FOUNDATIONS OF COMPUTING AND DECISION SCIENCES
Vol. 45 (2020) No. 4
ISSN 0867-6356
sciendo DOI: 10.2478/fcds-2020-0014 e-ISSN 2300-3405

Chessboard and Chess Piece Recognition With the Support of Neural Networks

Maciej A. Czyzewski[*], Artur Laskowski[*, †, ✉], Szymon Wasik [*,†]

Abstract. Chessboard and chess piece recognition is a computer vision problem that has not yet been efficiently solved. Digitization of a chess game state from a picture of a chessboard is a task typically performed by humans or with the aid of specialized chessboards and pieces. However, those solutions are neither easy nor convenient. To solve this problem, we propose a novel algorithm for digitizing chessboard configurations.

We designed a method of chessboard recognition and pieces detection that is resistant to lighting conditions and the angle at which images are captured, and works correctly with numerous chessboard styles. Detecting the board and recognizing chess pieces are crucial steps of board state digitization.

The algorithm achieves 95% accuracy (compared to 60% for the best alternative) for positioning the chessboard in an image, and almost 95% for chess pieces recognition. Furthermore, the sub-process of detecting straight lines and finding lattice points performs extraordinarily well, achieving over 99.5% accuracy (compared to the 74% for the best alternative).

Keywords: chessboard detection, chess pieces recognition, pattern recognition, chessboard recognition, chess, neural networks

1. Introduction

It is a natural behavior for people to simplify repetitive tasks because we typically prefer to focus on challenges that require high creativity. Therefore, machines are widely used for such automatable tasks. An example of such a take could be digitizing physical content, which is often very time consuming, does not require specialized knowledge, and is still

[*]Institute of Computing Science, Poznan University of Technology, Piotrowo 2, 60-965 Poznan, Poland
[†]European Center for Bioinformatics and Genomics, Poznan University of Technology, Piotrowo 2, 60-965 Poznan, Poland
✉ artur.laskowski@cs.put.poznan.pl

frequently performed or substantially supported by humans. However, recent developments in computer vision and artificial intelligence algorithms have improved the performance of software programs for performing such tasks and reduced the need for human assistance [42].

In this article, we focus on the problem of detecting chessboards captured in digital images. A chessboard is the type of board that is utilized to play the game of chess. A board consists of 64 squares (eight rows and eight columns) and the chess pieces that are placed on a board and can hide certain portions of that board. The aforementioned application is different from the most common application of the chessboard detection problem, which is typically utilized in the context of camera calibration processes [11]. Detecting chessboards for calibrating cameras is a much easier task because one can assume specific colors of squares, and there are no obstacles placed on the board. Nevertheless, this application is also an important issue for chess games. It can be utilized for loading a chess game state into the memory of an autonomous robot or artificial intelligence (AI) algorithm for playing chess [41, 28, 8]. Such a solution is crucial for many experienced players who wish to compete against AI bots, but prefer to make decisions based on the analysis of a physical chessboard instead of its digital representation. A similar problem occurs when someone wishes to digitize the play in a chess tournament, such as for a live online broadcast. Typically, such digitization tasks are performed by humans or with the aid of specialized chessboard and pieces. However, this is not an easy or convenient solution. To solve this problem, we propose a novel algorithm for digitizing the chessboard states. In this article, when referring to the chessboard detection problem, we consider it as two inseparable steps: detecting the board itself and then detecting and recognizing the types and locations of chess pieces on the board. The schema of such a process is presented on a 1. Only a method that implements both of these steps can fully support chess players.

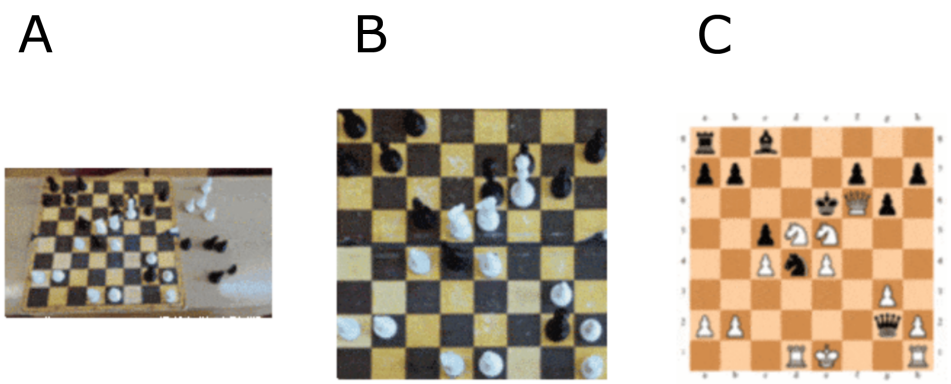

Figure 1. Schema presents the workflow of a full digitization process. The part "A" of a schema is an exemplary picture, part "B" is a picture cropped to enclose only the chessboard, and "C" is a visual representation of the game-state that was extracted from the picture "A".

Furthermore, it is worth noting that there are other computer vision problems that are indirectly connected to the problem of detecting chessboards. For example, the recognition of features characterizing chessboards (lines and grid points) is applied in autonomous vehicles

systems [30], which can identify road situations and make appropriate decisions based on the information they receives via optical sensors [25, 24]. Another significant application that could directly benefit from work on detecting chessboards, particularly the step for recognizing chess pieces, is face recognition [6]. Face recognition algorithms are widely used in various security applications, such as authenticating individuals [23].

The problem of recognizing a chessboard from an image is difficult, with a major issue being the quality of images (e.g., lighting issues and low resolutions for internet streaming). The topic of image quality and lighting conditions was thoroughly studied in the context of face recognition by Braje et al. [5] and Marciniak et al. [27]. Most of the issues discussed in the above papers are also relevant to the chessboard detection process. For this reason, most current methods for a chessboard recognition typically perform certain simplifications. This includes utilizing only a single chessboard style during experiments, capturing images with a direct overhead view of the chessboard or at a convenient angle specified in advanced, utilizing specially designed chessboards, such as boards with special markers that aid in detecting chessboard corners, and other more sophisticated methods for simplifying the detection task (e.g. [10]).

In this paper, we propose a novel algorithm for detecting chessboards and chess pieces that is robust to lighting conditions and image capture angles and performs extraordinarily well on a variety of chessboard styles. The exact colors of board fields are not important, meaning a board can be damaged, or certain obstacles can obscure an image without degrading detection performance. Later in this paper, we present a comparison between our method and other approaches that are currently widely used. Finally, we present a novel enhanced dataset that can be utilized in any future work on chessboard recognition.

It should be noted that many standard algorithms in the computer vision field are not currently able to process many real-world scenarios. They are successful on standard benchmarks but do not succeed in many non-trivial cases. When designing our algorithm, we decided to focus on its applicability in the real world and adapted all methods utilized to handle any conditions. This required us to design our algorithms to be non-parametric, self-adjustable, and accurate. Furthermore, we wished to limit execution time to less than five seconds for a single image to make it possible to utilize the algorithm during chess tournaments to process pictures during live games. For these reasons, we were forced to create replacements for various modules, such as line and lattice point detectors, and enhance them with machine learning techniques. As a result, we created a method that significantly outperforms all existing solutions. In the first few weeks after its publication on GitHub, it has already been noted by many prominent researchers and companies.

2. Related work

Many computer vision researchers have attempted to solve the challenging problem of chessboard detection. As early as 1997, Soh recognized that detecting an enhanced board is a trivial problem [35]. Utilizing markers or any other board-enhancing system improves the ability of algorithms to detect boards. However, such enhancements significantly decrease algorithm versatility. Unfortunately, more generic approaches tend to suffer from environmental effects (e.g., poor lighting). For this reason, there have been several attempts to design dedicated

hardware for supporting chessboard detection. In 2016, Koray et al. proposed a real-time chess game tracking system [22]. This system utilizes a webcam positioned over the chessboard. Other solutions involve the preparation of a special magnetic chessboard and a set of chess pieces [7]. However, such approaches are expensive and difficult to implement. Computer vision systems provide an excellent alternative because they are cheaper and relatively reliable if calibrated properly.

Most computer vision methods work by adapting and combining known and commonly used transformations and detectors. For example, De la Escalera and Armingol proposed a method utilizing Harris and Stephens's corner detection method [19] followed by a Hough transform [13]. The latter step was utilized to enforce linearity constraints and discard responses from the corner detector that did not fall along the chessboard lines because the authors were aware of defects in Harris and Stephens's detector, which is likely to find many corners outside the board [11]. The ChESS algorithm was designed to compete with Harris and Stephens's detector in the field of chessboard recognition [4]. However, this geometric detector can process only simple cases. It encounters difficulty in recognizing lattice points located near obstacles, meaning each chess piece decreases the algorithm's capability for detecting lattice points. The main advantage of ChESS is the speed of processing a single image.

One of the most accurate algorithms (up to 90% of accuracy) was proposed by Danner and Kafafy [10]. However, they simplified the problem by utilizing only a green-red chessboard, which is inapplicable in nearly all real-world scenarios. Many other researchers have proposed corner-based approaches, such as [2] or [47], which averaged the quality of results. Relatively fewer researchers have utilized exclusively line-based methods [35], [21]. The latter method, proposed by Kanchibail, emphasizes one of the weaknesses of these methods, which is sensitive to changes in image orientation. There have also been attempts to design methods that require additional user interactions, such as asking a user to manually select the four corners of the chessboard [12].

Finally, as discussed in the introduction, there are a variety of methods that utilize chessboard detection algorithms for calibrating cameras. One such method and a well-written review of the most commonly used approaches was published by Zhang [46] and later updated by Shortis in the context of underwater cameras [34]. Additionally, Tam et al. introduced the classification of such methods into line-based and corner-based approaches [38]. However, the capabilities of such methods are too limited to be applied to recognizing chessboards with chess pieces on them.

3. Materials and methods

The main objective of our research was to design a method for locating and cropping a chessboard in an image, which could then be utilized to create a digital record of chess pieces positions utilizing Forsyth-Edwards notation (FEN) [14], which is the most commonly used notation for representing states of chess games.

Precise chessboard positioning within images is a computer vision problem of extraordinary difficulty. It requires accurate locating of the four edges of a chessboard. It is a very computationally complex process to precisely locate these edges in a source image in a single

step. For this reason, our solution is based on iterative heat map generation. The generated heat map visualizes the probability that the chessboard is located in a specific sub-area of an image. After generating the heat map matrix, we assume that the chessboard is located in the tetragonal sub-area containing the highest probability values. We then crop this sub-area and correct its perspective to create a square image and iteratively repeat this procedure until the solutions converge. Hereafter, a single iteration, as described above, will be referred to by the term *stage*. Several example outputs of the consecutive stages are presented in Figure 2.

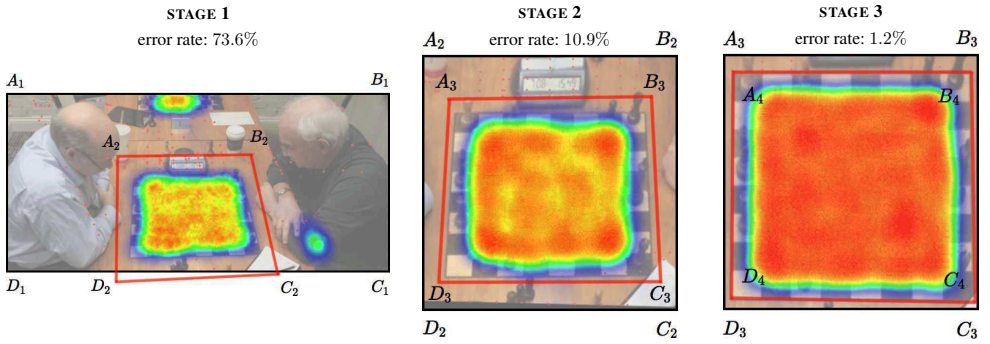

Figure 2. Demonstration of how the lookup process works. In this case, we have three stages. The final stage precisely locates the corners of the chessboard (i.e., $A_4B_4C_4D_4$). It is worth noting that during the first stage, the algorithm also finds a neighboring chessboard. A definition of the error rate can be found in Section 3.2.

The use of a heat map image improves the quality of results and reduces processing time, which is crucial for real-time applications. The iterative process that we implemented mimics the natural mental process that the human brain utilizes. To locate corners of an object, a human first focuses on the entire object by rotating their head to the correct orientation, and then find exact points by performing eye movements. Such an approach can be easily adapted for processing images containing many chessboards. It simply requires the inclusion of a clustering technique that can locate all chessboards during the first stage of the algorithm. However, this is a trivial generalization, so this paper focuses on the detection of a single chessboard.

In the following subsections, we first present the data that we used to perform experiments on and evaluate the results (Section 3.1). Then we present the algorithm that is executed during each stage of image processing (Section 3.2). This algorithm consists of three major sub-procedures that are executed consecutively to locate a chessboard in an image. These sub-procedures are detecting straight lines (Section 3.3), finding lattice points (Section 3.4), and constructing heat maps (Section 3.5). When the final stage of the algorithm is completed, it outputs a cropped image of the chessboard with a corrected perspective. This picture is utilized for locating and classifying chess pieces and generating a description of the board utilizing FEN notation. This process is detailed in the final subsection of the methods description (Section 3.6).

3.1. Materials

We prepared a set of images that we first used during our training process. We published those pictures along with our GitHub project page (`https://github.com/maciejczyzewski/neural-chessboard/tree/draft/test/in`). The selection of pictures was used during the validation process that is described in section 4.

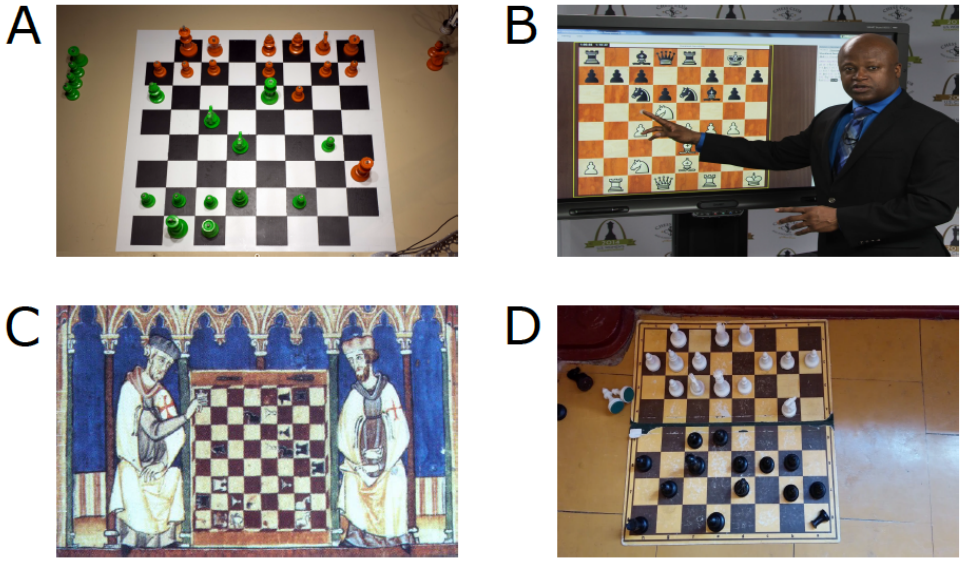

Figure 3. There are multiple pictures in our dataset. Pictures A and D present typical scenarios for chessboard detection algorithms (however, picture A is much easier to detect pieces properly). Picture B is much more interesting since there is a person covering part of a chessboard. Nevertheless we would like our algorithm to restore the chessboard accurately. There is also picture C that presents medieval painting of a chessboard, which should present some difficulty for algorithms.

The figure 3 presents exemplary pictures that are part of our dataset. We can observe that those pictures differ quite distinctly from each other. Some pictures represent images of perfect chessboards with pieces of different colors. Other shows that boards could be damaged or partially destroyed. There could be some obstacle that covers parts of a chessboard. Additionally, we decided to have one picture from a medieval painting.

3.2. Single processing stage

The objective of each stage of the proposed algorithm is to find a better approximation of the chessboard position in an image. It has been proved that the problem of localizing objects in images can be successfully solved by utilizing a deep convolutional neural network

that has been trained to perform image classification tasks. For example, an advanced and very promising method was presented in [17] for polarimetric synthetic aperture radar image classification. However, this method is only capable of classifying various objects appearing in arbitrary positions in an image, but not for exact object positioning. This technique could be extended, as described in [3]. However, it requires 16 iterations to locate an object inside a rectangular frame without accounting for perspective. Therefore, to drastically improve the effectiveness of our approach, we decided to design a hybrid method. We approached the problem by utilizing altered versions of a classic computer vision methods based on algorithms boosted by neural networks to solve various sub-problems. Our method finds the characteristic structures in an image, such as lines and lattice points, and then assesses their locations and shapes based on a scoring function called polyscore (cf. Section 3.5). The values of the polyscore function define the temperature of each point in the heat map. Based on these polyscore values, we can identify components representing a single chessboard as follows:

Definition 3.1 (Heat map component)**.** A heat map component is a connected sub-area of the heat map that is (1) as large as possible and (2) contains only points with temperature (i.e., polyscore value) greater than some threshold t_h.

At the end of each stage, the algorithm chooses the tetragonal frame from the heat map containing a single heat map component (in practice, the frame with maximal polyscore value is selected, cf. Section 3.5), crops this frame with an additional offset P based on the error rate value E, and warps the perspective. We define

$$E = 100\% - A/A_0 \tag{1}$$

$$P = \max\{\sqrt{EA}, \frac{\sqrt{A}}{6}\}, \tag{2}$$

where E in Equation 1 is an error rate and P in Equation 2 is an offset value. The variable A determines the area of a given heat map component and A_0 is the area of the entire image. Additionally, we must consider the compulsory offset in Equation 2 because our algorithm finds only the inner lattice points of the chessboard (i.e., a grid separating 6×6 squares, omitting lattice points located on the edges of the chessboard).

The number of stages required to process an image depends on the level of complexity in that image. Formally, stages are repeated iteratively until the error rate value decreases to below a small threshold value t_e. In practice, one or two stages are typically sufficient to correctly generate a final heat map. We divided the problem of generating a heat map into sub-problems that are solved by separate modules specialized at solving specific sub-tasks. Specifically, we utilize three principal modules:

- The straight line detector (SLID) finds straight lines in an image and filters out lines that are expected to have no usability for detecting a chessboard position. To simplify the recognition of lattice points by the next module, we also designed a novel heuristic algorithm that merges many collinear segments into a single line.

- The lattice points search (LAPS) module selects points that have a high probability of being a lattice point (i.e., a point where the corners of chessboard squares intersect).

This module utilizes a neural network to facilitate a geometric detector for recognizing difficult cases of lattice points.

- The chessboard position search (CPS) module computes a heat map representing the probability that a chessboard is located in a specific sub-area of an image (see Figure 2). The heat map is computed primarily based on the results of the SLID and LAPS modules.

3.3. Detecting straight lines

Line segment detection is a classical problem in image processing and computer vision. SLID is an extension of the standard line detector. Its additional objective is to merge all small segments that are nearly collinear into long straight lines. Therefore, there we utilize a standard line detector to detect segments that can be extracted from an image and merged later. There are several line detectors that can be applied to solve this problem. The Hough transform is a traditional line detector based on an edge map [13], but it typically extracts infinitely long lines instead of line segments and typically outputs many false detections, such as those described by Fernandes and Oliveira [15]. More satisfactory results can be achieved by the *Canny Lines* detector [26], which is the detector that we decided to utilize in our method.

The problem of segment merging has been described by Tavares and Padilha [40]. They presented a solution based on the analysis of the centroids defined by two segments. Unfortunately, this algorithm can generate results that are unintuitive for humans, who typically assume that a merged line should be as collinear as possible with the longest segment. This factor is also omitted by many other available algorithms. One interesting alternative is the method designed by Hamid and Khan [18], which is more perceptually accurate. However, it is a slow method with a computational complexity of $O(n^3)$. Additionally, it only links segments that overlap or are very close to each other, meaning it does account for broken lines (i.e., two segments that are collinear but are far apart from each other).

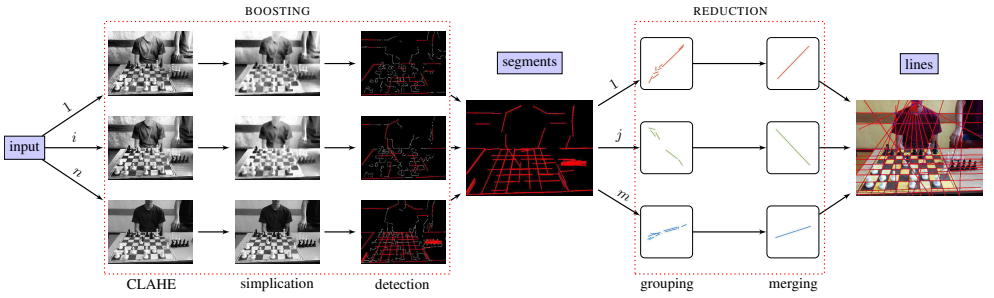

Figure 4. The process of straight line detection. We preprocess images utilizing four sets of parameters (i.e., $n = 3$), each which is utilized to find a portion of the line segments that are then divided into m groups and merged into m straight lines (m denotes the number of groups of collinear segments and is set automatically by the algorithm).

Our proposed SLID algorithm consists of three main steps (for a visual representation,

see Figure 4):

1. Boosting: find all possible segments utilizing multiple analysis of the same image via the gradiental threshold method proposed by Sen and Pal [33] and various contrast limited adaptive histogram equalization (CLAHE) masks [32, 36].
2. Grouping: separate segments into groups of nearly collinear segments utilizing the *grouping function* (cf. Section 3.3.2).
3. Merging: analyze and merge the segments in each group utilizing the M-estimator, resulting in one normalized straight line.

3.3.1. Boosting input

One effective method for boosting the detection of line segments is to adaptively adjust the low and high thresholds of the Canny operator based on the gradient magnitude of the input image. This method can ensure the completeness of an image's structural information and it was utilized in the CannyPF algorithm [26]. Our method is similar, but we also utilize the CLAHE algorithm [32] and a simplification phase based on erosion and dilation operations to reduce noise and remove insignificant details. Our input images pass through a pipeline with different parameters for CLAHE, erosion, and dilation functions (Fig. 4). We determined that to extract all structural information, one must only analyze four different cases, namely parameter sets selected manually to correct low light, overexposure, underexposure, and blur.

3.3.2. Grouping function

The grouping function analyzes two segments $\overline{AB}$ and $\overline{CD}$ and attempts to determine if they are located and oriented in such manner that they can be treated as a single straight line segment. To this end, it utilizes two heights x_1 and x_2, which are the lengths of perpendiculars dropped from points A and B to the line $\overleftrightarrow{CD}$ (cf., Figure 5) and two heights y_1 and y_2, which are calculated similarly for the line $\overleftrightarrow{AB}$. Additionally, we denote $a = |AB|$, $b = |CD|$, and

$$\Delta = (a+b) \cdot t_\Delta \tag{3}$$

$$\gamma = \frac{1}{4}(x_1 + x_2 + y_1 + y_2). \tag{4}$$

Here, Δ is the allowed matching error, γ defines the average deviation from the straight line, and t_Δ represents the acceptable differences in the positions and orientations of lines according to Equation 6.

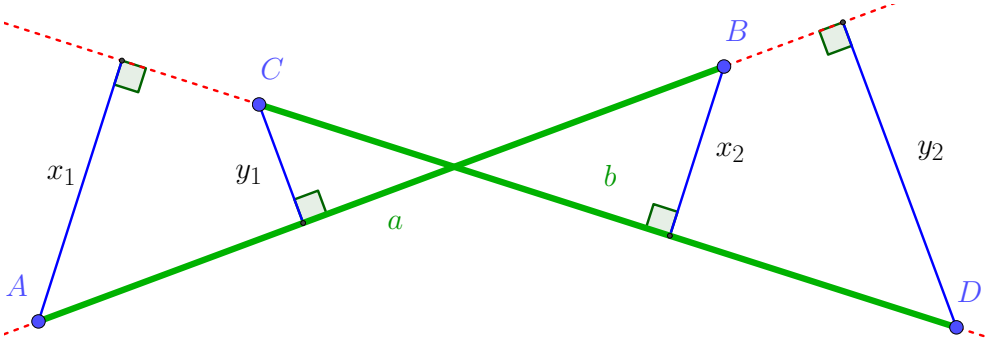

Figure 5. Statistics calculated for each pair of line segments detected in the image.

Based on the variables defined above, we can define a condition that determines if two lines should be merged as follows:

$$\left(\frac{a}{\gamma} > \Delta\right) \wedge \left(\frac{b}{\gamma} > \Delta\right) \tag{5}$$

The parameter t_Δ is defined based on two other parameters, namely p, which defines the degree to which segments should be similar, and ω, which is a scale constant defined with based on the area of the entire image A. Specifically,

$$t_\Delta = p \cdot \omega \tag{6}$$

$$\omega = \frac{\pi}{2} \frac{\sqrt{2\sqrt{A}}}{\sqrt{2A}} = \frac{\pi}{2} \frac{1}{\sqrt[4]{A}}. \tag{7}$$

We assume that it is relatively difficult to determine if two segments whose lengths are smaller than the square of the image size are collinear. For this reason. the image area A in the numerator of the Equation 7 is under a double square root. For example, in our study, pictures were always scaled to have the same size of (500×500 px = 250 000 px) and we wished to connect segments that were similar with a value of $p = 90\%$. Therefore, the calculation of t_Δ was performed as follows:

$$\omega = \frac{\pi}{2} \frac{1}{\sqrt[4]{500 \cdot 500}} \approx 7\% \tag{8}$$

$$t_\Delta = 90\% \cdot 7\% = 6.3\% \tag{9}$$

t_Δ is an important parameter because if segments are short compared to the height or width of the entire image, even if they do not appear collinear, they should be treated as one line. However, the longer these lines are, the stricter the requirement for their similarity should be. This models the real-life visual correctness (perceptual accuracy) of line similarity, which also depends on the scale that the observer sees.

Grouping can be implemented utilizing a disjoint set data structure to achieve the best computational complexity possible [16, 39]. We simply iterate over pairs of segments and if the grouping function determines that a pair is collinear, we perform a union of the sets that contain each segment.

3.3.3. Merging

The final stage of line detection is merging, which is performed for each group of collinear segments (see Figure 6). This process is divided into two steps: (1) converting segments to points, where the number of points depends on the length of a given segment, and (2) fitting a straight line to all points. The fitting algorithm that we utilize is based on the M-estimator of regression for approximately linear models [44], that iteratively fits lines by utilizing the weighted least-squares algorithm to minimize the sum of squares of the residuals. However, it should be noted that the type of estimator utilized does not have a significant impact on the results.

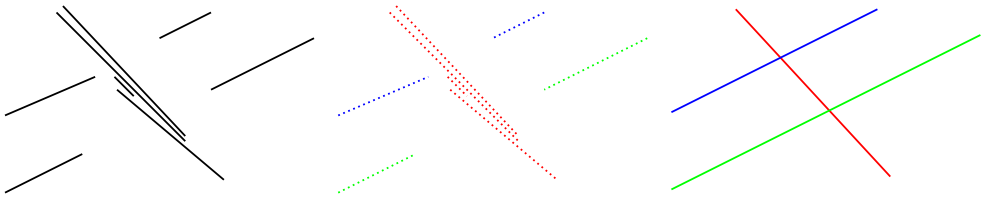

Figure 6. Results of grouping and merging operations. Each color represents a different group of collinear segments. These groups are converted to points and then utilized to fit straight lines.

3.4. Detecting lattice points

As described at the beginning of this paper, the detection of chessboard lattice points is a common problem in computer vision. It has many applications (such as calibrating video systems) and current detectors are fast and acceptably accurate, and robust for most practical applications. However, our algorithm demonstrates that such detectors can be improved even further. Improvements are particularly important for correctly processing poor-quality images, such as images with low-contrast, noise, or those many reflections. LAPS is an experimental, self-learning chessboard vertex detector based on an embedded neural network and is the direct successor of the [4] ChESS detector. Initially, our algorithm assumes that each intersection of any pair of lines detected by the SLID module can be a chessboard lattice point. It processes each of these points utilizing both geometric and neural detectors, and returns a list of points that it detects to be lattice points.

The LAPS algorithm takes a 21×21 matrix whose elements represent pixels as an input. To verify if an (x,y) point in an image is a chessboard lattice point, it utilizes a sub-image with coordinates ranging from $(x-10, y-10)$ to $(x+10, y+10)$ (we decided that this size of the neighborhood is sufficient, and those suspicions were confirmed within experiments). Then, the algorithm preprocesses this matrix utilizing the following steps: (1) conversion to grayscale, (2) application of Otsu method altered by Jassim and Altaani [20], (3) application of Canny detector, and (4) binarization. These four operations ensure the completeness of an image's structural information.

The preprocessed matrix is handled by two modules: (1) a simple geometric detector

that recognizes only perfect cases and (2) a neural network for recognizing deformed and distorted patterns. First, for the geometric detector, if the result is positive, we assume that it represents a chessboard lattice point. Otherwise, we utilize the neural network detector because its result definitively determines if the matrix represents a chessboard lattice point.

3.4.1. Geometric detector

The geometric detector is very simple, meaning it can only recognize trivial cases (see Figure 7). This detector utilizes the following algorithm: (1) add a 1-pixel-width frame of the background color (black) around the input matrix, (2) perform morphological erosion, (3) find all contours and (4) check if the contour resembles a rhomboid. If there are four rhomboids detected in an image, it means that the matrix contains a chessboard lattice point because the rhomboids correspond to four quadrants that are separated by crossing lines.

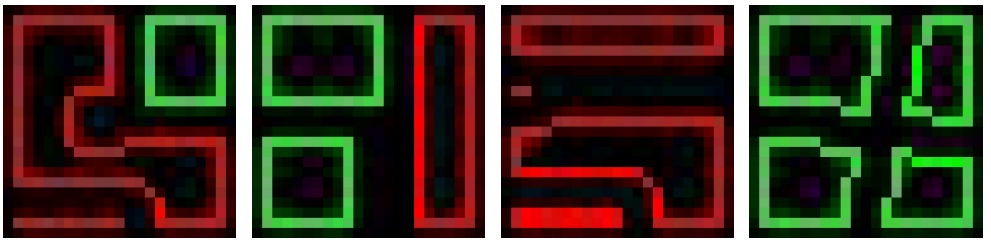

Figure 7. Example results of the geometric detector. Green color represents rhomboids that were identified by the detector. Only the rightmost matrix will be classified as a lattice point because it is the only one that contains four rhomboids.

3.4.2. Neural detector

As already described earlier in this section, a 21×21 matrix is given as the input for the neural network detector. We based our neural detector on a convolutional neural network consisting of two layers: a convolutional 2D layer with 12 filters and a flattened layer with a 0.5 dropout function. The neural network has two outputs with values between zero and one. Such a design is commonly used for binomial classification. In our application, there are two classes denoting if a matrix represents a chessboard lattice point (i.e. classes of positive and negative samples).

It should be noted that we analyzed images that were downscaled to a size of 500×500 pixels. Therefore, the input matrices for our network represent 0.176% of the input images and the detector can learn how other objects overlap with the chess lattice points. For example, our detector can recognize cases where there is a chess lattice point with half of the input matrix hidden behind a pawn's head (appearing as a circle with two tangents).

Our convolutional neural network was trained on a few thousand images of difficult chess lattice points and other patterns resembling lattice points (4,732 positive and 4,933 negative samples). To generate such a large training dataset, we utilized an automated procedure

that modifies perfect, artificial images of 2D standard 8×8 chessboards by warping their perspective in a random direction. In this manner, we prepared 10% of the initial dataset. Additionally, during a training session on real photographs at the end of the process of detecting chessboard positions, we removed all chessboard lattice points (even those that were not detected properly). We then added them to the dataset as positive or negative samples, but only if the current pre-trained network was certain that a sample was positive or negative with a confidence value over 75%. All other samples were ignored. In this manner, we ignored all positive cases in which an object was blocking a clear view (player hands, pawns, or other items) and all negative cases that had some chance to be a lattice point. As a result, we generated a dataset with very difficult, damaged, and deformed samples, which increased the capabilities of our neural detector.

Our dataset is accessible from the RepOD repository under the name LATCHESS21 [9], where we published it for open access [37, 29]. The neural network and pre-trained model are available in the supplementary materials.

3.5. Searching for chessboard positions

The CPS algorithm searches for chessboard positions by analyzing a four-sided frame that can enclose a chessboard. For this purpose, it selects four lines detected by the SLID algorithm that form a quadrilateral, which is later scored. The main problem is how to avoid $\binom{n}{4}$ operations. It turns out that it is trivial to optimize the algorithm by limiting it to a few cases. In particular, this algorithm utilizes the following operations:

1. Select a cluster of lattice points generated by the LAPS algorithm with the largest number of points. Cluster selected in such a manner should represent the primary chessboard in a picture. We donate this cluster as a group G for the rest of this chapter.

2. Calculate $\alpha = \frac{\sqrt{A_G}}{7}$ and the group centroid, where A_G determines the surface area of a group G. Based on this formula, α will approximate the width of a chessboard square.

3. For each chessboard lattice point in the group G, find the lines generated by the SLID algorithm that satisfies the following conditions:

   - their distance from the closest chessboard lattice point is at most α;
   - their distance from the group G centroid is at least $2.5 \cdot \alpha$, which will remove lines that are in the central area of the group;
   - they have a high chance of being near a frame edge (to verify this, we utilize a polyscore function (see Equation 11) and check if its value is not equal or almost equal to 0).

4. Divide the lines identified in the previous step into two groups, namely horizontal and vertical lines, while accounting for perspective.

5. Take pair of lines from each group (two horizontal and two vertical lines) to form a frame and calculate the polyscore of each frame. Finally, choose the frame that maximizes the polyscore.

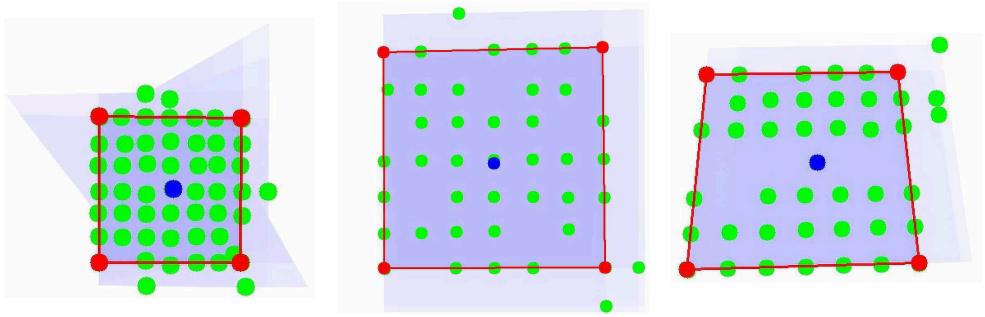

Figure 8. Polyscore values calculated for various frames. The frame with the highest score is marked with the red border. Higher scores are highlighted with darker blue backgrounds. Based on the average values of the polyscore functions for the frames covering each pixel, we can draw the heat maps presented in Figure 2.

In order to analyze the probability that a frame F defined by four lines encloses a chessboard, we utilize the scoring function $P(F)$, which is called a *polyscore* (see Equation 11 and Figure 8). This is an improved density function for chessboard lattice points in a given map segment:

$$W_i(x) = \frac{1}{1 + \left(\frac{x}{A_F}\right)^{\frac{1}{i}}} \tag{10}$$

$$P(F) = \frac{L^4}{A_F^2} \cdot W_3(k) \cdot W_5(l) \tag{11}$$

Here, $W_i(x)$ is a weight function that applies a weight i with respect to the area of the frame F, which is denoted A_F, where L is the number of points inside the frame, k is the average distance of points inside the frame from the nearest side of this frame, and l is the distance between the group G centroid and frame F centroid. The function $P(F)$ was designed in a way that its maximum value for a frame F represents a perfectly cropped chessboard.

3.6. Forsyth-Edwards Notation generation

After finding the position of a chessboard, the algorithm proceeds to the phase of recognizing a chess piece located on the chessboard. The most popular approach for solving this problem is based on utilizing color segmentation to detect pieces and shape descriptors to identify them. However, this method provides unsatisfactory results because chess figures from a bird's-eye view are nearly indistinguishable. In our study, we utilized a similar method described by Ding [12], which achieves approximately 90% accuracy for individual chess pieces. We attempted to utilize both the original support vector machine designed by Ding and an alternative convolutional neural network that we designed. The accuracies of both methods were similar. However, we discovered that we could increase accuracy significantly by implementing the following major improvements:

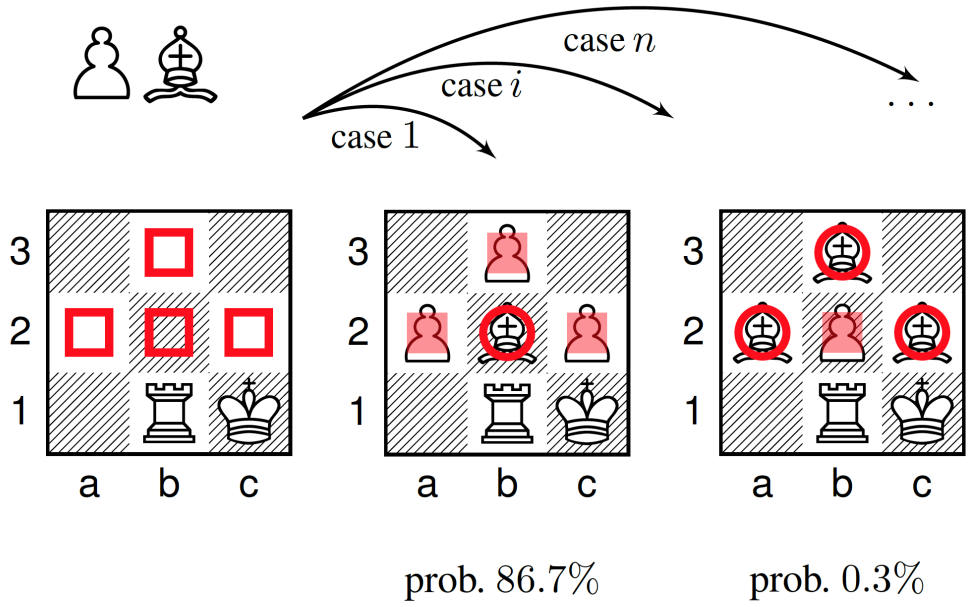

Figure 9. An example situation illustrating cases that we can easily discard as improbable by utilizing a chess engine, such as Stockfish.

1. Utilizing a chess engine: We utilized the open-source Stockfish engine, which allows us to calculate the most probable piece configurations. By calculating the probabilities of all possible configurations, we could choose the most probable candidate (see Figure 9). Additionally, we strengthened this method by utilizing large-scale chess game statistics in the manner proposed by Acher and Esnault [1].

2. Clustering into groups: After clustering similar figures into groups, we can reject certain trivial cases based on the cardinality of clusters. For example, having two groups of similar figures with cardinalities of $\{5, 1\}$ and candidates of $\{\text{bishop}, \text{pawn}\}$, it can be deduced that there are five pawns and one bishop.

3. Considering physical properties: We utilized the height and area of chess figures as additional parameters for classification. A similar mechanism was described by Wu et al. [45].

We determined that the effectiveness of the piece detector became less important after implementing the improvements described above. In fact, we only require a hypothesis regarding which piece could be located in a given square because our module deduces the most probable situation.

Table 1. Comparison of chessboard detection performances. A 0.5 score indicates that the chessboard was cropped properly, but the corners were not perfectly matched for various reasons. The main objective of Ding's research [12] was to introduce a novel approach for chess piece recognition and the authors assumed that a user would manually select the four corners of the chessboard. The accuracy is calculated as a fraction: number of properly detected chessboards divided by the size of a whole set.

Method	Chessboard ID										Accuracy
	1	2	3	4	5	6	7	8	9	10	
Our	✓	✓	✓	✓	0.5	✓	✓	✓	✓	✓	95%
[11]		✓				✓	✓	✓	✓	✓	60%
[10]						✓	✓	✓		✓	40%
[12]	-	-	-	-	-	-	-	-	-	-	N/A

Table 2. Correctness comparisons for chess piece detection. Empty entries in the table indicate that an algorithm did not locate a chessboard correctly, meaning it was unable to analyze it further. The individual entries indicate the numbers of negatively recognized individual squares. The error column contains the average numbers of incorrectly detected pieces. The header line contains the number of available pieces. For each chessboard we bolded the best result.

Method	Chessboard ID (number of pieces)										Error
	1 (29)	2 (24)	3 (21)	4 (21)	5 (19)	6 (23)	7 (15)	8 (14)	9 (31)	10 (26)	
Our	**0**	**0**	**2**	**1**	**5**	**2**	**0**	**0**	**2**	**0**	1.2p
[12]	**0**	5	3	4	23	**2**	2	1	**2**	3	4.5p
[10]						6	4	3		5	4.5p
Best	100%	100%	90.47%	95.23%	73.68%	91.30%	100%	100%	93.54%	100%	94.42%

4. Results

For testing our algorithm, we wished to prepare a challenging benchmark dataset consisting of images of chessboards captured under various, difficult conditions. To collect images containing chessboards and chess pieces that were difficult to recognize, we defined the following conditions for our benchmark images:

- there are objects in the picture other than the chessboard and chess pieces;
- straight lines in the image are not generated solely by the chessboard, but also by other objects;
- the chessboard is not perfect (e.g., computer graphics), meaning it does not fill an entire image and the corners and chess pieces are not emphasized in any way;
- there are shadows, reflections, distortions, and noise in image.

Similar to the methodology utilized by Ding [12], we prepared a benchmark dataset of 30 images of chessboards with a variable number of pieces placed in randomized configurations.

Table 3. Effectiveness comparisons of chessboard lattice point detectors. The final column lists the total times required to process all lattice points for all tested chessboards. The accuracy is calculated as a fraction: number of properly detected lattice points divided by the number of actual lattice points.

Detector	Accuracy	Time
LAPS	99.57±0.0147%	4.57s
ChESS	74.32%	3.38s

The sources of the images were very diverse; some were taken by us, some originated from broadcasts of chess tournaments (we received the consent of the organizers to utilize their images), and we even included one image scanned from a painting from the thirteenth century. To compare our algorithm to other methods, we selected the 10 most-challenging photos from our benchmark dataset. The other twenty photos were utilized for debugging and training classifiers.

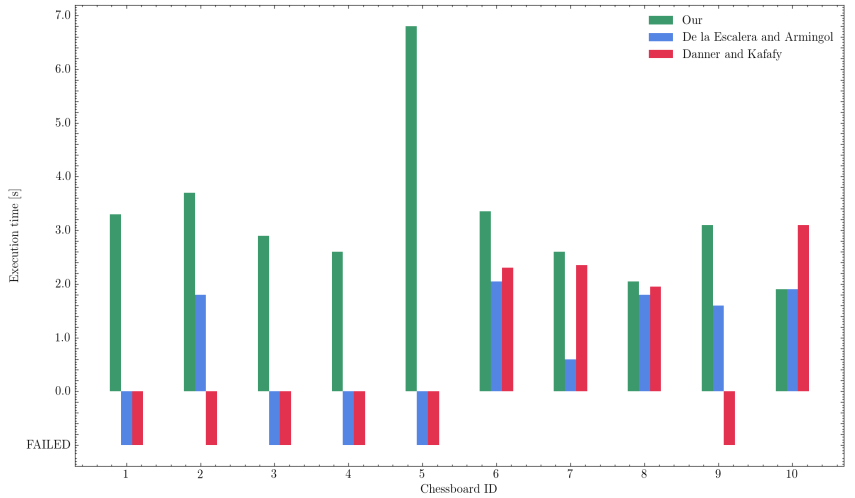

Figure 10. Comparison of the speeds of popular chessboard detection algorithms on our benchmark dataset. When an algorithm could not find the chessboard, the processing time was typically much shorter because certain parts of the algorithm were omitted. For this reason, we only included processing times for cases where a chessboard was successfully located in the source image.

Prior to testing the quality of chess piece recognition, we analyzed how our chessboard detection algorithm performs compared to other methods. For comparison methods, we selected the best-performing current methods. The results are listed in Table 1. One can see that cropping a chessboard from an image is not a trivial problem. Only our algorithm was able to locate chessboards in a high percentage of cases. The other methods tested located no more than 60% of chessboards correctly. This issue is often dissembled by the authors of

papers that are focused on chess piece identification. They simply select test photos for which their algorithm locates a chessboard correctly. It should be noted that we have not tuned our algorithm for these test cases because we utilized the other 20 photos from the benchmark dataset to train our algorithm.

To derive additional insights into why our algorithm is so successful at locating chessboards, we also compared our chessboard lattice point detector to the best current alternative, namely the ChESS detector [4]. As shown in Table 3, our algorithm has very high accuracy (over 99.5%) compared to the 74.3% accuracy of the ChESS detector. Additionally, it is only slightly slower than the ChESS detector.

Next, we tested the correctness and accuracy of our chess piece recognition algorithm. During our tests, we compared our algorithm to the same set of methods that we utilized to analyze chessboard detection performance, excluding the approach that was dedicated exclusively to chessboard detection and did not implement chess piece recognition (i.e., [11]). The results presented in Table 2 demonstrate that our method is characterized by high accuracy and stability for challenging input cases. The method by Danner and Kafafy [10] was unsuccessful when analyzing images of poor quality and those containing a large number of extraneous objects. The same problem was identified in the method by Ding [12], which was misled by images containing too many extraneous lines occurring in regular patterns.

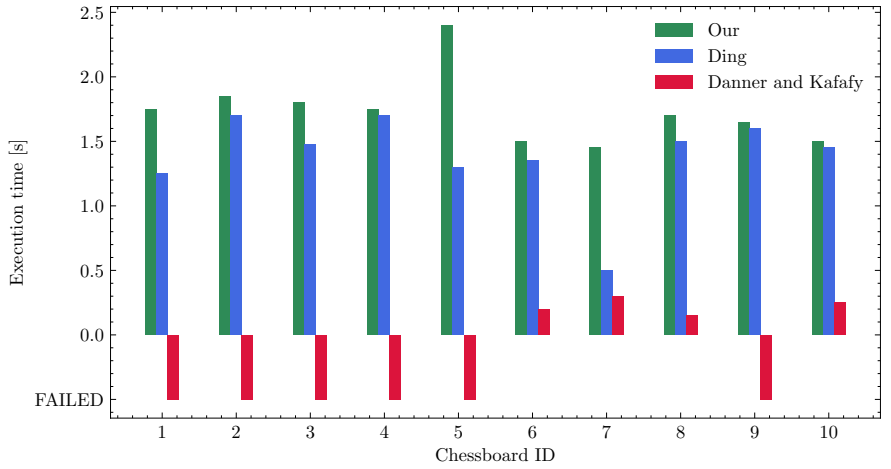

Figure 11. Comparison of the speeds of popular chess piece detection algorithms on our benchmark dataset. When an algorithm could not find the chessboard, it was not possible to utilize it for identifying chess pieces. For this reason, we only included processing times for cases where a chessboard was successfully located in the source image.

The only disadvantage of our method is that it is considerably slower than alternative approaches. As shown in Figure 10, in some cases our algorithm is over two times slower than its competitors (we performed all our tests at personal computer under macOS High Sierra operating system with 2.6 GHz Intel Core i5 and 8GB 1600MHz DDR3 RAM memory). The case that was especially difficult for our algorithm was chessboard number five. However, this is the only case for which the processing time was longer than 5 seconds, which was

defined as the time limit at the beginning of our study. When comparing the performances of chess piece detection algorithms, one can see that the method by Danner and Kafafy is an indisputable leader in terms of speed. It is sometimes up to 10 times faster than other algorithms when it works correctly (cf. Figure 11). Again, chessboard five was the most challenging case for our algorithm for chess piece detection.

Our implementation of the straight line detector (SLID algorithm) attempts to merge many short segments that are detected in an image into longer segments. This allows us to find essential lines that could be missed by algorithms that search only for longer segments. To demonstrate this principle, we visually compared our method to a method utilizing a single Hough transform for identifying straight lines in Figure 12.

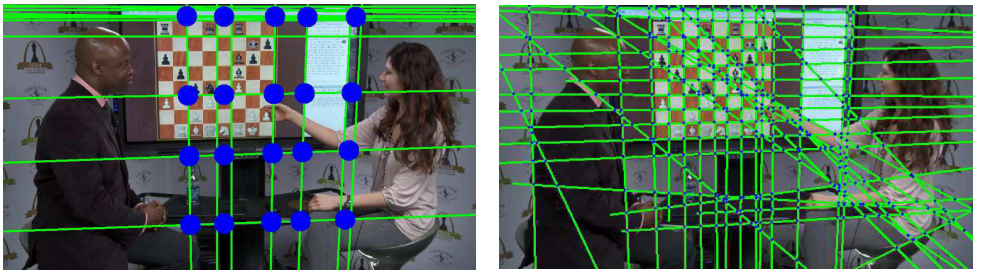

Figure 12. Visual differences between our algorithm (right) and the method by Danner and Kafafy [10], which utilizes a single Hough transform without segment merging (left). One can see that our method generates much more information from an image, which can later be utilized to find lattice points and localize a chessboard.

Another advantage of utilizing computer vision methods boosted by machine learning can be observed in our lattice points detector. By training the classifier utilizing a convolutional neural network, we are able to correctly detect deformed cases of chessboard lattice points, which are poorly recognized by other algorithms. For example, in Figure 13, we present the results of the best-performing alternative method, namely the ChESS classifier. We observed that this method incorrectly classifies 30% of deformed points, which has a negative influence on the accuracy of locating the chessboard because these points are often key features for analysis.

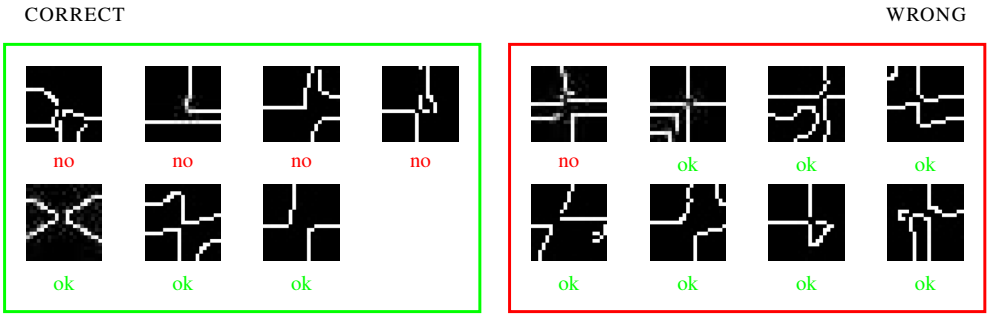

Figure 13. Results of the ChESS detector on randomly selected cases. The cases detected by ChESS as positive are on the left and those detected as negative are on the right. Under each chessboard lattice point, there is a ground truth label (*no* if it is not a chessboard lattice point and *ok* if it is a chessboard lattice point). Here, one can see that the ChESS detector only identified 43% of the true positive cases and 13% of the true negative cases.

5. Conclusions

In this paper, we presented a novel approach for chessboard and chess piece detection. To improve its performance, we enhanced various computer vision methods that are utilized during the detection process, such as the chessboard lattice point detector or line detector with segment merging. When possible, we attempted to utilize machine learning methods to improve the accuracy of the classifiers implemented at various steps of the algorithm. When designing our methods, we attempted to make them as robust as possible to achieve accurate results, regardless of the quality of a source image. We were able to develop a method that is generic and multipurpose. It allows us to analyze damaged chessboards with deformed edges, images with poor quality, and chessboards with certain parts hidden (e.g., board hidden behind a player's fingers). The only disadvantage of our method is that it is slower than alternative approaches. The full source code for our method can be found at our GitHub repository `https://github.com/maciejczyzewski/neural-chessboard`.
Additionally, we agree with Jialin Ding, who stated that the main difficulty in research on recognizing chessboards and chess pieces is the lack of comprehensive labeled datasets of images containing chessboards [12]. Lack of good datasets is often a problem in other fields of computer science research [43, 31].

We were able to achieve very satisfactory results when locating chessboards, largely because of the patient tuning of the methods that are utilized during our process. In this manner, we implemented a very robust method that performs much better than methods utilizing default versions of algorithms available from computer vision libraries (e.g., the basic algorithms implemented in the method by Danner and Kafafy [10]). Basic computer vision tools, such as line detectors, often fail in advanced analysis because they cannot identify a full chessboard structure. Therefore, they are not able to find chessboard positions accurately.

When our method succeeds in locating a chessboard in an image, it proceeds to recognize chess pieces. To solve this problem, we utilized a similar approach to that described by

Ding [12]. However, we introduced three major modifications (see Section 3.6) that have a substantial impact on the final results. Boosting our method with the *Stockfish* chess engine had a significant influence on the accuracy of classification. By analyzing the probability of different configurations of chess pieces based on the scores returned by Stockfish, we are able to reject many scenarios that have a small probability of being real cases. This helps to eliminate cases where bishops are predicted as pawns, which is a common scenario described by Ding [12].

In summary, we were able to implement a method that outperforms all alternative approaches that are currently publicly available. With an accuracy of chess piece recognition close to 95%, it is the first method that can be utilized in real-world scenarios to digitize chess tournaments or share records of chess games between friends. When digitizing an entire game, the accuracy can be improved further because the number of differences between two consecutive states of a chessboard is significantly limited by the rules of chess. The best confirmation of our method's usefulness is the significant interest that arose after the publication of our algorithm on GitHub. We have been contacted by several people who wish to utilize our method in practical applications. If our method is put into practice, it will be easy to improve it by adding data received from users to the training datasets utilized to train our classifiers.

Acknowledgements

The authors would like to give special thanks for the sharing of materials and images from various broadcasts: Saint Louis Chess Club, John Bartholomew, and Tata Steel Chess Tournament. The authors received consent from the organizers of these tournaments to utilize screenshots from broadcasts to train algorithms, visualize results, and provide supplementary materials. Additionally, SW and AL were supported by the Polish National Center for Research and Development grant no. LIDER/004/103/L-5/13/NCBR/2014.

References

[1] Acher M. and Esnault F. Large-scale analysis of chess games with chess engines: A preliminary report, 2016.

[2] Arca S., Casiraghi E., and Lombardi G. Corner localization in chessboards for camera calibration. In *Proceedings of International Conference on Multimedia, Image Processing and Computer Vision (IADAT-micv2005)*, 2005.

[3] Bency A. J., Kwon H., Lee H., Karthikeyan S., and Manjunath B. Weakly supervised localization using deep feature maps. In *European Conference on Computer Vision*, pages 714–731. Springer, 2016.

[4] Bennett S. and Lasenby J. Chess–quick and robust detection of chess-board features. *Computer Vision and Image Understanding*, 118:197–210, 2014.

[5] Braje W. L., Kersten D., Tarr M. J., and Troje N. F. Illumination effects in face recognition. *Psychobiology*, 26(4):371–380, 1998.

[6] Choudhury Z. H. Biometrics security based on face recognition. Master's thesis, India, 2013.

[7] CoolThings. Square off is a robot chess board that can move pieces on its own, November 2016.

[8] Cour T., Lauranson R., and Vachette M. Autonomous chess-playing robot. *Ecole Polytechnique, July*, 2002.

[9] Czyzewski M. A., Laskowski A., and Wasik S. Latchess21: dataset of damaged chessboard lattice points (chessboard features) used to train laps detector (grayscale/21x21px), 2018.

[10] Danner C. and Kafafy M. Visual chess recognition, 2015.

[11] De la Escalera A. and Armingol J. M. Automatic chessboard detection for intrinsic and extrinsic camera parameter calibration. *Sensors*, 10(3):2027–2044, 2010.

[12] Ding J. Chessvision: Chess board and piece recognition. Technical report, Stanford University, 2016.

[13] Duda R. O. and Hart P. E. Use of the hough transformation to detect lines and curves in pictures. *Commun. ACM*, 15(1):11–15, Jan. 1972.

[14] Edwards S. J. Portable game notation specification and implementation guide. *Retrieved April*, 4:2011, 1994.

[15] Fernandes L. A. and Oliveira M. M. Real-time line detection through an improved hough transform voting scheme. *Pattern recognition*, 41(1):299–314, 2008.

[16] Galler B. A. and Fisher M. J. An improved equivalence algorithm. *Commun. ACM*, 7(5):301–303, May 1964.

[17] Gao F., Huang T., Wang J., Sun J., Hussain A., and Yang E. Dual-branch deep convolution neural network for polarimetric SAR image classification. *Applied Sciences*, 7(5):447, 2017.

[18] Hamid N. and Khan N. Lsm: perceptually accurate line segment merging. *Journal of Electronic Imaging*, 25(6):061620, 2016.

[19] Harris C. and Stephens M. A combined corner and edge detector. In *Alvey vision conference*, volume 15, pages 10–5244. Citeseer, 1988.

[20] Jassim F. A. and Altaani F. H. Hybridization of otsu method and median filter for color image segmentation, 2013.

[21] Kanchibail R., Suryaprakash S., and Jagadish S. Chess board recognition. Not published in journal, 2016.

[22] Koray C. and Sumer E. A computer vision system for chess game tracking. In *21st Computer Vision Winter Workshop, Rimske Toplice, Slovenia*, 2016.

[23] Larson C. China's massive investment in artificial intelligence has an insidious downside. *Science*, feb 2018.

[24] Leonard J., Durrant-Whyte H., and Cox I. Dynamic map building for autonomous mobile robot. In *IEEE International Workshop on Intelligent Robots and Systems, Towards a New Frontier of Applications*. IEEE, jul 1990.

[25] Li Q., Zheng N., and Cheng H. Springrobot: A prototype autonomous vehicle and its algorithms for lane detection. *IEEE Transactions on Intelligent Transportation Systems*, 5(4):300–308, dec 2004.

[26] Lu X., Yao J., Li K., and Li L. Cannylines: A parameter-free line segment detector. In *Image Processing (ICIP), 2015 IEEE International Conference on*, pages 507–511. IEEE, 2015.

[27] Marciniak T., Chmielewska A., Weychan R., Parzych M., and Dabrowski A. Influence of low resolution of images on reliability of face detection and recognition. *Multimedia Tools and Applications*, 74(12):4329–4349, jul 2013.

[28] Matuszek C., Mayton B., Aimi R., Deisenroth M. P., Bo L., Chu R., Kung M., LeGrand L., Smith J. R., and Fox D. Gambit: An autonomous chess-playing robotic system. *2011 IEEE International Conference on Robotics and Automation*, pages 4291–4297, 2011.

[29] Mietchen D., Wodak S., Wasik S., Szostak N., and Dessimoz C. Submit a topic page to plos computational biology and wikipedia. *PLOS Computational Biology*, 14(5):1–4, 05 2018.

[30] Pomerleau D. and Jochem T. Rapidly adapting machine vision for automated vehicle steering. *IEEE Expert*, 11(2):19–27, apr 1996.

[31] Prejzendanc T., Wasik S., and Blazewicz J. Computer representations of bioinformatics models. *Current Bioinformatics*, 11(5):551–560, 2016.

[32] Reza A. M. Realization of the contrast limited adaptive histogram equalization (clahe) for real-time image enhancement. *Journal of VLSI signal processing systems for signal, image and video technology*, 38(1):35–44, 2004.

[33] Sen D. and Pal S. K. Gradient histogram: Thresholding in a region of interest for edge detection. *Image and Vision Computing*, 28(4):677–695, 2010.

[34] Shortis M. Calibration techniques for accurate measurements by underwater camera systems. *Sensors*, 15(12):30810–30826, 2015.

[35] Soh L. Robust recognition of calibration charts. In *6th International Conference on Image Processing and its Applications*. IEE, 1997.

[36] Stark J. A. Adaptive image contrast enhancement using generalizations of histogram equalization. *IEEE Transactions on Image Processing*, 9(5):889–896, May 2000.

[37] Szostak N., Wasik S., and Blazewicz J. Hypercycle. *PLOS Computational Biology*, 12(4):e1004853, apr 2016.

[38] Tam K. Y., Lay J. A., and Levy D. Automatic grid segmentation of populated chessboard taken at a lower angle view. In *Computing: Techniques and Applications, 2008. DICTA'08. Digital Image*, pages 294–299. IEEE, 2008.

[39] Tarjan R. E. Efficiency of a good but not linear set union algorithm. *J. ACM*, 22(2):215–225, Apr. 1975.

[40] Tavares J. M. R. S. and Padilha A. J. M. N. A new approach for merging edge line segments. *Proceedings RecPad'95, Aveiro*, 1995.

[41] Urting D. and Berbers Y. Marineblue: A low-cost chess robot. In *Robotics and Applications*, 2003.

[42] Wasik S., Fratczak F., Krzyskow J., and Wulnikowski J. Inferring Mathematical Equations Using Crowdsourcing. *PLOS ONE*, 10(12):e0145557, dec 2015.

[43] Wasik S., Prejzendanc T., and Blazewicz J. ModeLang - a new approach for experts-friendly viral infections modeling. *Computational and Mathematical Methods in Medicine*, 2013:8, 2013.

[44] Wiens D. P. Asymptotics of generalized m-estimation of regression and scale with fixed carriers, in an approximately linear model. *Statistics & probability letters*, 30(3):271–285, 1996.

[45] Wu Q., Zhang J., Lai Y.-K., Zheng J., and Cai J. Alive caricature from 2d to 3d, 2018.

[46] Zhang Z. A flexible new technique for camera calibration. *IEEE Transactions on pattern analysis and machine intelligence*, 22(11):1330–1334, 2000.

[47] Zhao F., Wei C., Wang J., and Tang J. An automated x-corner detection algorithm (axda). *JSW*, 6(5):791–797, 2011.

Received 30.05.2020, Accepted: 12.11.2020