Documentatie Tema1

Oproiu Matei

December 2024

1 Introduction

1.1 Objective

The goal of this project was to process and analyze "Mathable" game boards to discover changes and calculate scores between sequential game states. We used computer vision techniques and image processing to automate tasks that would otherwise require manual input.

1.2 Overview

The code works by taking as input fixed photos of the game board and extracting only the playing area, which it then compared to the prior state to determine where the new piece was introduced. After that, it determines which piece was placed on the board and calculates the score. We will go into greater detail in Chapter 3.

2 Preliminary Concepts

2.1 Libraries and tools

- **Python**: The programming language used for this project.
- **OpenCV**: Used for image processing tasks.
- **NumPy**: Used for numerical operations and handling of matrices.
- **Matplotlib**: Used to visualize images and intermediate results during debugging.
- **os Module**: Makes file system interactions easier, such as creating directories and accessing file paths.

2.2 Game board representation

We will refer to $m_{original}$ as the original game board that will not change and to m the game board that will change each turn, both of them are implemented as matrices in our code. We will consider *pieces* to be the list of all possible pieces in the game, *possible_moves* a list where we will store all the possible moves for each state of the game, and *visited* the list of the starting squares which we will update anytime a new piece is added to the board.

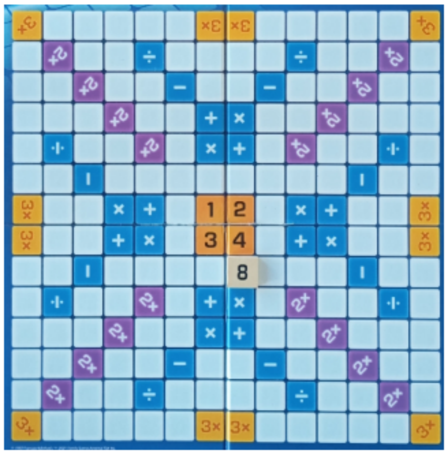
3 Algorithm Steps

We will explaining step by step the processes we go through to obtain our results.

3.1 Extrating the game board

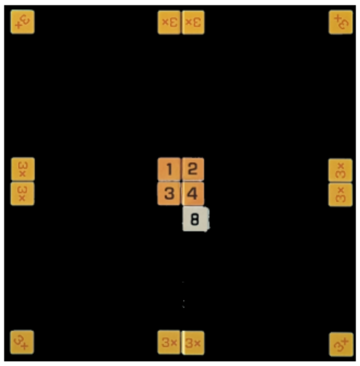
Knowing that the images are taken from a fixed position we can approximate manually a boundary for the game board leaving space for noise. We can then crop the image around this boundary to make the following processes easier.

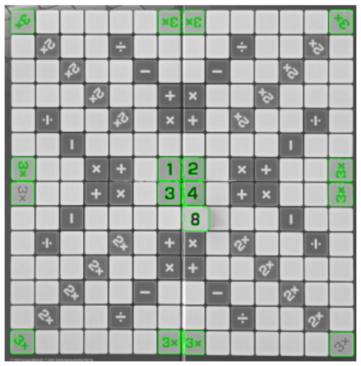
(a) Input image.

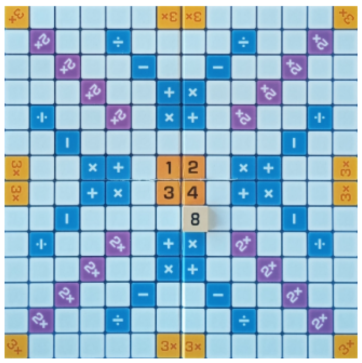

(b) Cropped image.

We observe that by taking our BGR image and converting it to HSV and applying a upper bound value of around 85 to the hue we can create a mask and apply it to our image to make everything black except for the orange like colors in the image. This makes the process of finding the corners of the image easier because it removes everything around them. We then apply the following image processing techniques on the grayscale masked image: median blur, Gaussian blur, image sharpening, threshold, erosion to find the edges of the image using Canny edge detection. After that we use the build-in function in OpenCV to find the contours.

- Median Blur: ksize=5
- Gaussian Blur: ksize=(3,3), sigma=0
- Image Sharpening: α=1.2, β=-0.8
- Threshold: thresh=50, maxval=255, type=cv.THRESH_BINARY
- Erosion: ksize=(3,3)
- Canny: treshold1=200, thresold2=400
- Find Contours: mode=cv.RETR_EXTERNAL, method=cv.CHAIN_APPROX_SIMPLE


(a) Masked image.

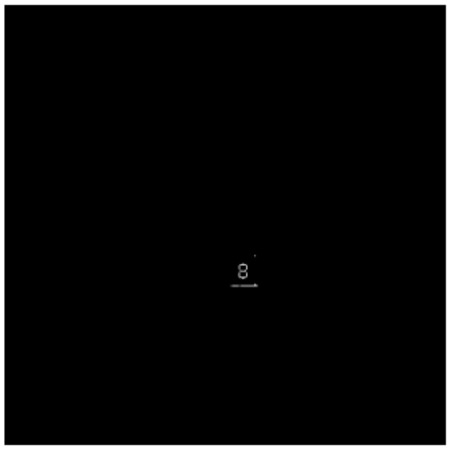

(b) The contours found.

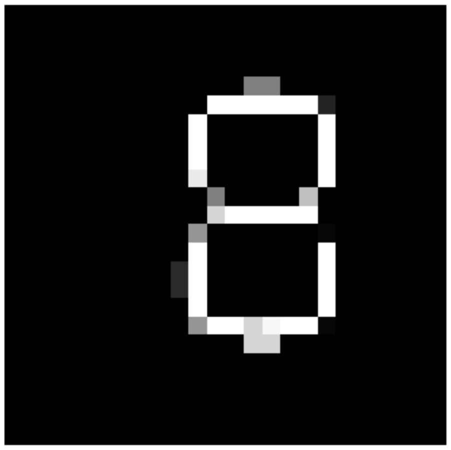

(c) Mapped game board.

After we have extracted the contours we can implement an algorithm that iterates through the contours and finds the corners of the game board. We can then apply a perspective transformation to map the game board to a 1400x1400 image.

3.2 Finding the following tile's position

To find the position of the new tile on the board, we compute the absolute difference between the current image and the previous image. After we have done that we apply a threshold with thresh=85, maxval=255 and type=cv.THRESH_BINARY. To ensure that we have minimal noise in our image we also apply a erosion with a kernel size of (5,5).


(a) Absolute difference image.


(b) Highest mean value patch.

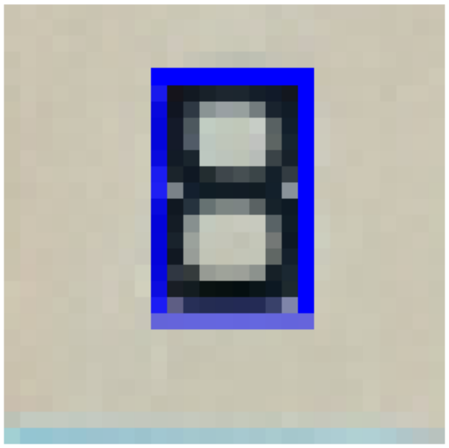
Knowing that our game board image is 1400x1400 we separate the image in patches of size 100x100 and we check only the patches in the *possible_moves* list to find the patch that has the highest mean value, this gives us the coordinates of where the new tile was put. After we find the coordinates of the new tile we add to the *possible_moves* list the neighbors of the tile.

3.3 Identifying the number

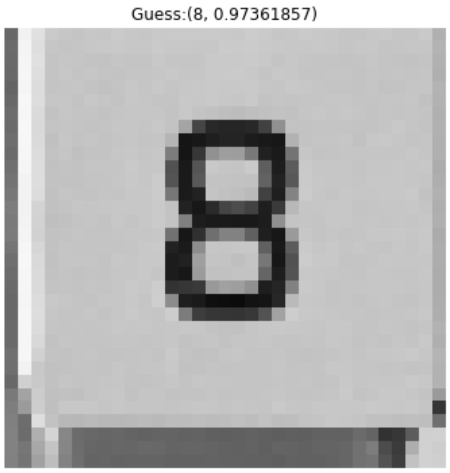
Before trying to find out what number is on the tile that was put on the board we will do two things to help with the process.

3.3.1 Differentiating between one and two digit numbers

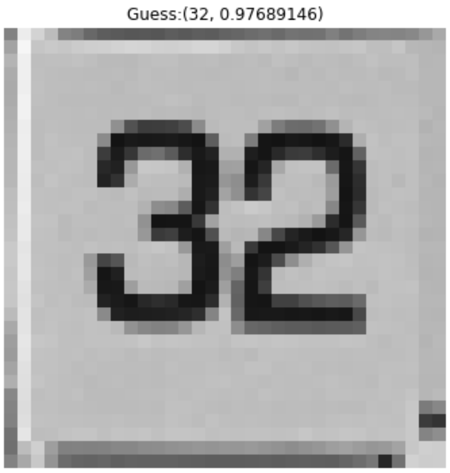
One of the fist challenges of identifying what number is on the new tile was differentiating between one and two digit numbers. To do that we take the patch where we know the piece was added with a margin of 10 pixels, we apply a threshold with thresh=120, maxval=255 and type=cv.THRESH_BINARY. We find the contours and to ensure there are no bad contours we add the conditions that the width and height of the contours need to be between 10 and 70 and the area of the contour needs to be between 200 and 2000.


(a) One digit patch.

(b) Two digit patch.



(a) Example template matching.



(b) Example template matching.

3.3.2 Finding the possible pieces

At this step calculate the *possible_pieces* list by checking what type of tile is the position where the new piece was added using the matrix *m_original*, then checking whenever the two neighbors in all directions are in the visited list and whether they can sum, subtract, multiply or divide between themselves depending on the type of tile, using the *visited* list and the m matrix.

3.3.3 Template matching

After we have the possible_pieces list and we know how many digits the number has, we iterate through a list of all the possible pieces that we initialize at the beginning and never change and we select only those that have the corresponding number of digits and that are in the possible_pieces list. We template match our patch with all the templates of the corresponding numbers we have found to have met our criteria and save the number that has the highest template matching value by modifying the corresponding position in the m matrix.

3.4 Calculating the score

Once we know what number we have in our new tile we can calculate the score by first checking what type of tile was the piece put on in the m matrix to ensure we check only the required mathematical operations. We check if the two direct neighbors of our tile are in the *visited* list and if they are and the corresponding mathematical operations match our number we add our number to the score, this ensures that even in the case where our new piece completes multiple equations we get points from all of them. Here we also need to be careful in case we are on a 2x or 3x multiplier score to multiply our number by that before adding it to the score.

4 Algorithm and Game Logic

Now that all of the algorithm steps have been explained we will bring everything together and explain how to program works.
First we are going to initialize everything we need: the lists and matrices we talked about before as well as a game, move and turn index and read the first image that will be the empty board. We find the current player by reading the first word of the first line of the turns.txt file and we create a list player_turns that will hold the amount of moves each player makes before their turn is over. We iterate through all the files in the input_folder that end with .jpg applying the steps we talked about in chapter 3:

1. Extract the game board
2. Find the position of the new tile
3. Identify the number on new tile
4. Write the tile position and number in a text file
5. Calculate the score

To implement the game logic we add the score to the current player and add one to the turn index. When the turn index is equal to the amount of moves that player has made or the player_turns list is empty add the score one last time and then change the current player. Each time we change the current player we add their score to a list. After doing all of this we increase the move index by one.
Once the move index is equal to 50 we set it back to 0, we re-initialize everything like we did in the beginning, we increase game_index by one and we output a text file similar to the turns.txt but where we add at the end of each line what score that player has accumulated each turn for this game.
This process will repeat for all of the files and output text files corresponding to the position and number of each new tile and the scores the players have accumulated throughout the games.