Revista Eletrônica de Iniciação Científica em Computação, 2025, 23:1
ISSN: 3085-8461 • doi: 10.5753/reic.2025.6046

RESEARCH PAPER

Aiding Novice Chess Moves with YOLO Pose Detection and Augmented Reality

Almir Moreira da Silva Neto [Universidade Estadual de Feira de Santana | *amsilva@ecomp.uefs.br*]
Cláudio Eduardo Góes [Universidade Estadual de Santa Cruz | *cegoes@uesc.br*]

Abstract. This paper presents the development of a system combining deep learning, computer vision, and augmented reality to aid learning chess beginner-level players. The proposed approach utilizes a deep learning YOLO model to detect the chessboard and pieces. A series of computer vision algorithms are applied to segment the chessboard into a grid and position detected pieces within the squares. The system provides interactive assistance to the user overlaying helpful information on the augmented reality view. This is achieved by highlighting possible moves for each piece. The custom-trained YOLO model achieved an overall precision of 95% in chessboard and pieces detection. The chessboard keypoints boundary detection reached 99% accuracy. Our chessboard segmentation algorithm was able to correctly identify 98% of the chessboards in the validation dataset. An error rate of 1.45% per chessboard square was achieved while positioning the pieces within the grid. The whole processing pipeline demands an average of 454 *ms* per image. Future works may explore end-to-end deep learning approaches to improve board and piece localization detection. Additionally, user studies are proposed to evaluate the system's effectiveness in aiding beginner chess players to improve their skills.

Keywords: Deep learning, YOLO, computer vision, augmented reality, keypoints detection, chess

Received: 25 May 2025 • **Accepted:** 29 August 2025 • **Published:** 12 September 2025

1 Introduction

Chess is one of the most played board games. It promotes cognitive aspects like memory, problem solving, concentration and creativity [Gobet, 2018; Joseph and Easvaradoss, 2021; Jankovic and Novak, 2019; Unterrainer and et al., 2006; Billinghurst, 2002]. Some emerging technologies, such as augmented reality (**AR**), can be used in teaching as it promotes immersion [Yusof *et al.*, 2019]. Integration between the real and virtual enhances the learning experience [Billinghurst, 2002]. AR can enhance user perception and integration with real world, making learning more interactive [Kesim and Ozarslan, 2012].

Developing applications with AR technologies could accelerate and simplify chess learning. Recent studies have focused on creating AR applications for helping non-beginners improve their skills or to record game state into a database for later use on analysis [Orémaš, 2018; Menezes and Maia, 2023].

To support the development of our proposal, we constructed a dataset containing images from chess games captured from a variety of positions, angles, environments, lightning conditions and different pieces distributions across the board. The final dataset consists of 1,428 images distributed into 1,228 for training and 200 for validation.

We hypothesize that using a richer annotation format, specifically keypoints, can enhance piece localization on the chessboard. This improvement may benefit a wide range of application such as AR, where accurate piece positioning is important to provide useful and reliable feedback to the end user. Due to the lack of publicly available keypoint-annotated chess games, we manually annotated each piece and chessboard present on original images in the dataset. This dataset provides good resources for further research in chess piece recognition and localization.

This paper describes the development of a system combining Deep Learning (**DL**) and AR to aid beginner-level players learn chess from real board games. A mixture of approaches is proposed to achieve desired result. DL technique is used to detect the chessboard and pieces as keypoints classes predicted in an image. Computer Vision (**CV**) algorithms are applied to split chessboard in a grid of squares and to position detected pieces on it. To provide interactive assistance, helpful information is overlaid on AR view. Studies like Mehta [2020] have used AR to help players with some chess knowledge. In Yusof *et al.* [2019], AR chess game was proposed, a completely digital version of game providing hints and tips.

Our approach demonstrates high accuracy in the visual understanding of chessboard scenes. It achieves a classification precision of 95% for both board and pieces. In addition to classification, it achieved piece positioning error of 18.19%, which corresponds to an average error of 1.45% per board square and it was able to precisely segment the board with a precision of 98%. Although we achieved high accuracy across all tasks, our system lacks efficiency in execution time, making it less suitable for real-time applications.

1.1 Related Works

Chessboard and chess piece detection and recognition is a complex task. Through CV there are different techniques to detect the chessboard, pieces, and classify them based on its shape and color.

Some authors, such as Delgado Neto and Campello [2019] uses DL on a set of synthetic generated images for detecting the class of the piece, achieving 97% of accuracy, and some algorithms like Hough transform (**HT**) [Gonzalez and Woods, 2017] for detecting chessboard and finding its boundaries.

In Orémaš [2018], DL was not used for identifying pieces' class. A template matching between the real images and a generated one was used, for the board detection HT algorithm was applied. The result of the detection is then used to generate an image containing two-dimensional (**2D**) representation of the pieces and their positions, this kind of approach allows chess players to log the state of the game from an image. The authors built a specific dataset for their work taking individual photographs for a game or sets of images of the same distribution from different camera angles. Alongside with chess game images, authors have also built digital versions of the pieces, which they considered much easier than photographing pieces from various angles and labeling them. One of the problems detected by the author is the need of more than one photograph of the game to improve it's accuracy since piece to piece occlusion confuses the template matching. By averaging the three images for classification achieves 98% of success rate.

Recent works, such as Menezes and Maia [2023], used cutting edge object detection models for the chess piece detection. They used two versions of YOLOv4 (tiny and full), to address the detection, through a dataset containing more than 6000 images, of which all images were taken from above, and each piece was labeled using bounding boxes, in addition to the bounding box, each sample was given a category, each one representing a chess piece name. They were able to identify the piece and it's location with a precision of 89%. A 2D representation of the state of the game is the output of the system allowing users to log moves in a fast way. How the chessboard is detected and piece positions are reflected to the digital one is unclear.

In Mehta [2020], an AR application was developed to improve chess players into logging and improving their games. DL for detecting and identifying the pieces was used. Binary detector to infer if the chessboard is present or not, a couple of algorithms to segment chessboard and an integration with a chess engine to log, suggest plays and improve user skills based on current state of the game. AR enables the system to suggest the best movement over the real chess game. This system was able to achieve an accuracy of 93.45% for entire chess position prediction pipeline. For model training, a dataset containing approximately 2600 piece samples was built, it consists of images taken from above and labeling each piece occurrence with a category in a similar way as in Menezes and Maia [2023].

Czyzewski *et al.* [2020] was able to achieve 95% accuracy when positioning the chessboard in an image. Almost 95% accuracy for chess piece recognition. They use a neural detector to predict chessboard lattice points. This algorithm outperforms the best alternative where it only achieved 60% for chessboard detection. It also uses a heatmap to plot the probability of a chessboard is located in a sub-area of the image. This work is focused in detecting the presence of a chessboard and precisely segmenting it, a dataset was constructed containing chessboard images taken under various conditions where chessboard lattice points are provided as labels.

By using a set of algorithms for detecting and splitting the chessboard into a 8×8 grid, Wölflein and Arandjelović [2021] uses different DL models to predict grid square occupancy and what piece is within it. This approach was able to achieve 99% of accuracy in most of tried models. Instead of relying on manually taken chessboard images, the authors employed a 3D model of a wooden chess set as a base and subsequently generated realistic images from different positions, angles, and lighting conditions. Each chess piece location is annotated using FEN format, a standard notation for chess piece localization.

In Masouris and van Gemert [2024], authors indicated that chessboard detection is commonly split in various steps, such as chessboard detection, square localization, and piece classification. They propose an end-to-end DL solution directly predicting the configuration from an image, successfully recognizing 15% of the pieces. Model receives an image as input and outputs the type and pieces positions relative to chessboard. This work introduces a distinct dataset, consisting of chess games images captured from different angles, annotating chess pieces using bounding boxes and labeling each piece with a chess piece category, as well as points defining boundaries of the chessboard.

Research on identifying chess pieces is a very complex task. They have different shapes, sizes and colors. DL use is a common technique for this kind of problem, as models are able to learn and identify complex and different shapes.

The related works above show that most of the efforts are aimed at aiding non beginners chess players, through logging chess game state and suggesting movements.

2 Methods

2.1 Dataset

Building a custom dataset was necessary for this work. The existing datasets from related works do not offer keypoint annotations required for our proposal. In fact, the available datasets vary significantly in format and purpose, Wölflein and Arandjelović [2021] provides a synthetically generated dataset but without any annotations, Masouris and van Gemert [2024] includes bounding boxes for the pieces and corner points for the board, top-down images of the board with piece annotations, but only from a fixed viewpoint, was presented by Menezes and Maia [2023].

Compared to existing datasets, our dataset introduces a new, unpublished annotation format based on keypoints. Each chess piece and the board's boundaries are annotated with keypoint locations. Additionally, the images in our dataset were captured from various angles and positions, instead of being restricted to top views, making the dataset more versatile and realistic for different applications.

A dataset containing examples of real boards and annotated pieces was built to feed the model. All the photos were taken with 4032×3024 px spatial resolution and 24 $bits$ color deep contrast resolution.

Photos were captured in different environments, lighting conditions, shadows and backgrounds to ensure image diversity. Some photos were obtained from different angles and various combinations of chess pieces. Sometimes only black pieces were placed on the board, while occasionally only white pieces were set. A few pieces are symmetrical, and thus rotation does not change their visual information. However, pieces such as the Knight and King can have different shapes based on the point of view. To ensure a wide variety of non-symmetrical piece rotations were collected, the same distribution was photographed, with only the rotation of the irregular pieces being varied: Knight, King, Bishop, and Rook.

Almost all images in the dataset contain a visible board in it. To improve the learning rate of the chess pieces, photos where only one or some pieces are visible were taken. Twelve images where only a single piece is visible, one image containing only black pieces in line, one image containing only white pieces also in lines and one image containing both white and black pieces in line were included. The dataset contains 72 images where only the board is visible. The remaining images were captured with no specific procedure, pieces were randomly placed over the board to guarantee a diverse combination of images. Some image samples present in the dataset are shown in Figure 1.

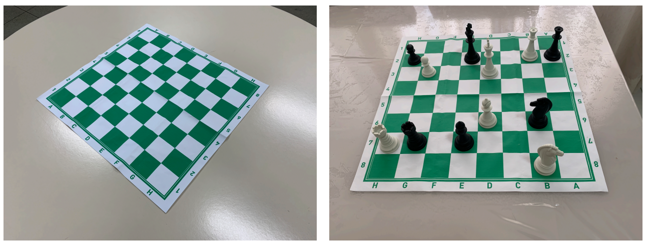

(a) Only board visible. **(b)** Random set of pieces over the board.

Figure 1. Some dataset image samples of this work.

Almost 500 photos were taken but only 453 images were kept in the dataset, because some images had defects such as blurring and light noise.

Each image in the dataset was annotated by identifying each chess piece according to its game class. As detailed in Table 1, chess pieces were assigned the corresponding chess class labels ranging from number 1 to 12, while the chessboard itself was assigned the label "board", corresponding to class 0.

Chess has an irregular distribution over its pieces i.e. there are more pieces of a class than another such as Pawns and Rooks, that have 8 and 2 pieces each, respectively. To prevent more classes appearing than another, some images were taken with fewer pieces of some classes. The same was done for the board, since for each chess game there is only board for a bunch of pieces. As showing in Table 1 board class exhibits one of the lowest frequency when compared to the other classes.

Initially, images were annotated manually using CV Annotation Tool[1] (**CVAT**). A better description of the structure of piece and board annotation using keypoints is discussed in Section 2.2.

The dataset was split into training and validation sets to maintain a separate subset of images for evaluating model accuracy. The original distribution consisted of 253 images for training and 200 images for validation, totaling 453 images. This separation ensures the validation process uses images not seen during training.

[1]CVAT. Available in: `https://github.com/openvinotoolkit/cvat`.

Table 1. Dataset distribution with data augmentation

No.	Class	Instances	Augment 4×	**Total**
0	Board	238	952	**1,190**
1	Black-Pawn	799	3,196	**3,995**
2	White-Pawn	818	3,272	**4,090**
3	Black-Rook	254	1,016	**1,270**
4	White-Rook	253	1,012	**1,265**
5	Black-Bishop	261	1,044	**1,305**
6	White-Bishop	253	1,012	**1,265**
7	Black-Knight	260	1,040	**1,300**
8	White-Knight	253	1,012	**1,265**
9	Black-Queen	142	568	**710**
10	White-Queen	142	568	**710**
11	Black-King	148	592	**740**
12	White-King	149	596	**745**

2.1.1 Image augmentation

Prior to model training, we increased the number and diversity of training samples in the dataset by applying image augmentation techniques using the open source library Albumentations[2]. It provides a variety of algorithms to perform image augmentation, an artificial image generation through the introduction of new samples from the original image, which works by applying a series of image processing and computer graphics operations over an image [Buslaev *et al.*, 2020].

This augmentation library transformation was applied over all training images in the dataset three times, increasing its size from 253 to 1012 training samples, including original ones. To achieve this result, each original image sample on the original dataset was iterated and a series of transformations was applied on the image.

As earlier discussed in this section, chessboard is a low-populated class due to the nature of chess games. To increase the number of chessboard occurrences in the dataset, all training images containing only a chessboard were augmented additionally. The original training dataset consisted of 72 chessboard-only images, each augmented three times, resulting in an expanded training set of 1,228 images. The final augmented dataset comprised 1,428 images in total, distributed as 1,228 (85%) for training and 200 (15%) for validation.

The following transformations were applied during data augmentation, each with a specified probability to prevent uniform augmentation across all images: normalization (100%), Gaussian blur (25%), Gaussian noise (20%), color jitter (50%), defocus (15%), shear (100%), rotation (100%), and scaling (40%). To ensure diversity, only one of the last three geometric transforms (shear, rotation, or scaling) could be applied to a single image at a time. Their probabilities were weighted such that scaling was less likely (40%), while shear and rotation shared equal probability (100% each).

The augmented image transformations described previously were applied to achieve diverse sample images. Normalization ensures all pixel values are constrained within a specific range.

A Gaussian blur transformation applies some kind of smearing to image reducing high frequency signals, where

[2]Albumentations: fast and flexible image augmentations. Available in: `https://github.com/albumentations-team/albumentations`.

the kernel follows a Gaussian distribution. Albumentations chooses, randomly, the size of the kernel.

Gauss noise was used to apply noise over an image. Color jitter transformation randomly changes brightness, contrast and saturation of an image. Defocus transformation is responsible for defocusing the image, randomly selecting a radius range.

Last three transformations apply a spatial distortion to the image. Shear deformation was used a range between $\pm 30^{o}$. Rotation transformation applied a random rotation, between $\pm 30^{o}$. Scaling could be applied as well to an image, using a ratio between 50% and 100% resizing the image.

These transformations were specifically chosen so different situations can be simulated such as focus problems, color distortions and noises. The last three transformations are able to mimic common camera movements similar to rotation and possible distortions due to human handling, and thus, are important transformations to be applied.

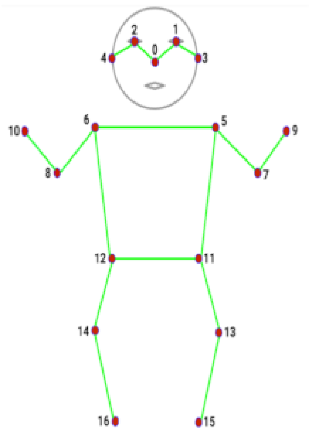
2.2 Keypoint detection

Through the development of DL, some models were developed to do specific tasks in several field of technology, such as *You Only Look Once* (**YOLO**) [Redmon *et al.*, 2016], SSD [Liu and et al., 2016] and Faster R-CNN [Ren and et al., 2015]. These models started as DL object detection models, i.e., they learned how to detect object in an image, YOLO is a fast, real-time approach to this task [Redmon *et al.*, 2016]. YOLOv8 from Ultralytics, implements detections like: object detection and segmentation, instance segmentation, object tracker and pose estimation. In this study, this framework[3] from Ultralytics was adopted.

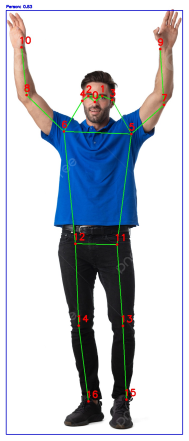
Different YOLOv8 models were tried to achieve object detection in this study. Firstly object detection model was used to provide a geometric information localization. Major problem of this approach was, a 2D geometric information like a bounding box around the object is detected. This prevents obtaining information about the piece base and chessboard orientation.

Instance segmentation was our next attempt. This approach assigns, for each pixel in an image, a class based on labels. Since this approach is pixel-based, some information can be lost during occlusion, for example, when a piece is behind another.

At last, pose estimation model was evaluated. Despite it's mainly used for human pose estimation, this model was tried to assess if it is a viable option to this scheme. Standard YOLOv8 pose estimation model uses 17 keypoints to identify parts of an human pose, in this specific task, human's parts such as hands, foot, eyes and more. That model was trained on Common Objects in Context (**COCO**) Keypoints 2017 [Lin *et al.*, 2014] dataset. It's provides two hundred thousand images labeled supporting 17 keypoints for human figures. One important resource for pose estimation is a skeleton. This shape composes structures of keypoints. Figure 2a shows COCO Dataset's Keypoints[4] configuration used in MoveNet and PoseNet. Figure 2b shows an example of YOLOv8 pose prediction.



(a) COCO skeleton keypoints. Adapted from: Rahman *et al.* [2025].



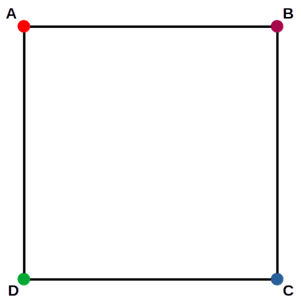
(b) YOLOv8 human pose detection using COCO skeleton keypoints.

Figure 2. Human pose detection baseline: COCO skeleton structure and YOLOv8 detection results used as reference for chess piece keypoint modeling.

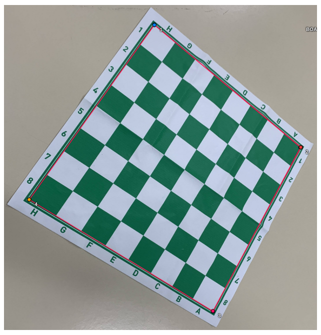
For this study, YOLOv8 large pose model was used as the DL model to detect both board and pieces in a chess game.

One of the main objects that needed to be detected was a chessboard. Since it is a square shaped object, at least four points is required to mark its boundaries. A way of changing some model's parameters is provided by the current available YOLOv8's API (Application Programming Interface). Keypoints' shape is one of the parameters that can be tweaked. Therefore, instead of predicting 17 keypoints, the model was customized in this work to predict only 4.

Because four points were needed to annotate a board, each piece also needed to be annotated using 4 keypoints (Figure 3). Three points were used to mark the base of the piece in a triangle-shaped polygon and one point was used to annotate the upper part of the piece. Three points for the base were utilized so the position of the piece in the board could be better triangulated (Figure 4).



(a) Board skeleton keypoints.



(b) Board image annotated.

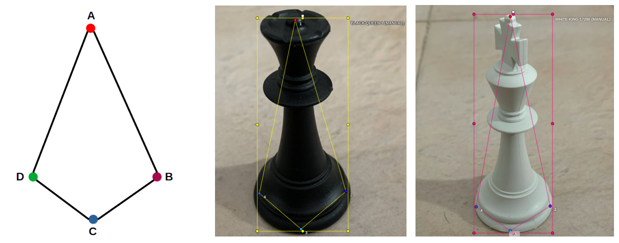
Figure 3. Customized board skeleton and annotation sample.

2.3 Board segmentation

Once the model was trained, board and pieces could be detected. Some processing is performed to extract relevant information from an image. Processing pipeline for detecting

[3]Ultralytics YOLO - the latest in real-time object detection and image segmentation. Available in: `https://github.com/ultralytics/ultralytics/blob/main/docs/en/models/yolov8.md`.

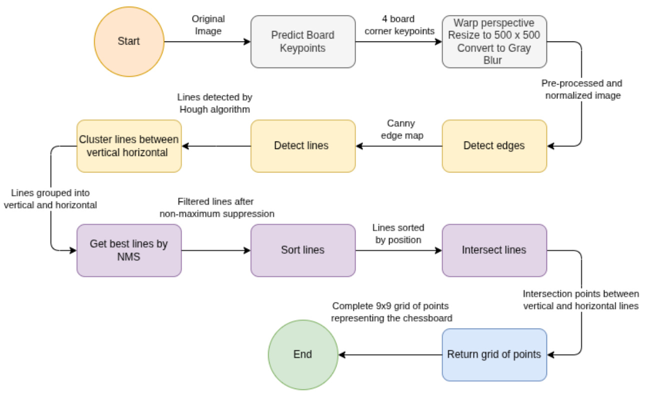
[4]COCO Keypoints. Available in: `https://github.com/tensorflow/tfjs-models/tree/master/pose-detection`.



(a) Piece skeleton keypoints. **(b)** Black piece annotated. **(c)** White piece annotated.

Figure 4. Customized piece skeleton and annotations examples.

and processing the board is shown in Figure 5. By using the 4 detected keypoints from the chessboard, as shown in Figure 6, a warp perspective transform was applied so the board is completely visible on screen as in Figure 7.



Start
Original Image
Predict Board Keypoints
4 board corner keypoints
Warp perspective Resize to 500 x 500 Convert to Gray Blur
Pre-processed and normalized image
Detect edges
Canny edge map
Detect lines
Lines detected by Hough algorithm
Cluster lines between vertical horizontal
Lines grouped into vertical and horizontal
Get best lines by NMS
Filtered lines after non-maximum suppression
Sort lines
Lines sorted by position
Intersect lines
Intersection points between vertical and horizontal lines
Return grid of points
Complete 9x9 grid of points representing the chessboard
End

Figure 5. Main pipeline processing for board and grid automatic detection.

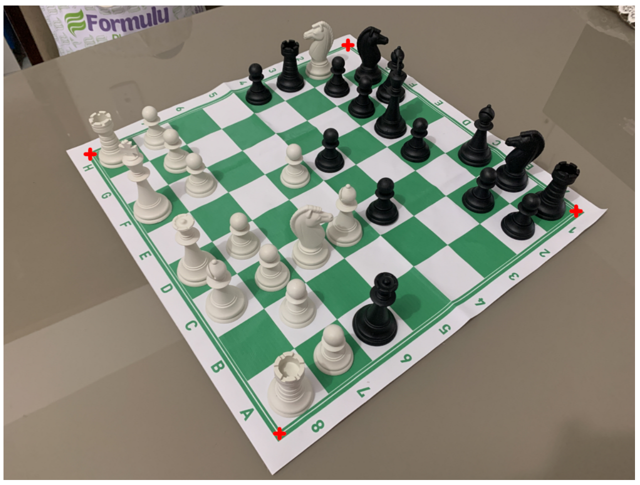


Figure 6. Board's keypoints automatic detection (Red cross dots), using your custom YOLOv8 model trained.

The board needed to be segmented into small squared pieces. Board edges were detected by using Canny edges (Figure 8a) and then the board squares were detected by applying HT (Figure 8b) ([Gonzalez and Woods, 2017]).

After all lines were detected, agglomerative clustering was applied to split lines into two groups: horizontal and vertical lines, by using the angle difference between the lines. Agglomerative clustering starts by placing each input in its own cluster and then begins merging clusters while trying to minimize computational cost [Ackermann *et al.*, 2014]. Figure 9 shows clustered lines based on their angles, red lines are horizontal and blue lines are vertical.

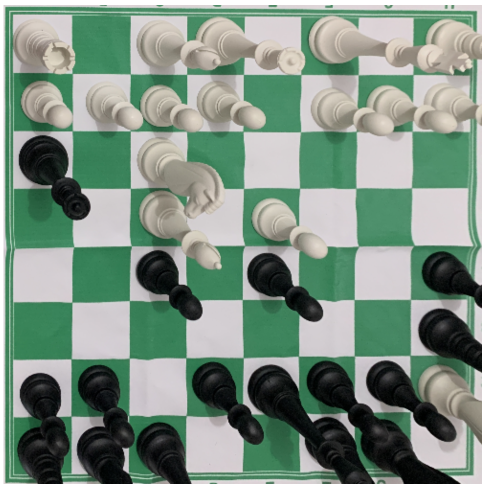
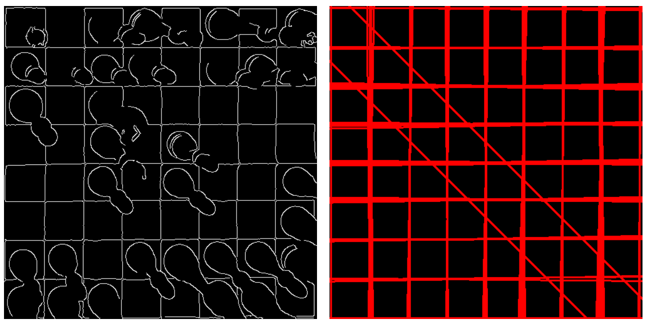


Figure 7. Image resized to 500×500 px and perspective warped.



(a) Canny edges applied to blurred image. **(b)** HT applied to edges image.

Figure 8. Canny edges and HT applied to blurred image.

Lines that are too close to each other are detected by HT, which caused some noise to the result lines (Figure 9). These lines were removed and only the better ones were kept by applying a non-maximum suppression technique. To select the better lines, the distance among the lines is calculated. Whenever they are too close, which line got more accumulation from HT accumulation algorithm was checked, then the line with more accumulation was kept over the other one.

Lines that are not as vertical and not as horizontal as desired are presented in Figure 9. These lines were filtered out by calculating the gradient on the distances among the lines and a reference point. The intersection point between each vertical line to a horizontal reference and the intersection point between each horizontal line to a vertical reference were obtained, as shown in Figure 10. After these intersections are obtained, lines are sorted based on the distance between one intersection point and a reference point. For vertical lines, the reference point is the point in the middle of the left side of the screen, and for horizontal lines, it is the point in the middle of the upper part of the screen. Once all lines are sorted, the gradient between the lines is calculated, and whenever intersections are too close compared to the average distance, that line is removed. Final filtering result can be seen in Figure 11.

With all horizontal and vertical lines sorted, horizontal lines could be intersected with vertical ones to get their intersection (9×9 points) to form a grid. By connecting each point with its neighbor, a grid of squares that represent each square of the board was able to be made, as purple in Figure 11.

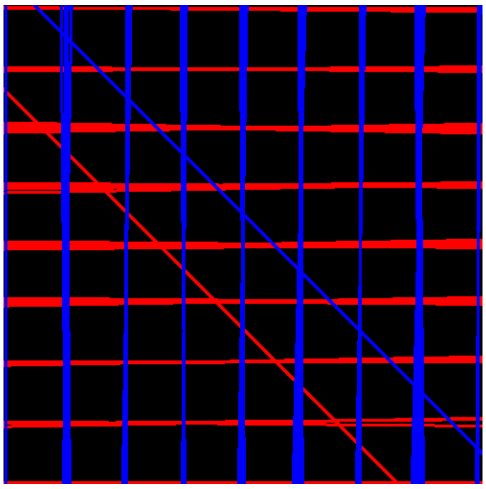


Figure 9. Vertical and horizontal lines clustered.

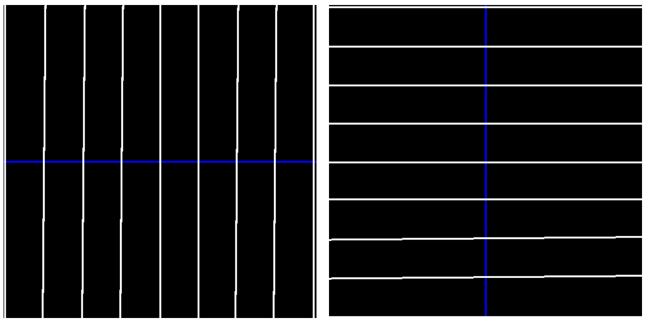

(a) Vertical lines and their reference. **(b)** Horizontal lines and their reference.

Figure 10. Vertical and horizontal lines identified by HT with reference points for line validation and filtering.

The grid points after the segmentation was finished are shown in Figure 12. The perspective was undone by using the same matrix used initially so the squares now lay over the board on the original image.

2.4 Filtering algorithms

After all processing steps were completed, the pipeline for detecting and splitting the chessboard was validated using the same dataset employed during model training. At first, the results were not satisfactory. Upon inspection of the images where the grid was not properly generated, it was observed that the issue occurred when keypoints detected by the model failed to accurately define the boundaries of the board. When the detected points position the board's boundaries inward-cutting out a piece of the chessboard—spatial information was lost, resulting in poorly segmentation or preventing segmentation. An example of this issue is illustrated in Figure 13b.

To overcome this, a geometric vector-based post-processing step was introduced. Four directional vectors were built where their origins were defined in the center of the board and pointing toward the points detected by the model. These vectors were then scaled by a small factor of 1%, expanding the board's region and attempting segmentation again. An illustration of this approach is shown in Figure 14. This process was repeated a few times in an effort to improve segmentation. The ideal result would be the same as shown in Figure 13a, which could not be achieved by only scaling the vectors; however, the result was capable of being improved. The results of this process are discussed in Section 3.

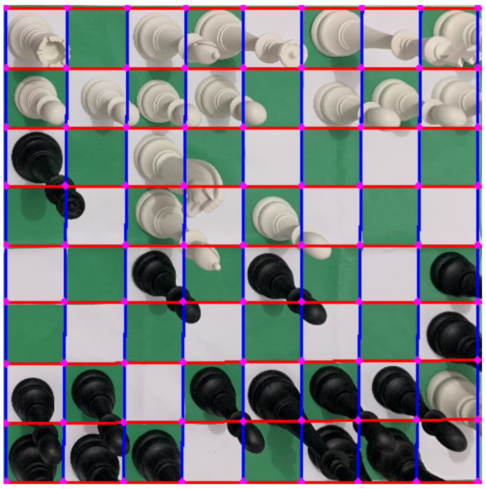


Figure 11. Grid over resized image.

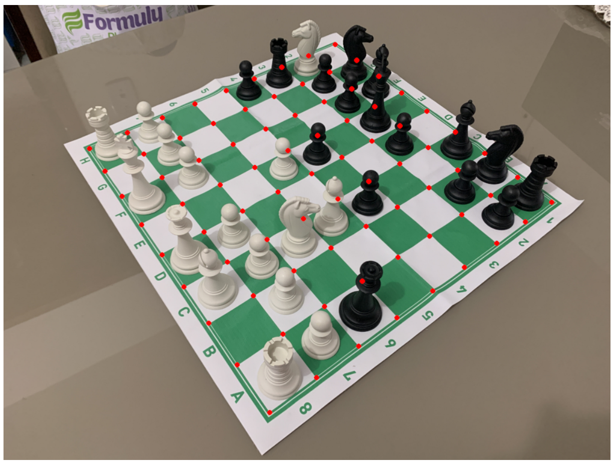


Figure 12. Intersection points over original image (Red dots).

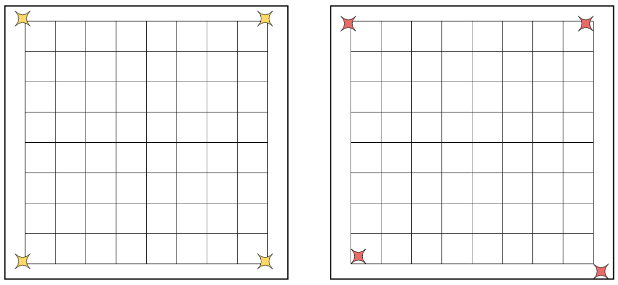

(a) Expected keypoints detection (Yellow marks). **(b)** Actual keypoints detection (Red marks).

Figure 13. Expected vs actual detection for keypoints.

2.5 Piece positioning

With the board's grid built, the information from the piece's keypoints was used to define what square board it is over. As discussed previously, 3 piece's keypoints were shaped as a triangle, which was used to get its centroid, that is the geometric center of the triangle and is defined by the mean position of all three points [Pedoe, 1970].

Given that the points P, Q and R forms the triangle ΔPQR, the three medians of a triangle passes to the same point C. Thus, triangle centroid C can be defined as in Equation 1.

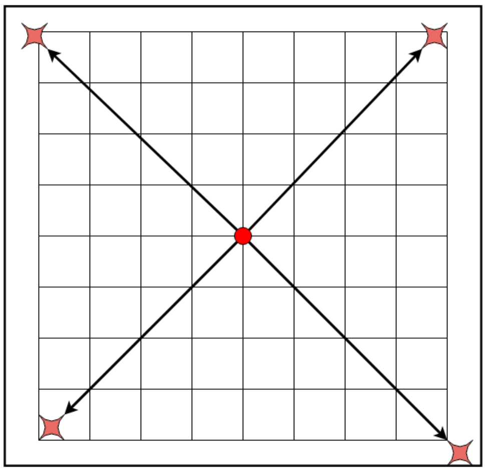


Figure 14. Vectors built from board keypoints, red circle marks center of board.

$$C = \frac{P+Q+R}{3} \tag{1}$$

Each point can be expressed in terms of x and y.

$$\begin{aligned} P &= (x_1, y_1) \\ Q &= (x_2, y_2) \\ R &= (x_3, y_3) \end{aligned} \tag{2}$$

Substituting, Equation (2) in (1):

$$\begin{aligned} C_x &= \frac{x_1+x_2+x_3}{3} \\ C_y &= \frac{y_1+y_2+y_3}{3} \end{aligned} \tag{3}$$

After all centroids are calculated, each board square is iterated over to check what piece is over it. To do so, it is verified if the piece's centroid is inside the board square polygon. Whenever this happens, it means that piece is inside the square.

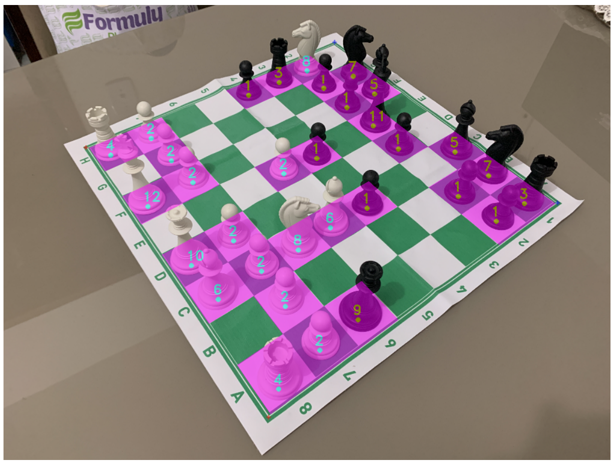


Figure 15. Custom YOLOv8 model trained in this work demonstrating automatic piece classification (index numbers correspond to Table 1) and detection (yellow for black pieces, cyan for white pieces) with occupied squares marked in purple.

Figure 15 illustrates the final output of the piece positioning system, where the custom-trained YOLOv8 model successfully identifies and classifies each chess piece on the board. The numerical labels displayed above each piece correspond to the class indices defined in Table 1, enabling precise piece identification. For demonstration purposes of the system's detection capabilities, a color-coded visualization scheme is applied: yellow points mark the centroids of detected black pieces, cyan points represent the centroids of white pieces, and occupied board squares are highlighted with a purple overlay. These centroid points, calculated from the triangular base formed by three keypoints of each piece, serve as the precise reference for positioning pieces within their respective board squares. This comprehensive visual representation demonstrates how the system processes and understands the current game state and the spatial distribution of pieces across the chessboard grid.

Once all the relevant information is processed, some feedback over the image and their actions can be given to the user.

2.6 Movement proposal

Our last goal is to propose to the user, based on current state of game, movements that can be made with a certain piece over the board. Movement proposal was based on matrix observation and some calculations applied into board's state, Figure 16 shows a representation of board's matrix.

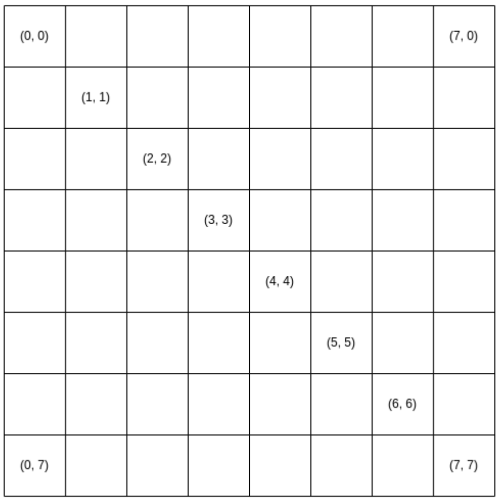


Figure 16. Board matrix representation.

Whenever the user navigates through the board, a group of algorithms are applied so proposal can be calculated and shown to the user. Movements characteristic of each piece were analyzed, and algorithm development was split to cover more ground. Queen's movement was analyzed, verifying that it's a combination of Rook's and Bishop's movements as shown in Figure 17.

With this, their movements were able to be separated into two generic movements: side and diagonal movements. Given that a piece is located in position (x, y), where x and y are the indexes where the piece is located on the matrix, its possible positions in those directions can be identified. If the piece is a Rook, side movement logic needs to be applied; if that is a bishop, diagonal movement logic is applied; and if that is a queen, both logics are applied, side and diagonal.

For the side movement, one axis at a time is incremented and decremented while the other axis is kept fixed. If a piece

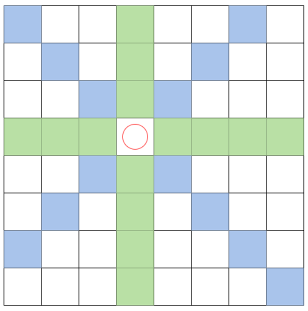


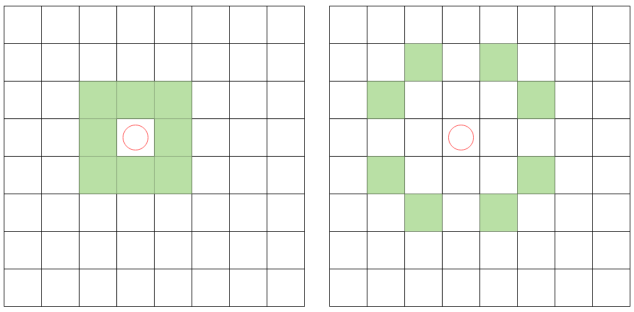
Figure 17. Queen movements are a combination of Rook (green's) and Bishop's (blue's). Red circle is piece position.

is located in $(4, 4)$, x is incremented all the way up until $x = 7$ is reached, then x is decremented all the way down until $x = 0$ is reached, then this process is repeated for y. By doing this, all squares in those directions are visited.

Diagonal movement follows almost the same steps. Here, x and y are incremented at the same time, then both of them are decremented together, then one is incremented and the other is decremented, and lastly one is decremented while the other is incremented. With this logic, the diagonals of that piece are walked towards. The increment is stopped when $x = 7$ or $y = 7$ or when $x = 0$ or $y = 0$.

Both movement strategies stop when iteration finds a piece of same color, because a piece cannot go over another piece. Note that iteration also stops when a piece of different color is found; the difference is that the located piece is marked to be captured as it is an enemy piece.

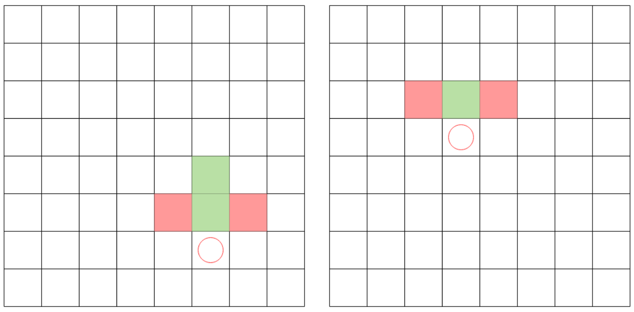
Last three pieces have unique movements: King can only walk one adjacent square at a time, its movements are shown in Figure 18a; Pawn can only walk forward two squares during its initial move, as in Figure 19a, and one square otherwise, as in Figure 19b, and captures pieces that are one adjacent square diagonally in front of it; Knight walks and captures in a specific way—it can go two ahead and one to the side, in any direction. This movement follows an L-shaped pattern, so Knight's movements are not obstructed if there is a piece between the starting and ending positions. How Knight moves is shown in Figure 18b.



(a) King. **(b)** Knight.

Figure 18. Green's represent possible movements. Red circle is piece position.

All movement proposals have the same final condition and feedback logic. If there is no other piece in a board square this piece can walk to, it is marked so user know it can move there and if there is an enemy piece, the square is marked in red so user know it can capture that piece. Algorithm 1 describes the movement's flow.



(a) First piece move. **(b)** Not in initial position.

Figure 19. Pawn movement from each player's perspective. Green squares indicate possible movement directions. Red squares denote enemy capture positions. Red circle marks the current piece position.

Algorithm 1 Chess Movement Analysis for AR.

```
function AnalyzeChessMoves(b, p)
    piece, color ← GetPieceInfo(b, p)
    m ← EmptySet()
    if ValidPos(p) then
        if piece ∈ {ROOK, BISHOP, QUEEN} then
            m ← GetLinearMoves(b, p, piece, color)
        else if piece ∈ {KNIGHT, KING} then
            m ← GetPatternMoves(b, p, piece, color)
        else if piece = PAWN then
            m ← GetPawnMoves(b, p, color)
        end if
        HighlightAR(m)
    end if
    return m
end function
function GetLinearMoves(b, p, piece, c)
    m ← EmptySet(); dirs ← GetDirs(piece)
    for dir ∈ dirs do
        pos ← p
        while true do
            pos ← Next(pos, dir)
            if !ValidPos(pos) or OwnPiece(b, pos, c) then
                break
            end if
            m.Add((pos, EnemyPiece(b, pos, c) ? CAPT : MOVE))
            if !Empty(b, pos) then
                break
            end if
        end while
    end for
    return m
end function
function GetPawnMoves(b, p, c)
    m ← EmptySet(); x, y ← p.x, p.y
    f ← c = WHITE ? -1 : 1
    if Empty(b, x, y+f) then
        m.Add((x, y+f, MOVE))
        if InitRank(y, c) and Empty(b, x, y+2*f) then
            m.Add((x, y+2*f, MOVE))
        end if
    end if
    for dx ∈ {-1, 1} do
        if EnemyPiece(b, x+dx, y+f, c) then
            m.Add((x+dx, y+f, CAPT))
        end if
    end for
    return m
end function
```

3 Results

This section describes the results obtained throughout this work.

The problem was split into four well-defined steps. A DL model was trained from YOLOv8l-pose to detect keypoints for chessboard and pieces, by adapting a human pose detection model to predict four points of the corner from chessboard and for each piece.

3.1 Model training

The training process was made using the model **YOLOv8l-pose**[5] (YOLO from Ultralytics version 8, for pose detection and large model) on NVIDIA TESLA P100 available freely on Kaggle[6].

[5]YOLOv8 Large Pose Detection. Available in: `https://github.com/ultralytics/assets/releases/download/v8.3.0/yolov8l-pose.pt`.

[6]Kaggle: Platform for data science and machine learning competitions. Available in: `https://www.kaggle.com`.

A Python[7] script was used to configure YOLOv8 Standard Development Kit (**SDK**) for training. Main configuration data file was setup to use 4 keypoints instead of 17, as described in Section 2.2.

YOLOv8 SDK provides built-in augmentation during the training phase. However, this feature was disabled to prevent doubly applied transformations.

Image size hyperparameter was set to $640 \times 640\ px$. Epochs was increased to 150 instead of the default value of 100, due to the size of the dataset. The system achieved learning convergence with this value.

The following hyperparameters were used during training: batch size (16), learning rate (0.000588), and AdamW optimizer. Loss function weights were set as classification loss (0.5), distribution focal loss (1.5), keypoint loss (3.0), pose loss (24.0), and box loss (6.5). A considerable increase in pose loss - from the default value of 12.0 to 24.0 - was employed with an idea of influencing the model to predict keypoints position more accurately, a small tweak in keypoint and box losses was also performed, the first being increased by 1.0 and the later reduced by 1.0. All other hyperparameters retained their default values.

Some hyperparameters such as batch size were changed trying to improve training speed. Unfortunately when batch size increases, memory needed for training increase as well. Because this fact batch size was kept low to make it possible to train using GPU's (Graphic Processor Unit) available memory.

The last five hyperparameters are model's losses gain, i.e., when model calculate the error in comparison to ground truth, errors are multiplied by a gain to increase the importance of that loss, and therefore, pose loss has increased importance over other losses. The objective is minimize pose estimation loss over box estimation, for example.

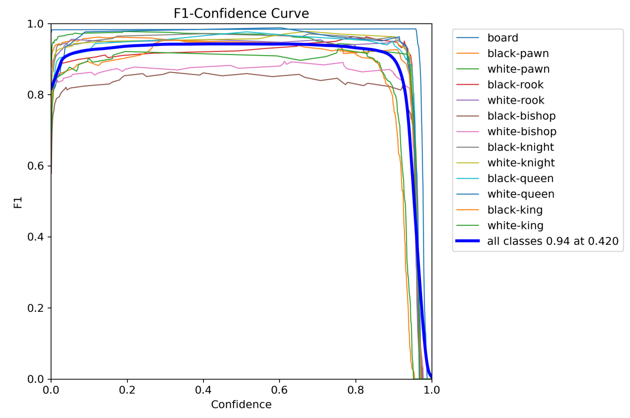
A pretrained version of this model was not used, because it did not learn very well. In fact, training was early stopped because results were not being improved. It is supposed that this happened because the model was trained to estimate human pose and when applied to a non-human estimation, poor results were produced.

3.2 Training and validation results

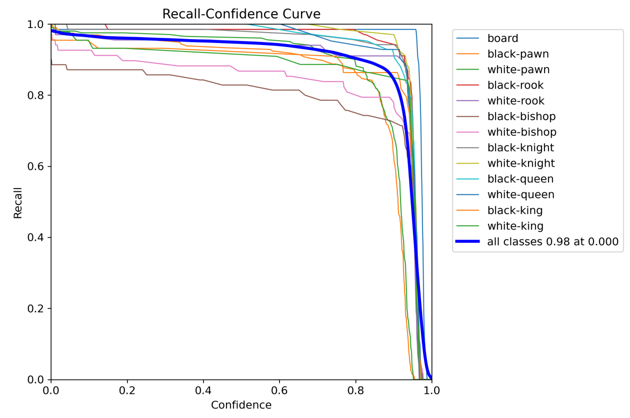
As shown in Figure 20, our model maintains high performance across a wide range of confidence thresholds. The F1 score remains stable and above 0.9 for most classes until approximately 0.9 confidence, after which it drops sharply, achieving a peak F1 score of 0.94 at a confidence threshold of 0.42. Similarly, the Recall-Confidence curve shows that the model reaches a recall of 0.98 at a confidence threshold of 0.00, and decline gradually as the threshold increases. These results suggest our model is capable of precisely detect keypoints and classes, even at low confidence thresholds. The average result for each class and for the entire model is shown in Table 2.

The YOLOv8l-pose model chosen for keypoint detection was a suitable option due to its precision of prediction. The model was able to achieve 95% overall pose estimation precision, where the detection of keypoints on the board

[7]Python Computer Language. Available in: `https://www.python.org/`.



(a) F1-Confidence Curve.



(b) Recall-Confidence Curve.

Figure 20. Performance curves showing F1-score and Recall metrics across different confidence thresholds for chess piece and chessboard keypoint detection.

achieved 99% accuracy as shown in Table 2. All classes of pieces achieved a precision greater than 92%, the only piece that did not achieve a good precision compared to its partners was the black Rook that achieved only 89%.

Table 2. Model validation results.

Class	Precision	Recall	mAP50	mAP50-95
Board	0.993	0.995	0.992	0.991
Black Pawn	0.945	0.968	0.975	0.957
White Pawn	0.976	0.966	0.992	0.985
Black Rook	0.894	0.955	0.970	0.940
White Rook	0.970	0.961	0.987	0.975
Black Bishop	0.935	0.829	0.908	0.892
White Bishop	0.924	0.893	0.943	0.889
Black Knight	0.956	0.949	0.983	0.938
White Knight	0.957	0.996	0.976	0.953
Black Queen	0.954	1	0.993	0.965
White Queen	1	0.998	0.995	0.976
Black King	0.975	0.901	0.961	0.949
White King	0.991	0.909	0.969	0.947
All	**0.959**	**0.948**	**0.973**	**0.950**

In Figure 21, classes predicted by the model versus actual expected class can be seen. From the confusion matrix, it can be inferred that the model struggles to correctly detect white Bishop class. In addition to that, Figures 20a and 20b show that besides the low precision for the white Bishop

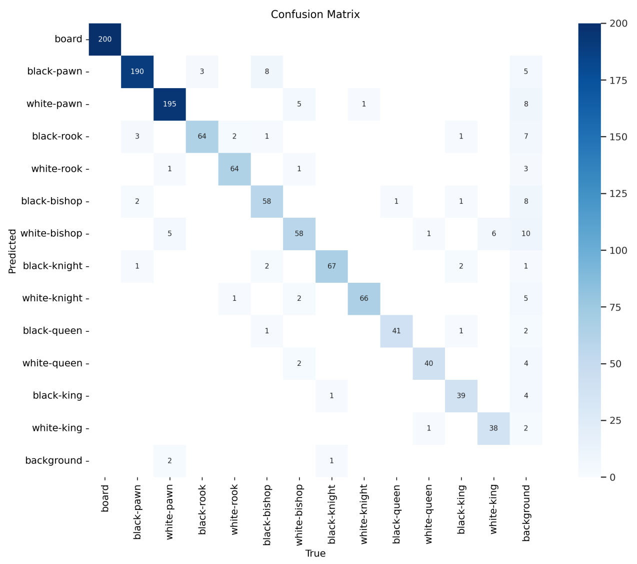


Figure 21. Model confusion matrix.

class, the model also struggles to correctly detect and classify the black Bishop class, indicating the problem can be due to the structure of the piece itself. Although the reason for the lower scores observed for these classes can not be precisely defined, variations in image capture conditions or the positioning of pieces on the board may have negatively impacted the model's performance for these classes.

Table 3. Comparison between piece classification algorithms. Where ML (Machine Learning), HO (Handles Occlusion) and MVA (Multi-View Acquisition).

Author	Year	ML	HO	MVA
Delgado	2019	Yes	Yes	No
Mehta	2020	Yes	Yes	No
Czyzewski	2020	Yes	-	Yes
Wölflein	2021	Yes	No	Yes
Menezes	2023	Yes	-	No
Masouris	2024	Yes	No	Yes
Ours	**2025**	**Yes**	**Yes**	**Yes**

3.3 Board segmentation results

Second step was segmenting the board into a grid of squares where each piece could be located in. By using a set of well known CV algorithms such as Canny edges and HT, all lines present in the board were found, and a clustering algorithm, agglomerative clustering, was applied to split these lines into two groups: vertical and horizontal. Similar lines were removed by applying a non-maximum suppression based on proximity between lines and keeping the best lines. Lastly, each vertical lines was intersected with horizontal ones to build a grid of squares.

The accuracy of chessboard segmentation was validated by using the same dataset used to validate the model, all images in validation dataset have exactly one board visible. As discussed in Section 2.4, filtering algorithms were added to improve the segmentation step due to some detection problems of the model. Initially, when validating with no additional processing, segmentation worked only in 9% of the boards in the dataset, Table 4 shows the results for each additional processing applied to the segmentation pipeline. The board segmentation result was improved considerably from 9% to 98% by mixing board keypoints' vectors and removing lines too close based on gradient from lines distances.

Table 4. Board segmentation results.

Additional processing	Accuracy
No additional processing	9.0%
Scaling board keypoints	82.5%
Removing close lines	18.0%
Combining scaling and removing	98.5%
Limiting scaling iteration to 20 steps	98.0%

3.4 Piece positioning

As third step, each detected piece was positioned in its square based on piece's centroid, the mean point based on all three detected points from piece's base. By using the calculated grid in previous step and piece's centroid, the square over which a piece is located could be determined by calculating if its centroid is inside the board square's polygon. To evaluate piece positioning step, it was verified if there was any duplicate positioning on same square. An error rate of 18.19% was achieved when positioning pieces, this gives an error rate per-square of 1.45%. Although the error seems small it directly affects how pieces are positioned over the board, potentially leading to incorrect or even invalid game states, affecting system's quality.

3.5 Movement proposal

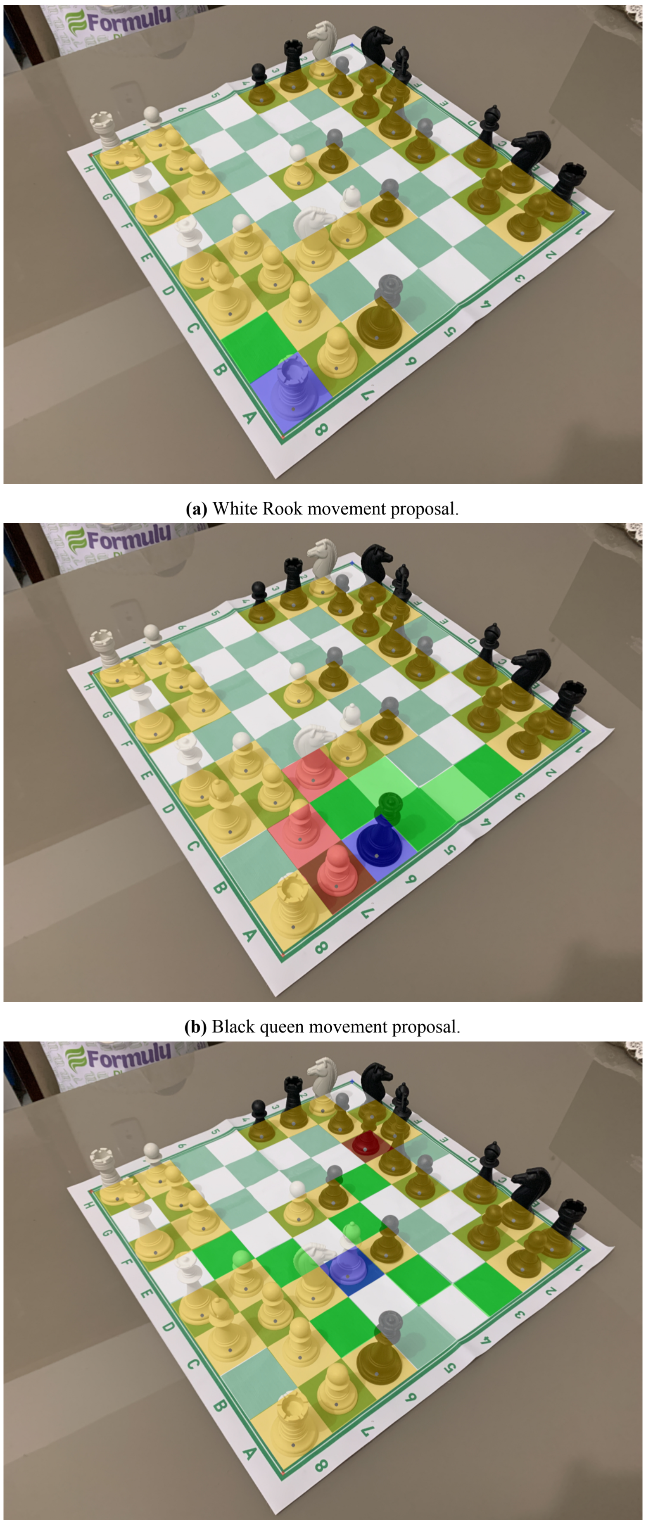
By considering the board as a 8×8 matrix, mathematical manipulations can be made to achieve a specific result, each square is a position in the matrix which enables easily navigation between each square. Piece's position is associated to a square in the board grid, so the selected piece can be identified and some operations can be executed.

As our last step, overlays of information were drawn over the original image so user can get insightful proposal of movements as discussed in Section 2.6. The user can select any piece on the board, with the selected piece being identified by a blue square. Yellow squares represent all pieces automatically detected by the system. After selection, the system displays dark green squares showing all possible movements for that piece. Red squares represent opponent's pieces that the selected piece can capture.

Figure 22 shows some examples of movement proposal overlay. In Figure 22a, the available movements of a white Rook are shown. Figure 22b displays the selection of a black queen with its valid movements in dark green squares and red squares indicating the opportunity of capturing one white Knight and two white Pawns. Figure 22c shows the selection of a white Bishop, its permitted movements in green squares, and the potential for capturing an opponent's black Pawn indicated by a red square.

3.6 Processing time

A performance evaluation was performed over the processing steps from detection to segmentation, currently, this application supports only images, but it is important to identify the time to process one image since it could be evolved to



(a) White Rook movement proposal.

(b) Black queen movement proposal.

(c) White Bishop movement proposal.

Figure 22. Movement proposal results.

a video based application. This performance evaluation was performed on a notebook with NVIDIA GTX1650 graphic card, this information is important since model's predictions are directly impacted by availability of a GPU, Table 5 lists inference time from model and board segmentation time, piece localization takes approximately 1 ms, that is why it is included within board segmentation.

Table 5. Average processing time.

Step	Time
Custom YOLOv8 inference	67.3 *ms*
Board/piece localization algorithm	386.6 *ms*
Total	**454.0 *ms***

4 Conclusion

In this work, the development of a system that used DL, CV and AR to locate board and pieces of a chess game, position each piece in its board square and propose movements to a beginner-level chess player, aiding them to improve their playing skills, was discussed. The DL model is capable of precisely predicting 99% of the boards, it has a 95% overall precision for board and pieces classification, board segmentation achieved 98% precision and reached an error of 18.19% for piece positioning, which results in an error of 1.45% per board square. Evaluation was performed over the validation dataset composed of 200 images.

Although this system was not evaluated on videos, an assessment of its performance over one single image was performed, which achieved, on average, 454 ms per image, making it not a suitable solution for real time analysis, which would require an average time of 41.7 ms per frame, to achieve at least 24 frames per second. From this evaluation, only 67.32 ms was taken by the YOLOv8l-pose model for inference, on GPU, which means the board localization is holding the performance of the system.

Most of related works are focused on helping already advanced chess players by detecting and logging game state into some database so it can be resumed later. A possible solution able to help beginner-level players to understand basics of a chess game such as movement rules is presented by this work.

4.1 Future works

One problem discussed earlier in this section is performance. This could be solved by adapting board localization algorithm, reducing keypoints filtering iterations or completely replacing this algorithm. A possible challenge, also aimed at solving the performance issue, would be trying board segmentation using an end-to-end approach based on DL, since a performance bottleneck is line detection and thus, board segmentation. This end-to-end system should be able to locate the board, all pieces and board lines. The challenge is building a dataset composed of board corners and lines, pieces classes and their positions on board.

A satisfaction assessment could be employed for users in order to provide an understanding of how useful this system could be for beginner players, helping demonstrate the development of such systems or applications could be beneficial for certain individuals.

Declarations

Authors' Contributions

CEG contributed to the conception and design of the study, guided key decisions throughout the project, and participated as one of the manuscript authors. AMSN conducted all experiments, served as the primary contributor, and drafted the majority of the manuscript.

All authors reviewed, edited, and approved the final version of the manuscript.

Competing interests

The authors declare that they have no competing interests.

Availability of data and materials

The dataset generated during the current study can be acquired upon this link address[8]. Source code from this work is publicly accessible at this repository[9].

Further relevant information

This work does not require ethics committee approval as it involves only technological development of chess recognition systems using DL and CV, with no human subjects, personal data collection, or experimental procedures that could pose risks to individuals.

References

Ackermann, M. R., Blömer, J., Kuntze, D., and Sohler, C. (2014). Analysis of agglomerative clustering. *Algorithmica*, 69(1):184–215. DOI: 10.1007/s00453-012-9717-4.

Billinghurst, M. (2002). Augmented reality in education. *New horizons for learning*, 12(5):1–5. Available at: `https://www.academia.edu/803776/Augmented_reality_in_education`.

Buslaev, A., Iglovikov, V. I., Khvedchenya, E., Parinov, A., Druzhinin, M., and Kalinin, A. A. (2020). Albumentations: Fast and flexible image augmentations. *Information*, 11(2). DOI: 10.3390/info11020125.

Czyzewski, M. A., Laskowski, A., and Wasik, S. (2020). Chessboard and chess piece recognition with the support of neural networks. *Foundations of Computing and Decision Sciences*, 45(4):257–280. DOI: 10.2478/fcds-2020-0014.

Delgado Neto, A. d. S. and Campello, R. M. (2019). Chess position identification using pieces classification based on synthetic images generation and deep neural network fine-tuning. *2019 21st Symposium on Virtual and Augmented Reality (SVR)*, pages 152–160. DOI: 10.1109/SVR.2019.00038.

Gobet, F. (2018). *The Psychology of Chess*. Routledge.

Gonzalez, R. C. and Woods, R. E. (2017). *Digital Image Processing*. Pearson, 4th edition.

Jankovic, A. and Novak, I. (2019). Chess as a powerful educational tool for successful people. In Tipurić, D. and Hruška, D., editors, *7th International OFEL Conference on Governance, Management and Entrepreneurship: Embracing Diversity in Organisations. April 5th - 6th, 2019, Dubrovnik, Croatia*, pages 425–441, Zagreb. Governance Research and Development Centre (CIRU). Available at: `https://hdl.handle.net/10419/196101`.

Joseph, E. and Easvaradoss, V. (2021). Thinking out of the box, enhancing creativity and divergent thinking through chess training. *Revista Mundi Engenharia, Tecnologia e Gestão (ISSN: 2525-4782)*, 6. DOI: 10.21575/25254782rmetg2021vol6n11525.

Kesim, M. and Ozarslan, Y. (2012). Augmented reality in education: Current technologies and the potential for education. *Procedia - Social and Behavioral Sciences*, 47:297–302. DOI: 10.1016/j.sbspro.2012.06.654.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014). Microsoft COCO: Common Objects in Context. In *Computer Vision – ECCV 2014*, pages 740–755, Cham. Springer International Publishing. DOI: 10.1007/978-3-319-10602-1_48.

Liu, W. and et al. (2016). Ssd: Single shot multibox detector. *Computer Vision–ECCV 2016: 14th European Conference*, pages 21–37. DOI: 10.48550/arXiv.1512.02325.

Masouris, A. and van Gemert, J. (2024). End-to-end chess recognition. In *Proceedings of the 19th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*. SCITEPRESS - Science and Technology Publications. DOI: 10.5220/0012370200003660.

Mehta, A. (2020). Augmented reality chess analyzer (archessanalyzer): In-device inference of physical chess game positions through board segmentation and piece recognition using convolutional neural network. *arXiv preprint arXiv:2009.01649*. DOI: 10.48550/arXiv.2009.01649.

Menezes, R. and Maia, H. (2023). An intelligent chess piece detection tool. *Anais do L Seminário Integrado de Software e Hardware*, pages 60–70. DOI: 10.5753/semish.2023.229800.

Orémaš, Z. (2018). Chess position recognition from a photo. Master's thesis, Masaryk University, Faculty of Informatics, Brno, Czech Republic. Available at: `https://is.muni.cz/th/meean/?lang=en`.

Pedoe, D. (1970). *Geometry: A Comprehensive Course*. Dover Publications.

Rahman, N. N., Mahi, A. B. S., Mistry, D., Masud, S. M. R. A., Saha, A. K., Rahman, R., and Islam, M. R. (2025). Fallvision: A benchmark video dataset for fall detection. *Data in Brief*, 59:111440. DOI: 10.1016/j.dib.2025.111440.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788. DOI: 10.1109/CVPR.2016.91.

Ren, S. and et al. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28. DOI: 10.48550/arXiv.1506.01497.

Unterrainer, J. and et al. (2006). Planning abilities and chess: a comparison of chess and non chess players on the tower of london task. *British journal of psychology*. DOI: 10.1348/000712605X71407.

Wölflein, G. and Arandjelović, O. (2021). Determining chess game state from an image. *Journal of Imaging*, 7(6):94. DOI: 10.3390/jimaging7060094.

Yusof, C. S., Low, T. S., Ismail, A. W., and Sunar, M. S. (2019). Collaborative augmented reality for chess game in handheld devices. *2019 IEEE Conference on Graphics and Media (GAME)*, pages 32–37. DOI: 10.1109/GAME47560.2019.8980979.

[8]Dataset. Available in: `https://kaggle.com/datasets/5d2e48b41c9c2811eeff436a177b26a36973e1f95674b5456b3309b0bf5a7b8b`.

[9]Source-Code. Available in: `https://github.com/cegoes-UESC/aidedchess`.