Mathable Scoring Application

Project Report

Ocnaru Mihai-Octavian

1. Introduction

The scope of this project is to be able to compute score for a Mathable game. For the simplicity the game is only played by two players and the number of rounds has been restricted to 50 per game.

For more information about the rules or any other technical aspect of the game please consult the following resource.

1.1 Objectives

In order to be able to determine the score for each turn we will have to:

1. determine the position of the played piece on the board
2. determine its value
3. calculate the score for the respective round
4. sum up all the scores that represents the rounds for the player's turn.

The 4) subpoint is already provided to us in a file which clearly separates each turn. Its format have the following structure:

```
player name _ the starting round index
```

The "player name" can either be "Player 1" or "Player 2", the "starting round index" refers to the round index at which the player starts its turn, for example "player 1" will always start at round 1, as an internal convention.

For the first 3 subpoints we will explain in details how they were caomputed in the following lines.

2. Implementation

2.1 Computing the position of the played piece

For this approach we will have to iterate over two important steps. The first one refers to the cropping of the board. The images provided as training and evaluation data were both 3072×4080 pixels, which are very big for an image, but also contains a lot of background content that is not relevant for our evaluation.

Fig1. An empty board

What we will help us will be to obtain only the playable region of the board, which contains of a 14x14 grid. For the rows we will use arabic numbers and for the columns notation we will use latin capitalized letters. So the first piece from teh board, the 3x orange multiplayer, will have the coordinated (0, 0) which translates to "1A" position.

In order to be able to obtain that, we first have to eliminate the background. If we look closely we can see that the predominant color is a orange-brown pigment. The board background color is also an intense blue, so by using a BGR filter like **[0, 39, 149]** we can eliminate the background for the image and only have the main content.

After this, we can convert the image to gray scale and apply a Gaussian blur, preparing the image for an edge detection algorithm. We apply a Canny edge detection, with the following paramters and values:

- lower threshold: 60
- upper treshold: 80
- Sobel kernel shape: 3x3.

The resulted image may result in minor noise and other imperfection that we may want to correct, so a closing morphological operation. We use a 5x5 kernel as input and apply the dilation operation 2 times, followed by a one time erosion.

We proced by applying a finding contour algorithm to the processed image. This may find the column with the number of pieces, spotted in the right of the board and maybe other imperfrction such as "Mathable" logo from the board.

For the proper board extraction we will use that the images provided are taken from a fixed angle and the board is moving slightly, almost not at all. After measuring the board we can approximate its length to be 1470, which is also a multiple of 14(105 pixels for a piece). So knowing this, we can extract the largest contour from the image, which is the board, maybe with logo glued to it, and find its centroid. Having the centroid will allow us to draw a square, with the center of the centroid, the length of a side 1470, determine thsi way its coordinates.

With the board coordinates provided we can simply extract the board from the main image. The whole process explained ealrier can be visualised below:

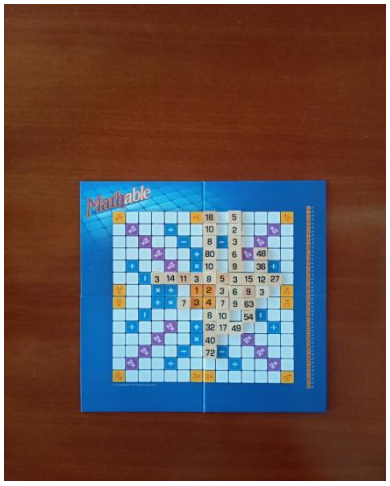


Fig2. Original Image

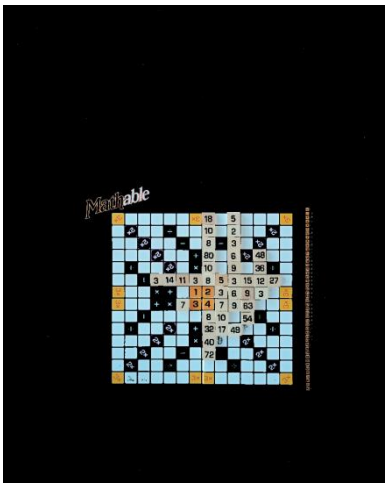


Fig3. Filtered Image

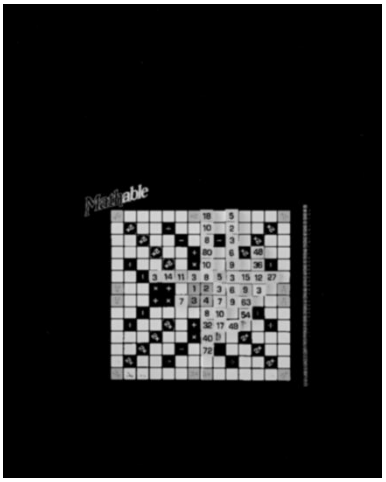


Fig4. Blurred Image

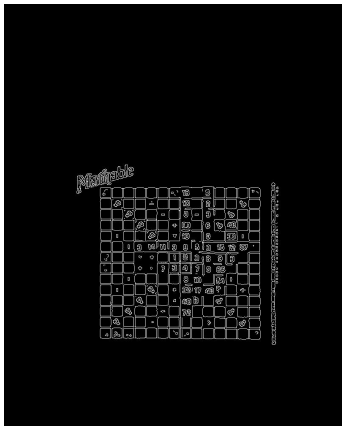


Fig5. Edges Image

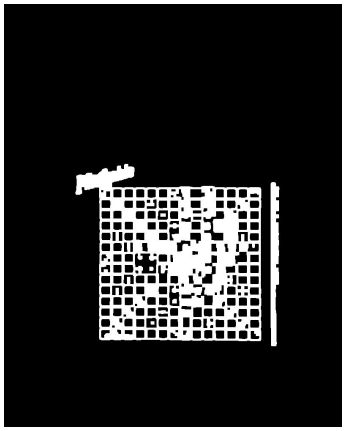


Fig6. Postprocessed Image

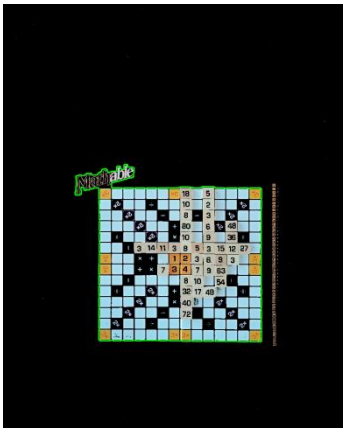


Fig7. Contoured Image

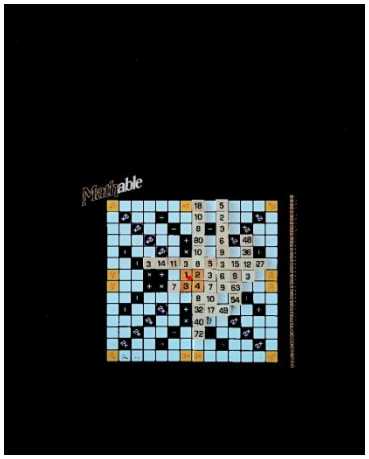


Fig8. Centroid Image

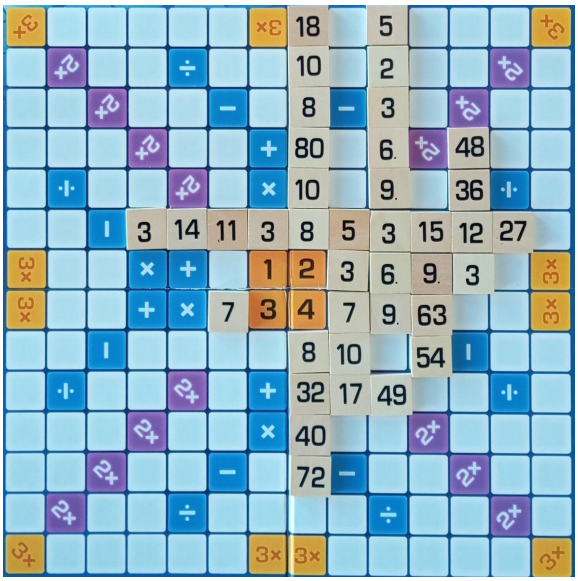


Fig9. Board of the image 1_38.jpg

Now that we have a way for extracting the main content of the image, we will focus on how to find the played piece. Suppose that we are in the round 38 of the first game provided as training data.

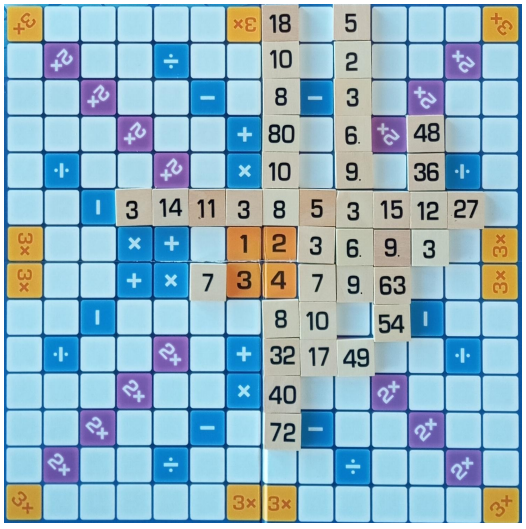


Fig10. Board of the image 1_38.jpg

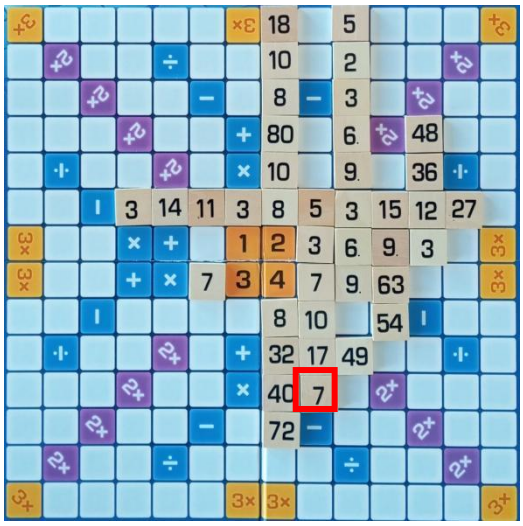


Fig11. Board of the image 1_39.jpg

We can clearly see that the piece played during the 39th round was a 7 placed on the "11I" position, which will translates to (10, 8) coordinates for our board.

We will have to determine both the position and the value of the played piece, so let's first begin with the easier one, its position. For this part we just add a Gaussian blur to both images, then convert them to gray scale and apply an absolute difference function. The result may contains noise, so we will threshold them and apply a secvention of morphological operations, an open followed by a close operation.

This process is based on the fact that the photos are taken from a stady point, so the variations between one image to another is little. Also this process assumes that there is no variations in the cropping of the board, which is also true, based on the algorithm explained above.

Below are illustrated the steps from above for a better illustration:

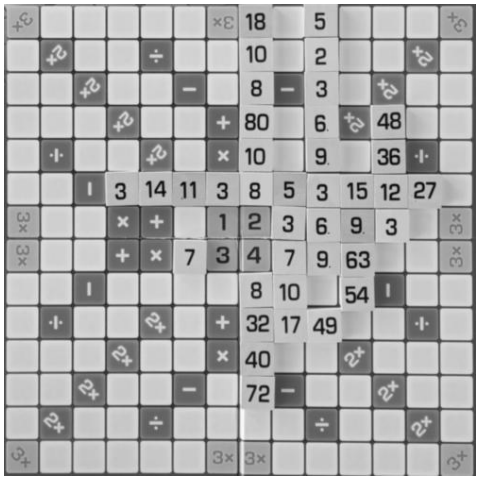


Fig12. Board 1_38 blurred

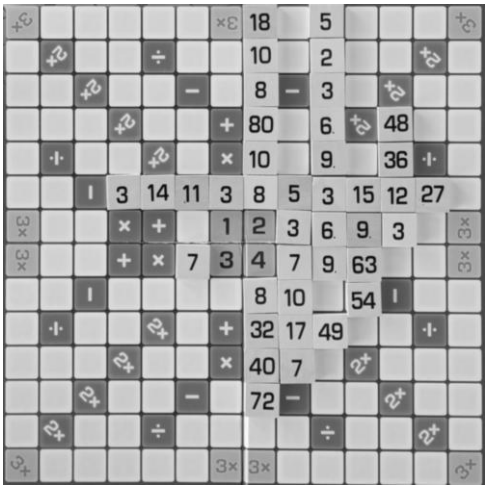


Fig13. Board 1_39 blurred

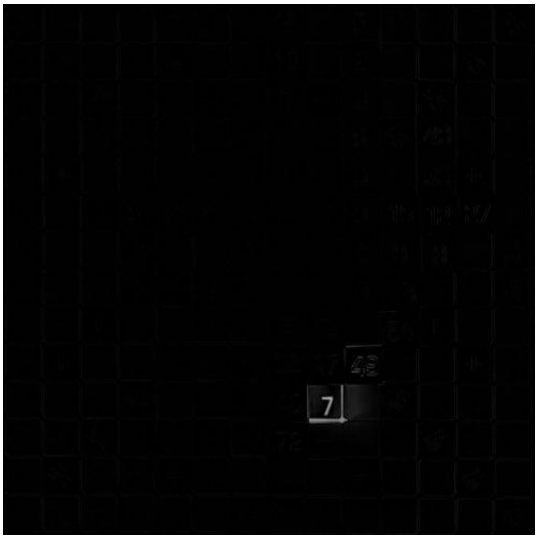


Fig14. Difference between boards

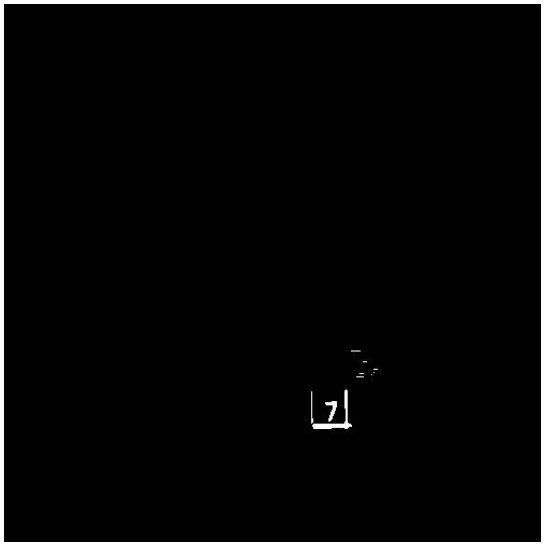


Fig15. Difference thresholded

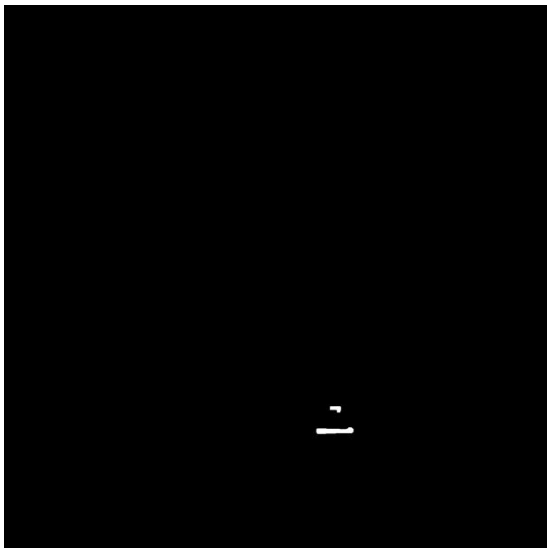


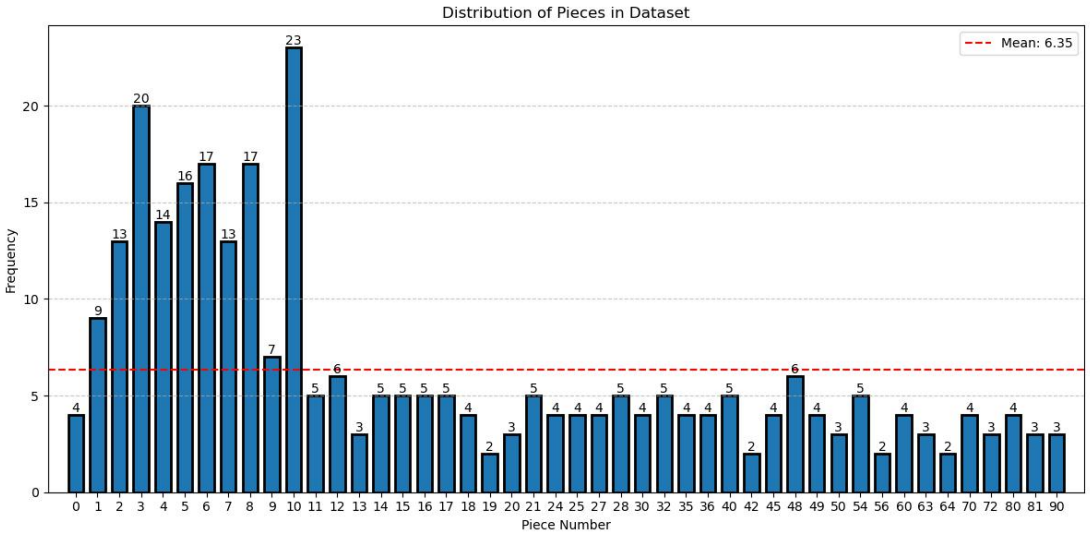
Fig16. Applied morphological operations

After this we will use a sliding window techinique to find the coordinates for the placed piece. Knowing that we choose the dimesnions of the board to be divisible by 14, we can split the board in a 14x14 grid and find the block with the heightest mean.

Once we have found the coordinates of the piece, we can return to our board and crop the piece from there. The hardest part come now with determining the piece value. For this part we will use a convolutional neural network.

2.2 Computing the position of the played piece

For training a CNN we will need a consistent dataset, which is not provided for us. We will augment our data extracted from the provided training images and auxiliary images. The below plot shows the distribution of the images for the training and auxiliary images:



As we can see, the distribution of the data is not uniform, which may seems unusual at a first glance, but this is due to the mathematics properties of the number. For example, the number 42 is only used 2 times, in comparison to the number 10 used 23 times, this can be exaplained as the number 10 appears in the game pieces 10 times, rather than only one piece.

Keeping this distribution may produce a negative effect in the training of the image, as some classes may have no representation in the weights of the model. That's why we will set a target which will represent how many images we use for representing each class.

The augmentation of the images will be made by changing the brightness, saturation and zooming in the image, providing a widely range of a result images.

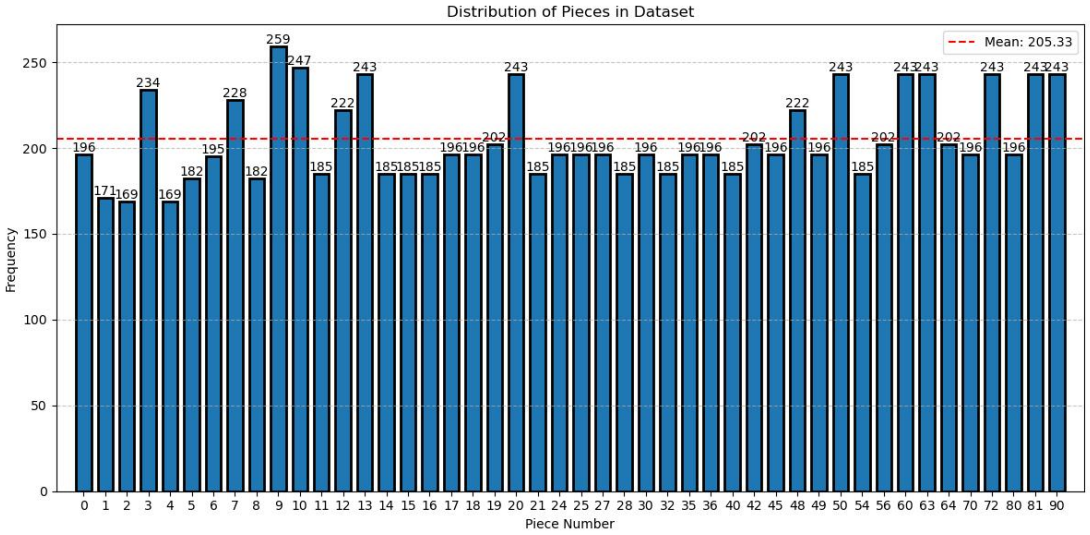
The way in which we are determining the number of stops for brightness and the other paramters is by first setting the desired target, we will use 1000 examples for a class. Then we will start with teh value 1 for each parameter and keep increasing them uniformely until the product of the parameters is above the desired target. Once this happens we will check if we are in a 20% upper limit regarding the target, if we are not, we will start by first decreasing the number of zoom steps and if we are still above, we will decrease the number of saturation steps.

As an example, suppose that we have the piece 2 with a 20 piece for representation. Following the table we can see how the increment is made, by saturation we will refer to number of steps for saturation, this is applying for all:

saturation	brightness	zoom	calculation	result
1	1	1	20 * (1 + 1*1*1)	40
2	2	2	20 * (1 + 2*2*2)	180
3	3	3	20 * (1 + 3*3*3)	560
4	4	4	20 * (1 + 4*4*4)	1300
4	4	3	20 * (1 + 4*4*3)	980

The +1 from the calculation formula comes from the base image that we are augmented. Once the target is achived, we will stop and compare the result with the 1.2 * target to see if we are above. Because the result is above this threshold we will first decrement the zoom steps. Doing so will result in a closer result to the desired target so we will stop.

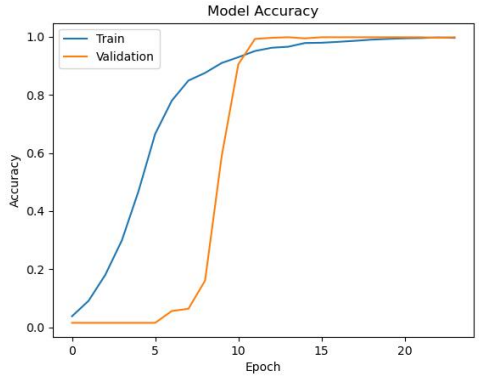
After performing this, the distribution of the data will look like this:

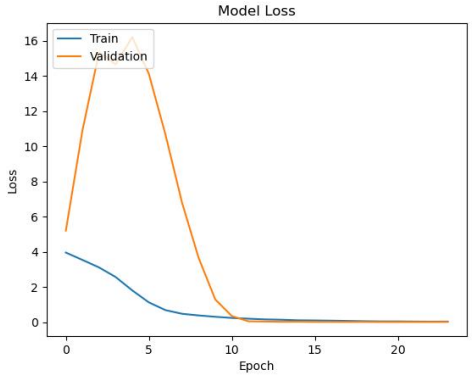


We can clearly see now that the distribution is an uniform one, which will guarantees us the correctness of the model.

For the model, the architecture is provided in the code and we used an early stopping mechanism for preventing overfitting, a dynamic learning rate for a better adaptation and a batch size of 128. The distribution for the training and validation data was 80%-20%.

The model was then tested on the fake test provided and it achived 100% prediction accuracy as long as a 100% prediction accuracy of the training data.





2.3 Calculating the score for current round

First we need a way to represent our board, we will go with a matrix approach, in which we instantiate the 4 pieces from the center with their values. We will also keep track of the current player, their score, the starting position and current turn.

After placing a piece we receive its coordinates and value. With this we will first have to verify for any equation that it may fulfill. This is done looking in all orthogonal direction and check if we have 2 otehr pieces that we can form an equation with. Besides checking for a valid equation will we also have to check for any defined constrained. For example we may have two pieces 2 and 4 and the played piece 6 and a constraint of multiplication, in that case we can not score this equation as 6 is obtained via addition.

For every equation found above we will sum up the value of the piece and its sum we will multiply it with a bonus value, if the played piece was placed on a tile with a multiplier.

When the current turn ends, we will switch the players, reset the score and update the starting position for the current player, along with saving the last player progress.

We do this for every round, until the game is done, when we dump all the players progress in a file and end the predictions for the current game.