2D Chessboard Recognition

Marian Sandor
Technical University of
Cluj-Napoca
Email: Sandor.Da.Marian@utcluj.didatec.ro

Cluj-Napoca
Romania
Email: marianshandor@gmail.com

***Abstract*—In this paper, I aim to correctly detect and identify a chessboard along with its configuration of pieces through application of image processing.**

I. Introduction

The purpose of the project is to create an application that is able to detect and correctly identify chessboard configurations. The input will be images focusing on a chessboard along with a specific configuration of the pieces. The application will process the image and recreate the configuration in memory and display it as a matrix as it can be seen in the picture below (Figure 1). This is not at all a new concept in the chess industry as it is commonly used in tracking live games in competitions. Taking a step further, reading and interpreting the chessboard can be of great benefit for chess-playing robots giving them the ability of "seeing".

II. Related Work

A good example of such an application was developed at MIT by Emil Osterhed [1]. In this case, the application goal is to recognize the chessboard and build the characters string according to the FEN encoding used as standard to describe the pieces distribution inside the chessboard. The images with which the application works are from the magazine "La settimana Enigmistica". An example of its input-output flow is shown in Figure 2. The approach embraced here follows some predefined stages. First is the chessboard segmentation in which the actual chessboard is identified in the given image. Next a projection transformation is performed to rectify the image and crop it around the checkboard in order to prepare it for the separation of the 64 cells. Having the squares of the board, the next step is to use a kNN classifier to recognize the cells (Bishop, Empty Square, King, Queen, Rook, Knight, Pawn) and then find the pieces' colors and orientations. Lastly, the FEN string is constructed according to the standard to describe the pieces distribution inside the chessboard. Even though there are some constraints such as all the images should be taken from the magazine "La settimana Enigmistica" and the image shouldn't be much blured, the accuracy is quite high (from 64 images in 57 of the cases the output FEN was correct).

III. Method

Before getting into the actual implementation, it is worth mentioning that the application will be working with images generated from https://www.chess.com/analysis. Even though the pieces style as well as the board colors can be changed, all the images will have the same size. The images are 720 by 720 pixels meaning that each of the 64 squares of the board will be 90 by 90 pixels. Knowing this and also the fact that the white square is always in the top left corner of the board will be be of great use.
Since we want the application to be as independent as possible of the board color and pices style in a way that changing the pieces style or board color from the website when generating the image should not influence the performance of our application. In order to achieve this, the application should work based on a model which will be generated from a training set containing specific images of the pieces and the board. The training set will consist of exactly 13 images (one for each piece and an image with the empty board) with a format as seen in Figure 3. The output model will contain 12 90 by 90 images each of a specific piece (Figure 4) and a text file storing the average rgb color of the white and black squares along with the darkests greyscale value of them.

Next the training process is described. The application is given the path to the folder containing the training images and will start by analyzing the image with the empty board first. It will compute the average color for the white and black squares as well as the darkests greyscale value for them which will then act as thresholds. These values are stored

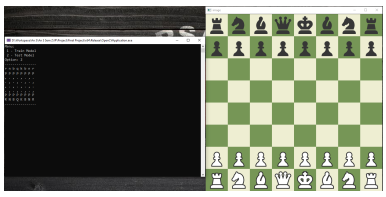

Fig. 1. Expected input-output flow

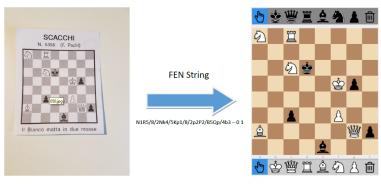

Fig. 2. Expected input-output flow

in a file and the next stage is to create de image models of the pieces. The first step is to crop the square containing the piece into a 90 by 90 image. This image is then converted to greyscale and a binary threshold is applied, the threshold being the darkest greyscale for white squares previously computed. As each of the pieces (black or white) has a black outline contour, the resulting image will have a white background with the contour remaining black and the inside a combination of white and black. What matters the most is the contour, because Knowing it will always be black we can now start a bfs from the top left pixel and mark all the background pixels. The last step is to set all the white pixels to black as they can only be in the inside of the piece. At this point, the image has only two colors, grey for the background and the shape of the piece is black. This image is then used to filter the initial piece image such that the background of the piece is set to a predefined color (green as seen in Figure 4).

Now we'll focus on the testing process which is the main functionality of the application. The user selects an image of a configuration and the application will output its interpretation in the console. The program stores the representation of the board in memory as a matrix of chars where pieces names are given by their notation (https://www.ichess.net/blog/chess-notation/) and black and white empty squares are denoted as '.' and ',' respectively. The identification of the configuration is done by parsing the given image square by square where a square is a 90 by 90 image. Each of the 64 squares is passed to a matching function which will return the class type with the best matching score. The class type is one of the bishopB, bishopW, pawnB, pawnW, rookB, rookW, knightB, knigthW, queenB, queenW, kingB, kingW or in case no piece is detected in the square then it is marked as an empty square and its color is deduced. Because all the magic happens in the matching function we'll take a closer look at what it is doing. First step is to identify if the square contains or not a piece. The square image is converted to greyscale and then a binary threshold is applied to it with the threshold value being the darkest greyscale value for black squares computed in the training process. This will result in a full white image if there is no piece in it else the piece black contour will be present. To answer the question if the square is empty we simply just check if the image contains only white pixels. If the image does contain a piece, the next step is to identify its class type. Each of the model images is checked against the square ignoring the pixels matching the background color of the model images. The pixels of the piece in the model images are compared to the pixels from the square with a given margin of error to accomodate for possible noise and if they match a counter is incremented. For each model image the number of matched pixels is computed and the one that matches the most, decides the class of the square.

IV. Evaluation and Results

For most of the cases the algorithm has no problem generating the model as it is expected. The algorithm struggles a bit for more complex styles of the pieces to detect the exact shape of it (Figure 5) but nonetheless it gets close enough to not influence the matching. The application was tested on three different models having different pieces styles (from simpler to more complex) as well as different colors for the squares. The algorithm managed to correctly identify correctly all the configurations from the given test images which makes it reliable. It is important to mention that a non-uniform color for the squares will badly affect the accuracy of the generated model as the algorithm is very dependent the colors to be relatively disjoint.

V. Conclusion

In the end, the application turns out to work just fine, despite its limitations, managing to correctly recognize the chessboard configuration from 2D images most of the time. The algorithm for extracting the model is what I'm proud the most along with the matching strategy for classifying the squares of the board. The fact that at the moment it works with images of a very specific format is clearly something that can be worked on. As a first step, improvments can be made such that the input images are photos of 2D characters and the actual identification of the board should be also performed. And of course, taking a step further and dealing with real world

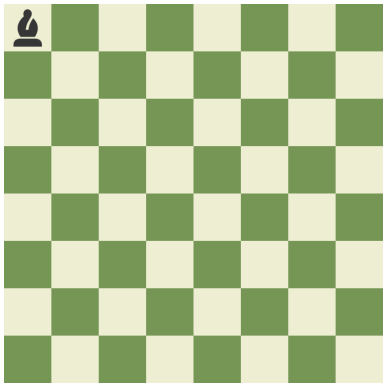

Fig. 3. Train Image

Fig. 4. Model Image

situations, we could consider images with 3D chessboards which complicate things way more, but achieving this would be of great interest for the chess industry.

REFERENCES

[1] Emil Osterhed. Detect-Chessboard-FEN. https://github.com/Automatik/Detect-Chessboard-FEN, 2018.

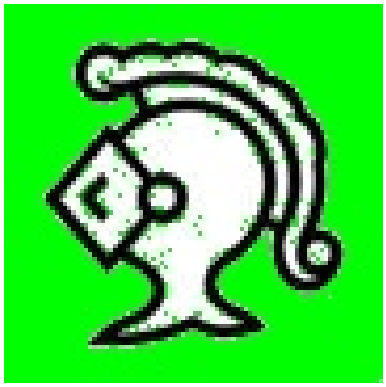

Fig. 5. Model Image Complex Piece Style