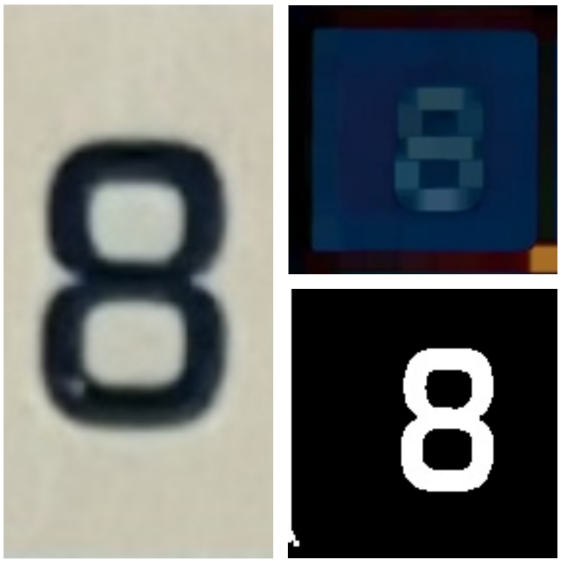

Figure 1: Original, Difference between two images and Preprocessed for Template Matching

3 Conclusion

- **Results**
  I was able to get 100% accuracy on the training and validation datasets, even though at first I had a few setbacks and not everything worked right, but doing template matching with only possible numbers made the program more robust.

- **My experience**
  I was able to make a code that worked with just classical computer vision techniques, and that didn't take too much time to run. I enjoyed working with images and experimenting with different techniques in order to make the program as good as I could.