ChessNET: Computer Vision for Chess Board Recognition

Avi Balsam & Marcus Bluestone

Abstract

A central problem in computer vision involves mapping real-world simulations to virtual representations. In the last decade, this has seen applications within the competitive chess community, as efficiently automating the process of converting live games into digital boards for records/analysis would save players time. Many YOLO-adapted object recognition architectures have been proposed to solve this problem using as input a single image of the game. We improve upon previous architectures by utilizing a projection function that can address a chess board oriented in any direction, and by implementing a mask so classification occurs on individual patches rather than on the entire image. This change maintains the same accuracy levels as the state-of-the-art models, while running over sixty times faster. In addition, our approach allows for extremely quick data collection and annotation. Furthermore, we propose an extension of the model to efficiently analyze video stream inputs using a binary search algorithm. This would make analysis and recording of chess games significantly simpler, as users need only to capture one video instead of a collection of photos. In total, we propose a new tool called ChessNET, which recognizes video streams of chess games and is capable of efficiently outputting the sequence of moves that were played throughout the game.

Introduction

An essential component of competitive chess matches is a simple and efficient system for recording and analyzing games. In the past, much of this work was done manually both by the players and spectators. With the rise of computer vision systems in the past few decades, a variety of different efficient methods have been proposed to autonomously map physical chessboards to virtual representations. Once they are saved in a digital format, preexisting online software, such as the Stockfish engine, can be used for efficient analysis. [9] Such a tool could thus save players, spectators, and broadcasters significant time and energy.

Our precise goal is to take a video stream of a chess game as an input, and efficiently output the sequence of moves that were played throughout the game. There exist many methods for performing this mapping on single chess-board images; we improve on some of these architectures by developing a new projection function, implementing a novel masking technique, and extending the model to allow video streams as input. Our approach involves four steps: 1) finding the corners of the chessboard, 2) performing a perspective projection to map the chessboard into an aerial view 3) Dividing up the board into an 8x8 grid and re-orienting if necessary, and 4) applying image classification with an adapted ResNet-18 to each grid square. See Figure 1. We choose a CNN-based approach, as CNNs are well-known for achieving high accuracy results on many vision tasks [11], and more specifically, CNNs have proven to be very accurate in recognizing individual pictures of chess pieces [2]. We believe the same accuracy can be actualized on segmented images of a chessboard as well.

Because our model runs so efficiently (inference of a single image takes less than a second), we also propose a method to efficiently analyze an input video stream of chess-board images by binary searching through the frames of the chess game. We measure the precision and recall of the corner detector models and piece classifier separately. We measure the accuracy of the piece classifier model, including the confusion matrix for both piece type and color. This type of detailed analysis was not included in much of the past literature on this topic.

1. Related Work

1.1. Board Boundary Detection

Extensive work has been done on recognizing the precise location of the boundaries of chessboards, and they can be classified into two main approaches: recognizing lines and recognizing corners. The first approach recognizes all of the straight lines and lattice points on the 8x8 chessboard grid. Previous work has accurately found edges and lattice points of chessboards using object segmentation and vanishing lines. [2, 6, 8]. The second approach recognizes just the corners of the chessboard using a YOLO object detection architecture. Prior work has achieved accuracies near 99.5% using this simple technique [4, 7].

The advantage of the former approach is its specificity – if one can get access to the locations of the lattice points in the image, segmenting the board is a relatively easy task and doesn't require any projections. However, we favor the latter approach for two reasons. First, we believe that the simplicity of finding corners and performing a perspective projection outweighs the possible benefits of the former ap-

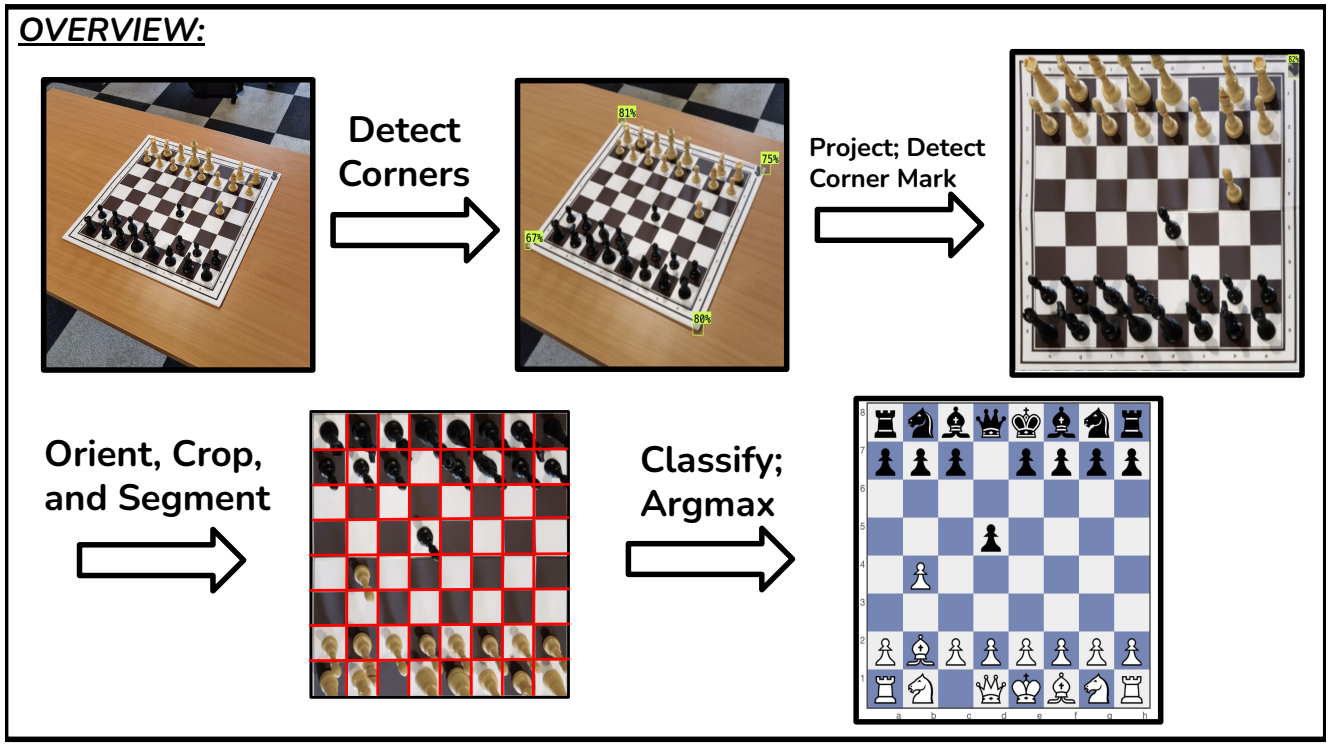

Figure 1. The pipeline for ChessNET. First, a YOLO v7 model detects the corners. Then, the chessboard is projected to occupy the entire image; and a corner marker is identified. Next, the image is re-oriented, cropped, and segmented accordingly. Finally, a masked classification is applied to each square in the grid. Piece type/color are chosen per square based on its maximum value in the output.

proach. This simplicity is crucial for our model since we plan to run it many times across a stream of video frames and not just a single image. Second, the latter approach is more resistant to occlusion of the chessboard, for example, by the hand of one of the players or by the pieces themselves. This is because the corner-based model only needs to find the four corners of the boards, not all of the edges and lattice points.

1.2. Piece Classification

Much work has also been done on classifying the chess pieces themselves. Ding et al. [3] trained binary SVMs for each chess piece, predicting both piece type and color with high success. The authors used two separate descriptors – scale-invariant feature transform (SIFT) and histogram of oriented gradients (HOG), and used these features as input to the SVM. This approach achieved on average 85% accuracy, though this accuracy is misleading since many pieces such as the bishop and knights, were classified with closer to 50% accuracy. It appears to us that the high overall accuracy was a product of the large number of "empty" squares in the test data set. In our experiments, on the other hand, we re-balanced the dataset and applied a CNN architecture, achieving higher accuracies for each piece. See Figure 3

Another approach, by Gallagher et al., applied object detection to find bounding boxes around each of the chess pieces. Then, he compared the intersection over union (IoU) values between the bounding boxes and the segmented board grid (found through a corner detection model and projection) to map pieces to their locations on the board [4]. We believe this approach to be sub-optimal for two reasons. First, occlusions and overlap may seriously affect IoU values, making this an unreliable way to map pieces to board locations. Gallagher et al. notes their piece classification model achieved high accuracies, but did not include any measure of accuracy regarding the mappings between pieces and board locations. After inspecting their model's predictions on around five images, we found many instances of poor predictions. Second, Gallagher's model required that pictures must be taken from the white-forward position. Rotated images could produce faulty results since the projection function that was used assumed a proper orientation.

A third approach by Masouris et al. avoided this issue by training a ResNet-50 model on the entire picture of the chess board and mapping it to a 384-dimensional vector corresponding to all possible choices of chess piece at each location on the board [1]. This allowed the model to interpret boards regardless of camera angle. Despite its high accuracies ($\approx 98\%$) in piece classification, we have three

with his approach. First, accuracy values per piece type are not listed in the paper. This means the model could be predicted extremely accurately for certain pieces (such as blank squares), while significantly lower worse on others. It is for this reason that our analysis includes a confusion matrix. Second, since this approach feeds the entire chess board image into the model without segmenting, we fear this approach may learn patterns between pieces and board orientations instead of explicitly classifying the board. For instance, it may implicitly take into account the probability of some board positions being played rather than simply reading the image. This may not always be desirable. Third, Masouris et al. used a ResNext-101 model, which when tested, took approximately one minute to analyze a single chess board image. This is much too slow for efficient analysis of a video input, and so we chose a much simpler ResNet-18 model for our approach. This gave us a 60x speed-up in inference time.

Another, more obscure, approach proposed using a 3D Stereo camera and a volumetric convolutional network to model the 3D nature of the chessboard [12]. While this achieved high accuracy results, we believe that setting up the 3D camera may be infeasible for large-scale chess tournaments and is perhaps unnecessary. A simple and efficient model is of the utmost importance for our specific problem, as we are applying the model across a large number of frames per video input. Moreover, because ChessNET has potential for real world deployment, a simple technical setup would be particularly important.

2. Proposed Method

Our approach, we believe, combines the best aspects of many of the aforementioned models. We use a YOLO model for corner-detection, followed by a prospective projection in order to align the image. We also take advantage of a marker in the bottom left corner to orient the image regardless of camera angle. Finally, we apply masked classification to each grid square instead of the entire board at once to allow for more efficient and accurate results (See Figure 1). We construct this new model and test the accuracies of the piece classifier as compared to the prior works done in this field.

2.1. Corner Detection & Projection

The first step in the pipeline detects the boundaries and lattice points of the chessboard. For this, we used RoboFlow's v7 YOLO model. Next, we used a perspective projection to map our original – possible skewed image – to a square image whose corners match perfectly with the chess board, interpolating the rest of the image to fit. The projection function takes as input the four detected corners and outputs the projected version of the image. To designate each corner as upper/lower and right/left, we divided them up using their medians in each dimension, thus guaranteeing a valid projection regardless of any initial camera orientation. Then, we used the cv2.PerspectiveTransformation function to warp the image.

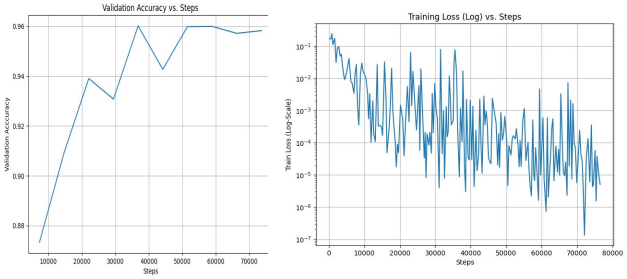

Figure 2. Plot of the validation accuracy and training loss (log scale) against the number of steps during the training and evaluating of the piece classifier model using the re-weighted distribution of samples. Note that training loss steadily decreases, while validation accuracy increases. This implies that we produced an accurate model that can generalize well to new examples.

Next, we applied another YOLO v7 object-detection model to locate the true bottom-left corner (which is marked with a unique identifier. See right-most image in Figure 1) and re-oriented/cropped the image accordingly. After this was done, we easily found the locations of lattice points on the chess grid by segmenting the board into 64 equally sized squares, since the locations of the lattice points on the chessboard should be approximately equally spaced throughout the board and parallel to the image axes.

2.2. Piece Classification

The next step in the pipeline detected the chess piece present in a single board cell. Applying this process to each of the 64 segmented squares from the previous step, we got a representation of the chessboard as an 8x8 array. We used a pre-trained Resnet-18 CNN architecture (additionally training on chess pieces) with a fully connected output layer with 13-dimensions, and soft-maxed the output. The output was a 13-dimensional vector, representing a one-hot for each possible piece (6 types, 2 colors, 1 blank). We measured the accuracy of our model by taking the NLL loss on each square.

2.3. Addressing Video Stream Input

We wanted our model to take video sequences, instead of images, as input. Thus, we also propose a method for binary searching through the frames to efficiently compile the list of moves played throughout the game. Specifically, at each iteration of our model, starting from the last frame processed, we look at the frame halfway between the current and last frames and run it through our model. We then use the chess engine Stockfish to determine whether this new

frame corresponds to a valid move from the last board state, whether the new frame is unreachable from the last board state in one move, or whether the new frame is the same as the last board state. Depending on which of these possibilities is true, we will recurse between our current frame and the future frame.

The following is an outline of our binary search algorithm, in pseudocode. Note that we assume every frame can be correctly processed by ChessNET. If this is not the case, we can remedy it by removing corrupted frames from the video when we encounter them and moving on to their immediate neighbors until we find a frame corresponding to a valid board state. In addition, if we make the assumption that the video camera is stationary, we can use the same corner values across all frames, thus significantly reducing the probability of a failed read. We will assume access to helper functions IS-VALID-MOVE(a, b) and IS-SAME-POSITION(a, b), which respectively determine whether there is a valid move taking the a-th frame to the b-th frame and whether the a-th frame and b-th frame represent the same chessboard position [9]. These functions have been implemented in chess engines such as stockfish, and are freely available online.

```
BINARY-SEARCH(i, l, u)
1  m = (l + u)/2
2  if IS-VALID-MOVE(i, m)
3       return m
4  elseif IS-SAME-POSITION(i, l)
5       return BINARY-SEARCH(i, m, u)
6  else
7       return BINARY-SEARCH(i, l, m)
```

Given a frame index i for a list of frames of length n, we can determine the frame index of the next move by running BINARY-SEARCH(i, i, n). This will allow us to make $O(\log n)$ passes through the network, a significant speedup compared to a naive linear pass.

3. Experimental Results & Discussion

3.1. Data Collection & Training

We used the Chess Recognition Dataset from 4TU.ResearchData [10], which consists of 100 chess games, each with an arbitrary number of moves and therefore images, amounting to a total of 10,800 images being collected. The pictures were taken from a variety of different angles and from different distances. In addition, the images were annotated in FEN Notation [5].

To train the corner detector models (both for their locations for the projection and for the unique marker in the bottom left corner), we used RoboFlow to annotate the dataset with bounding boxes. We trained on 60 images, resizing, applying grey-scale, and contrast-stretching, as prepossessing steps.

Next, for each image in the dataset, we used these two previous models to project, orient, and segmented the image into 64 pieces corresponding to the 8x8 grid. Before segmenting, we resized each image to 1024×1024 and normalized its color values across all channels. We parsed the annotations in conjunction with this segmentation, such that each image of the original dataset corresponded to 64 labeled images of the form (patch of original grid, piece label). If the corner detection models returned too many or too few corners, we discarded the data sample. We ended up with a dataset of 39,424 patches broken down by type as follows: P-3232, R-642, N-594, B-716, Q-236, K-616, p-3148, r-680, n-685, b-612, q-234, k-616, blank-27413, where letters are designated here using the previously described FEN notation. Because of the non-uniform distribution of the dataset, we re-weighted the sample distribution during training such that the probability of picking a specific type of piece corresponded to the average probability that that piece appears in a game. We trained the model for 10 epochs.

3.2. Results and Discussions

The corner-detector model obtained a precision of 90.5%, a recall value of 90.9%, and an accuracy of 89.7%. The detection of the unique identifier in the bottom left corner achieved a precision of 99.7%, a recall of 100%, and an accuracy of 99.5%. The high accuracies in these models despite the relatively small amount of training data point to the ease of generelizability of this approach. If implemented in the real world in competitive chess tournaments, it would not be difficult to add a small marker to the corners and train the models on approximately 60 or so images.

We plotted the training loss (log scale) and validation accuracy against the steps throughout the training process, reaching a final overall validation accuracy of 96% (see Figure 2). In addition, we analyzed the accuracies of the piece classifier module with respect to each piece (see Figure 3). This type of individualized analysis was left out from many prior works in this area, leading us to believe that other works may have achieved unbalanced results in their classifications [1, 4]. The classifier was almost perfect at detecting the color of the piece, and maintained a consistently high accuracy for all pieces. Interestingly, the model performed the worst in distinguishing between knights and queens. It is possible this confusion stemmed from a similar bumpy texture found on both of the pieces. Further work is required to understand why this mistake is occurring and how to fix it.

When bench-marked against past Piece Classification methods in the Related Works section (see (1.2) for their results), our models shows a competitive accuracy rate with

Figure 3. Accuracies of the ChessNET Piece Classifier module evaluated on a re-weighted training set of around 40,000 patches. Note, we split up the predictions into classes: color (left) and piece type (right). It is noteworthy to highlight the almost perfect predictions between black and white pieces and the consistently high accuracies for piece type.

an extremely fast runtime ($\approx$ 1 second per board). What makes our model especially efficient is the ability to parallelize the classification of each square in the chess board independently of the others. This speeds up the classification time by 64x, allowing for the efficient processing of high input video streams.

4. Conclusion

In sum, ChessNET produced extremely accurate and efficient results, achieving a 96% accuracy rate with an inference time of less than one second. These results are massive improvements compared to some of the past work done in this area. It was this efficiency in the model – combined with a clever Binary Search algorithm, and its ability to parallelized its classifications – that allowed the model to be extended to process video input streams. It is out hope that this model is adapted in the future and applied to real-world scenarios and competitive chess tournaments.

4.1. Individual Contributions

Avi and Marcus worked on data collection and annotation for the corner detection model together. Marcus implemented the perspective projection, as well as the segmentation of the board into an 8x8 grid. Marcus built build the chess piece detection model, and Avi fine-tuned its specifications. Avi prepocessed the data and trained the model, producing the images seen in Figure 2. Marcus built the confusion matrix, producing the images seen in Figure 3. Finally, Avi devised and implemented the video binary search.

References

[1] Jan van Gemert Athanasios Masouris. End-to-end chess recognition. 2023. https://doi.org/10.48550/arXiv.2310.04086.

[2] Manuel Prieto Mat´ıas David Mallas´en Quintana, Alberto Antonio del Barrio Garc´ıa. Livechess2fen: a framework for classifying chess pieces based on cnns. 2020.

[3] Jialin Ding. Chessvision: Chess board and piece recognition. 2016.

[4] James Gallagher. Represent chess boards digitally with computer vision. 2023.

[5] https://en.wikipedia.org/wiki/Forsyth

[6] J. A. Lay K. Y. Tam and D. Levy. Automatic grid segmentation of populated chessboard taken at a lower angle view. 2008.

[7] Can Koray and Emre Sumer. A computer vision system for chess game tracking. 2016.

[8] Artur Laskowski Maciej A. Czyzewski and Szymon Wasi. Chessboard and chess piece recognition with the support of neural networks. 2020.

[9] Alfredo Ibias Manuel Núñez Manuel Méndez, Miguel Benito-Parejo. Metamorphic testing of chess engines. 2023.

[10] Athanasios (2023): Chess Recognition Dataset (ChessReD). Version 2. 4TU.ResearchData. dataset. https://doi.org/10.4121/99b5c721-280b-450b-b058-b2900b69a90f.v2 Masouris.

[11] MDPI and S.; Bai Q.; Yang J.; Jiang S.; Miao-Y. Review of Image Classification Algorithms Based on Convolutional Neural Networks. Remote Sens. 2021 13 4712. https://doi.org/10.3390/rs13224712 ACS Style Chen, L.; Li.

[12] Hwann-Tzong Chen. CHESS RECOGNITION FROM A SINGLE DEPTH IMAGE. https://htchen.github.io/chess-recognition-single.pdf Yu-An Wei, Tzu-Wei Huang.