Chess Position (FEN) generation using Chessboard and Piece Recognition

Satya Biswal, Samruddhi Kahu, Matt Yang
Stanford University
{spbiswal, skahu, msryang}@stanford.edu

Abstract

Our work presents an end-to-end pipeline for automatically converting images of a physical chessboard into standardized chess notation (FEN). We first use a custom-trained YOLOv8 detector to locate and classify each piece. In parallel, we use the U-Net model to do segmentation and extract the corners of the chessboard. Once the corners are identified, we apply a perspective transformation using the detected corners and warp the board into a top-down view. Subsequently, by subdividing this image into an 8×8 grid, we can assign each detected piece to the correct square. In experiments, even though the individual YOLO and U-Net models achieve great results, the final ensemble model could only achieve 67% accuracy for the final FEN notation.

1. Introduction

Currently, chess enthusiasts and learners interact with chess positions in a variety of ways. For example, they might play on a physical board, read a chess book or magazine to follow a game, solve puzzles from printed diagrams, or view a picture shared on social media. In each case, if they want to analyze the position with a computer, they must manually recreate the position in an analysis program. This process is tedious and prone to errors. Our goal is to alleviate this burden by automating the translation of visual input into an FEN notation that can be directly fed into any chess engine or analysis software.

The motivation for this work stems from several observations. First, even experienced players often pause their workflow just to input a position to see engine evaluations. Second, learners working through puzzles or studies cannot immediately benefit from computer-generated annotations without first reconstructing the board by hand. Third, coaches and trainers lose valuable teaching moments when students struggle to transcribe positions. By automating piece detection and position recognition, we can provide immediate feedback. This approach helps players build intuition about piece placement on a physical board, which remains the standard format for most tournaments. It also reduces the risk of transcription mistakes that can lead to incorrect analyses.

Detecting chess pieces in a real-world scene or a printed diagram presents several technical challenges. Chess sets vary in style, and lighting conditions or shadows can obscure piece contours. A camera mounted to one side of the board will capture a perspective view rather than a perfectly top-down image, so we must correct for geometric distortions. Printed diagrams from books may suffer from scanning noise, low resolution, or handwritten annotations. To address these issues, we employ a YOLO based object detector that has been trained on a diverse set of real photographs showing different piece styles and board angles. After detecting pieces and classifying their types, we trained a U-Net model with manually annotated mask for detecting the board. We then used the predicted mask to detect the corner and then performed a perspective transformation to map each detected bounding box onto an 8×8 grid. And eventually used the mapping to extract FEN notation that represents the exact layout and piece position in the board.

Throughout this work, we combined several chess piece datasets, sourced from Kaggle [1, 2]. The dataset contains 265 training images, 64 validation images, and 33 test images, each annotated with bounding boxes for every piece type. These 265 images include 2894 labeled objects in total. All photographs were captured from a consistent viewpoint with a tripod positioned to the left of the board; however, to ensure accurate results, we made sure to include pictures taken of various kinds (wooden, plastic, paper) of boards. Some pictures also have noise augmentation applied. By training on this dataset, our model learns to recognize pieces even when they are partially occluded or when the board appears tilted. Examples of unlabeled images and their corresponding labeled versions can be seen in Figures 1, 2 (a) and (b).

The input to our algorithm is a single image or a video frame depicting a chessboard under typical lighting conditions. The input may come from a camera mounted on AR glasses, or a webcam mounted on a stand. We then use

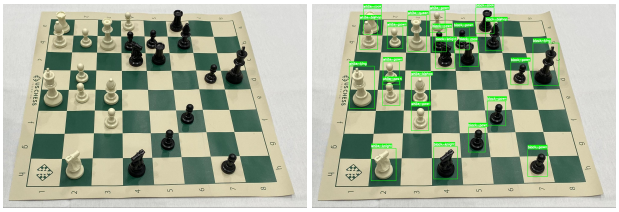

Figure 1. White and green chess board dataset (a) Unlabeled image (b) Labeled image

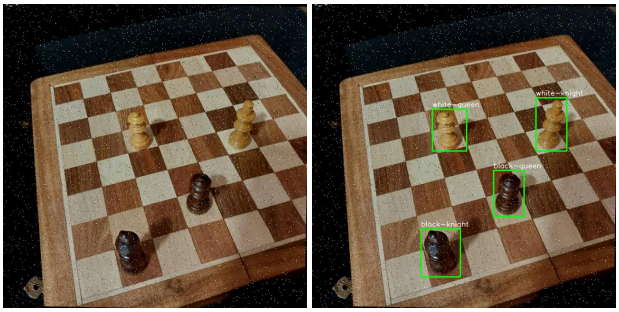

Figure 2. White and brown chess board dataset (a) Unlabeled image (b) Labeled image

a YOLO-based object detector to locate and classify each chess piece in the image. Once pieces are detected, we apply a UNet-based perspective-correction step that estimates the board's plane and maps detection coordinates onto a standard 8×8 grid.

Automating this detection and translation process has several important benefits. First, it greatly reduces the time required to go from seeing a position to analyzing it. Second, it minimizes human error in typing or placing pieces in a GUI. Third, it opens new possibilities for real-time coaching and augmented-reality overlays. For enthusiasts, we can show move recommendations. For learners, we can illustrate why certain moves work or fail. In a tournament setting, players can seamlessly verify that a physical board position matches the digital record. Audiences can also get live commentary about the moves based on engine analysis. Finally, this technology could be extended to digitize historical photographs or book diagrams, creating a searchable database of past games and studies.

To summarize, our work addresses the following problem: Given an image or video of a 3D or 2D chessboard, identify each piece's type and color and determine its location on an 8×8 grid. The input is a visual frame captured under natural conditions. We then use a YOLO for piece detection and a UNet-based geometric mapping step to assign each piece to the correct square. The output is an annotated board that is used to extract FEN notation. By automating this pipeline, we streamline the analysis process, reduce errors, and enable new modes of interactive, augmented learning and coaching.

2. Related Work

In our literature review, we have found methods to tackle this problem that fall predominantly in three categories: traditional computer vision and hybrid methods, CNN-based pipelines, and hybrid methods.

Early methods relied on edge detection, line finding, and template matching to locate the board and classify pieces. Czyżewski proposes a pipeline that first detects straight lines and lattice points to position the board, then identifies pieces using handcrafted rules [3]. Yang's "Building Chess ID" project uses Canny edge detection with a Hough transform to find grid lines, followed by template matching for each piece type [4]. Rizo-R's open-source "chess-cv" platform combines contour detection with template classification but assumes a nearly front-facing board and uniform set design [5]. All of these pipelines perform well in controlled environments but generalize poorly to pictures of chessboards taken from different angles or with different lighting conditions.

With the advent of deep learning-based methods, researchers began splitting detection into categories. Mehta's ARChessAnalyzer uses segmentation for board detection with an AlexNet CNN for piece classification [6]. Wölflein uses a RANSAC-based homography to warp the board and then applies two CNNs [7]. Conner and Talvi train a YOLO-based detector on 292 labeled images (2,894 objects) to enable real-time piece detection and classification [8]. Maia combines depth-based segmentation with a YOLOv4 model to identify squares and pieces [9]. These modular CNN pipelines achieve high accuracy under limited variation but require separate training for each component and depend on careful geometric preprocessing.

Finally, more recent end-to-end methods predict the entire 8×8 board configuration directly, avoiding explicit board segmentation. Masouris trains a single network to output a 64-cell grid of piece probabilities, achieving full-position recognition in 15.26% of test images. Chen uses a depth camera to capture segmented point clouds for each piece and feeds them into a CNN that classifies piece types and colors [10]. The caveat is that end-to-end models require large, diverse training datasets and still often lag behind deep learning based methods.

Overall, given the obsolescence of early methods and the limited effectiveness and complexity of end-to-end methods, we decided to stick with deep learning-based models and methods exposed in opencv library for our project.

3. Data

Our work leverages two distinct chess piece detection datasets sourced from Kaggle ([1] and [2]). Together, these

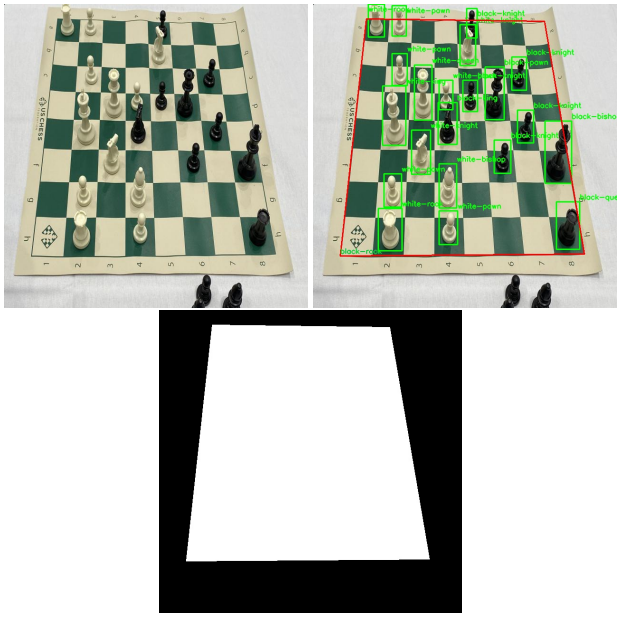

Figure 3. (a) Unlabeled chessboard image (b) Labeled chessboard and chess pieces (c) Mask for chessboard detection

datasets comprise 265 images for training, 64 for validation, and 33 for testing, each accompanied by detailed labels and bounding box coordinates for each of the 2894 labeled chess pieces. The images themselves feature two types of chessboards: one with classic green and white squares, and another with brown and white squares. All photographs maintain a consistent perspective, captured from a tripod positioned to the left of the chessboard. However, it's worth noting that the brown and white square dataset contains images affected by salt and pepper noise and sometimes presents only partial views of the chessboard. The annotations categorize individual chess pieces, ranging from "white-king" down to "black-pawn," with a total of 2894 labeled objects distributed across the 362 images. Visual examples of both raw and annotated chessboard images from both datasets are provided in Figures 1 and 2.

We employed various image augmentation techniques during the training of our YOLOv8 and Unet models. Specifically for chessboard detection with Unet, we meticulously hand-labeled the chessboard corners across both datasets using Roboflow's annotation tools. These corner annotations were then used to generate a binary mask, clearly segmenting the chessboard as foreground and other regions as background. Figure 3 (a), (b), and (c) demonstrate this process, showing an original image, its chessboard annotation shown in red color, and the derived binary mask. By pre-generating these binary masks for all training, validation, and test images and feeding only the image and mask file paths to our custom dataset and dataloader, we significantly reduced the computational overhead for mask

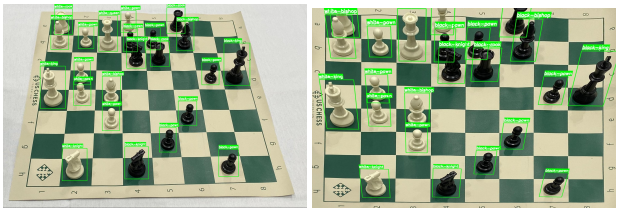

Figure 4. (a) Unprojected image (b) Projected image

generation during training, thereby considerably accelerating the entire process.

4. Methods

In this study, we introduce a pipeline for automated understanding of chess scenes. Our method begins with employing the state-of-the-art YOLO object detection model for the simultaneous detection and recognition of multiple chess pieces in an image followed by chess board (corners) detection using Unet. A perspective transformation will then be applied to project the detected chessboard into a canonical 2D plane, as shown in Figure 4. This process yields a 2D image with annotated chess piece locations. With the UNet-based determination of the chessboard corners and the subsequent projective transformation, we can then segment the board into its 64 constituent squares by uniform division of the image dimensions in Figure 4(b). Given a fixed board orientation, the identification of individual square labels is then straightforward.

As depicted in Figure 5, our algorithm prioritizes YOLO-based chess piece detection before identifying and projecting chessboard corners to a 2D plane. This order is intentionally chosen to prevent the potential loss of chess pieces located near the board's periphery, which might occur during a corner-based cropping and projection step.

4.1. YOLO based chess pieces recognition

In this section, we describe our chess-piece-detection method using YOLO. We begin by formulating the inputs and outputs, then explain how we built on the Ultralytics YOLO codebase (YOLO $\geq$ 8.2) to train a detector for all piece classes. To train the model, we begin with a dataset provided in CSV format, where each row lists an image filename, a piece or board class, and bounding-box coordinates. Each entry in this spreadsheet identifies which image file is under consideration, specifies whether it contains a particular chess piece or the board itself, and gives the rectangle that encloses that object. We convert these CSV annotations into the text-file format expected by YOLO by normalizing each box's center and size relative to the image dimensions. This means that if an image is 800 pixels wide and a pawn's bounding box extends from pixel 200 to 300 horizontally, we translate those pixel values into frac-

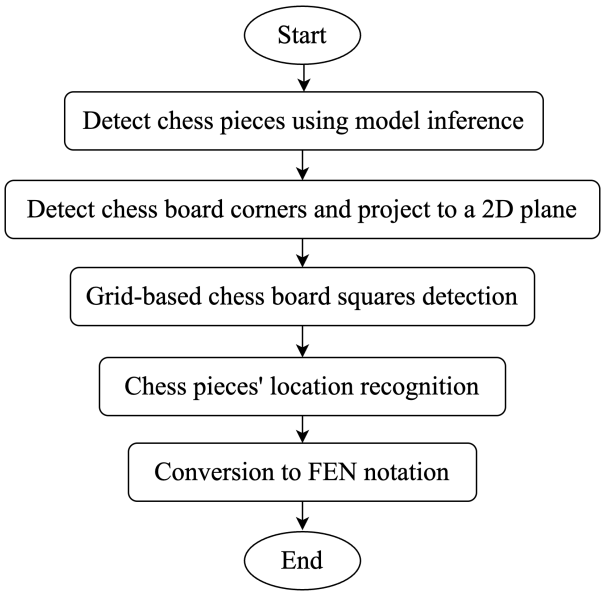

Figure 5. Algorithmic flowchart for chessboard analysis

tions of the total width so that YOLO always sees values between zero and one, regardless of actual image size. In addition, we randomly select ten percent of all training images to act as ”hard negatives” by creating empty label files for them, forcing the network to learn that some images contain no detectable objects. By including these blank examples, the model does not assume that every image must contain a piece or a board, which reduces false positives when no chess content is present.

With this configuration, we train the large YOLOv8 variant for twenty-five epochs on images resized so that the shorter side is 1280 pixels. Resizing ensures that even small details, such as a pawn near the edge of the board, remain large enough for the network to detect reliably. During training, standard YOLOv8 augmentations (mosaic, mixup, copy-paste, small random shifts in hue, saturation, and brightness, affine transformations, and horizontal or vertical flips) are applied to encourage generalization. In other words, we create many slightly altered versions of each image so that the model learns to recognize pieces under different lighting conditions, camera angles, and backgrounds. This process helps the detector perform well on images that differ from the original training set.

We also adjust the loss multipliers so that precise localization is emphasized by setting the box-regression weight to 7.5, and classification quality is given a modest boost by setting the class-loss weight to 1.2, while keeping the distribution focal loss at its default value of 1.0. This weighting tells the network that it is more important to draw very tight bounding boxes around each piece than to simply guess the correct class. At the same time, slightly upweighting the classification loss ensures that confusing a knight with a bishop is penalized more heavily than in the default configuration. After twenty-five epochs of training with these settings, we evaluate performance on a held-out validation split by computing mean Average Precision at a fifty percent Intersection over Union threshold (map50), and select the checkpoint that achieves the highest score for deployment.

At inference time, besides using the finetuned model for detecting chess pieces in images, we also used finetuned YOLO for detecting chess pieces in videos. At inference time, each video frame is processed in two stages to ensure accurate detection even under uneven lighting or modest board tilt. First, we resize the original frame so that its shorter side is 1280 pixels, producing a ”big frame.” By running YOLOv8 once on this resized version with a moderate confidence threshold (0.15) and non-maximum suppression at IoU = 0.5, we isolate one or more candidate chessboard boxes; if multiple appear, we keep only the largest by area. We then pad this board box by five percent on all sides to ensure no pieces near the edge are omitted, and crop that padded region. Since dark pieces on dark squares can be difficult to detect, we apply Contrast-Limited Adaptive Histogram Equalization (CLAHE) to the V channel of the cropped image, boosting local contrast. In the second stage, we run YOLOv8 again on the CLAHE-enhanced crop with a lower confidence threshold (0.05) and enabled test-time augmentation. Each resulting piece detection is filtered by a per-class threshold—0.04 for black pieces and 0.12 for white pieces to balance recall across colors. To project these detections back into the original frame, we account for both the crop’s offset within the resized frame and the scaling factor used initially. Finally, we perform a custom, class-agnostic non-maximum suppression with an IoU threshold of 0.3 to avoid suppressing adjacent pieces of different classes. Any remaining boxes are drawn on the original frame, along with a green rectangle around the board. The result is an annotated video where each frame shows exactly one detected chessboard and all detected pieces, each labeled with its class and confidence. At the core of our filtering step lies the standard IoU formula given by eq (1).

$$IoU = \frac{area(A \cap B)}{area(A \cup B)} \tag{1}$$

which ensures that overlapping detections with IoU $\geq$ 0.3 are suppressed in favor of the most confident prediction. This streamlined training and inference approach achieves real-time performance (approximately 5 FPS on an RTX 4070) while maintaining high accuracy under varied lighting and viewpoint conditions.

4.2. Chess board corners detection

Following YOLO-based chess piece recognition, precise detection of chessboard corners is essential. This allows for the projection of the chessboard onto a 2D plane, enabling its segmentation into an 8x8 grid for accurate chess piece localization and subsequent FEN notation generation. Achieving this precise corner detection proved to be a significant challenge, leading us to explore the following approaches:

4.2.1 GrabCut Segmentation based chess board corner detection

In this algorithm a robust image processing pipeline is implemented to detect chessboard corners. The algorithm commences with initial image enhancement, applying a morphological closing operation to reduce noise and bridge small gaps, followed by GrabCut segmentation to precisely isolate the chessboard from the background. This segmented image then undergoes a series of edge detection steps: conversion to grayscale, Gaussian blurring for noise reduction, and Canny edge detection, with subsequent dilation to ensure connected contours. The system proceeds to identify contours within the processed image, prioritizing the largest detected contours and approximating them to find a four-sided polygon, presumed to be the chessboard. Once these four corner points are accurately identified and consistently ordered (e.g., top-left, top-right, bottom-right, bottom-left), the algorithm calculates the optimal dimensions for the rectified board. This method also incorporates an outlier detection mechanism, comparing the dimensions of the rectified image to the original; if the projected image is significantly smaller than expected, it signals a potential failure in the projection, indicating the need for reprocessing. The successfully projected chessboard is then returned, optionally with a slight border crop.

This method proved highly sensitive to varying illumination and the presence of nearby objects, as detailed in the results. Furthermore, optimal hyperparameters (e.g., morphological kernel size, GrabCut iterations) varied significantly across images, necessitating a second application of the GrabCut algorithm with adjusted settings in some cases. Despite these efforts, sensitivity to illumination and border objects persisted. A key limitation is that this approach detects the physical chessboard corners, not the precise corners of the 64-square checkerboard pattern (see Figure 3 (b) or 6 (c) for an example of checkerboard corners). The varying offset between physical and checkerboard corners across different boards prompted exploration of alternative corner detection methods, including an unsuccessful attempt with a YOLO model.

4.2.2 U-Net based corner detection

We finetuned a pre-trained Unet image segmentation model [11] for detecting the corners of the 64-square checkerboard pattern. We used the efficientnet-b0 version of the Unet model pretrained using the imagenet dataset. Dice Loss and BCE with logits loss were used as the loss functions for training the Unet along with the Adam optimizer. The labels for the Unet segmentation model contain pixel values 0 and 1 in a mask. In other words, Unet takes an image as input and outputs a mask containing 0s and 1s. One indicates presence of the 64-square checkerboard pattern and zero indicates its absence. The binary mask was generated from the chessboard bounding box which was manually annotated using roboflow annotation tool [12]. On the testset, we achieved an IoU accuracy of 99.22%. However, IoU accuracy measures the overlap between the ground truth mask and the predicted mask. For our application, we needed accurate chessboard corner detection. Therefore, we also measured accuracy of correctly predicting the chessboard corners which was crucial for successfully generating accurate FEN notations. We achieved around 70% accuracy for a tolerance of 20 (which is the Euclidean distance in pixels error allowed for each detected chessboard corner). Detailed results are discussed in the results section.

4.3. FEN notation generation

Our methodology, outlined in Figure 5, initiates by employing a fine-tuned YOLO model to detect chess pieces within the input image. Subsequently, a fine-tuned U-Net model processes this image to precisely identify the chessboard's corners. Using a perspective transform, the chessboard's 64-square checkerboard pattern is then projected onto a 2D canonical plane, thereby providing a clear, unwarped view. On this rectified checkerboard image, the 64 squares are established by uniformly dividing the image's width and height into eight segments. Each square is labeled, and recognized chess pieces are assigned to their respective squares, serving as the basis for FEN (Forsyth-Edwards Notation) generation. The FEN generation itself involves transforming the center coordinates of YOLO-detected pieces to this 2D canonical projected plane, and mapping these transformed piece locations to their corresponding 8x8 grid positions to form the FEN string. The complete end-to-end pipeline, from initial image to FEN output, is visually demonstrated in Figure 6 (a)-(e). The image / information in Figure 6 (e) is used for FEN notation generation.

5. Experiments

5.1. YOLO

As aforementioned, we quickly identified YoloV8 as the state-of-the-art model for this task. Thus, we tested the var-

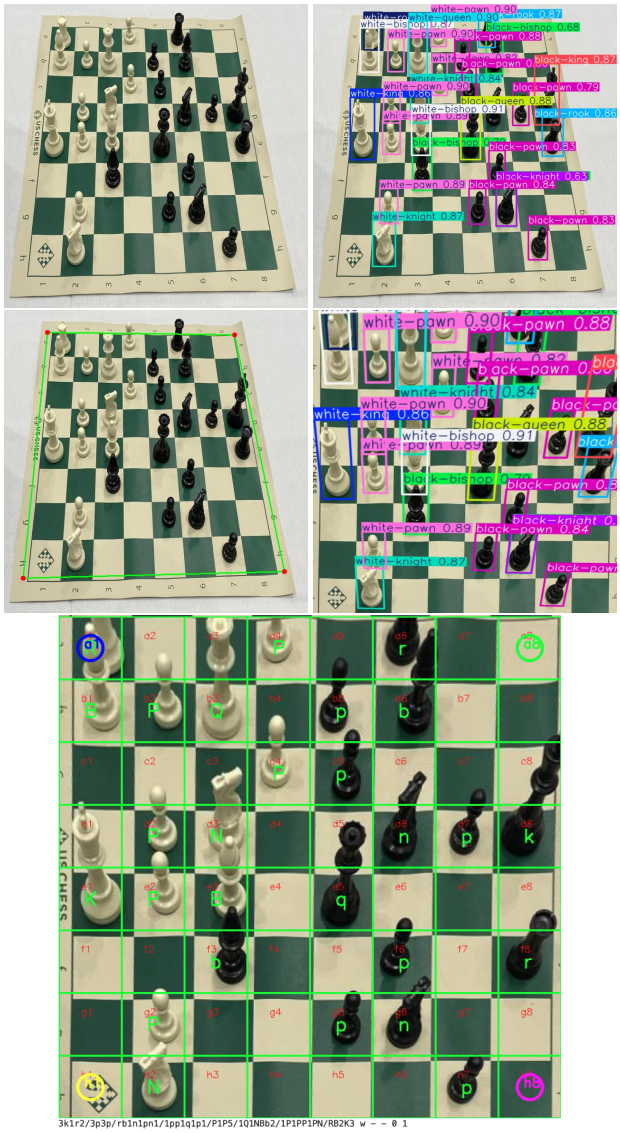

Figure 6. (a) Input Image (b) Chess pieces (YOLO inference) (c) Chessboard corners (Unet Inference) (d) 2D planar view / warped image (e) Grid image with chess pieces and chess board squares recognized.

ious YoloV8 submodels for 10 epochs of 640 images to see which one was best-suited for our task. We focused on four metrics:

- Box: The fraction of predicted bounding boxes whose overlap (IoU) with a ground-truth box exceeds 0.5. It measures how often detections are correct.
- Recall: The fraction of ground-truth objects that the model detects at IoU $\geq$ 0.5. It shows how many true objects are recovered.
- Map50: The mean of the Average Precision (AP) over all classes, calculated at a single IoU threshold of 0.50.
- Map50-95: Similar to Map50, but with a stricter threshold.

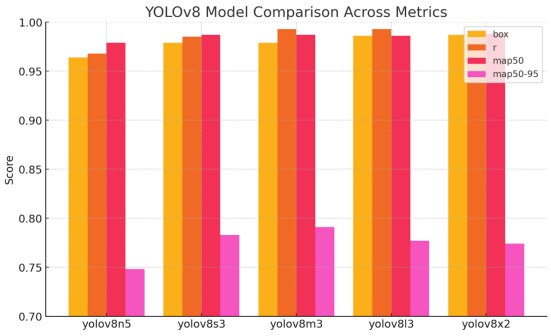

Figure 7. YOLOv8 model size vs accuracy

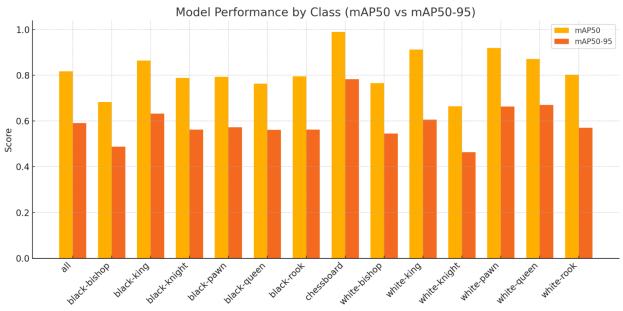

Figure 8. Accuracies of final model across various pieces

The graph in Figure 7 shows that as model size increases from nano to x2, box precision, recall and mAP50 steadily improve, while mAP50-95 peaks at the medium variant before plateauing. The l3 and x2 models offer negligible mAP50 gains over m3 despite a large jump in parameter count, suggesting an efficiency maximum at the medium scale. As efficiency is not a major concern, we used the large model as our basis. We focused on YoloV8 because it provides a complete family of variants. YoloV9 and later versions remain under active development with limited stable releases, and require some additional information to train. Overall, the accuracy of YoloV8 models out of the box is relatively high. The validation mAP50 accuracy is 81.3%. However, accuracy varied across different pieces, as shown in Figure 8.

Among all piece classes, the model struggled most with white knights and black bishops, as evidenced by their comparatively low mAP50-95 scores (0.464 for white knight and 0.488 for black bishop). One likely factor is the limited number of training examples: only 27 white-knight instances and 32 black-bishop instances were available, whereas more common pieces like pawns appeared over 90

Figure 9. Accuracy of Ensemble model

times. With so few examples, the network had less opportunity to learn the distinctive contours and textures that differentiate a knight's carved head or a bishop's slanted mitre, especially under varied lighting and board designs. In addition, knights and bishops can resemble each other when viewed from certain angles or when partially occluded, making them harder to distinguish. The knight's curved, animal-like shape sometimes blends into shadows or board patterning, and the bishop's pointed top may be confused with the more angular rook in lower-contrast scenes. Taken together, sparse training data plus similar visual characteristics under real-world conditions likely explain why these two piece types showed the weakest performance.

An average mAP accuracy of 81.3% seems reasonable: however, in video format, the problems with this emerge. Figure 9, below, is a still from the model being run on a video of a chess match, demonstrates this.

As shown, when run on videos dissimilar to the training, test, or validation datasets, the model sometimes incorrectly predicts similar-looking pieces: for example, recognizing a black bishop as a black pawn or a black queen as a black bishop. While these incorrect predictions are still certainly in the minority, they occur enough to disrupt accurate full-board recognition. We theorize that this is a result of the model overfitting on the training set: while we did endeavor to include pictures taken of several different types of chessboards and chess pieces, ultimately, the variation was likely insufficient.

5.2. Chessboard Corner Detection using Unet

For chessboard corner detection, our exploration encompassed two primary techniques, as outlined in Section 4.2: one utilizing GrabCut segmentation and another based on a U-Net model. The GrabCut approach, however, exhibited significant susceptibility to varying illumination and the presence of objects near the chessboard's periphery. More critically for our application, GrabCut identified the physical corners of the chessboard (as shown in Figure 4), rather than the precise corners of the internal checkerboard pattern, which are essential for the method's success. This led us to focus on the U-Net-based corner detection.

The U-Net-based chessboard corner detection leverages a fine-tuned U-Net, a state-of-the-art Convolutional Neural Network known for its high accuracy and relative computational efficiency in segmentation tasks. We fine-tuned this U-Net for checkerboard detection using 265 training images and evaluated its performance on 33 test images, as specified in the Dataset section. Initial evaluation using IoU (Intersection over Union), defined in eq (1), yielded an accuracy of 99.22%. However, upon visual inspection of predicted masks, it became apparent that IoU alone was not an ideal metric for our specific objective. Given that the efficacy of our overall method critically relies on the exact localization of chessboard (checkerboard) corners rather than broad pixel classification accuracy, we shifted our evaluation metric to the Euclidean distance between the predicted and actual checkerboard corner coordinates. Table 1 summarizes the average pixel-wise Euclidean distances for the four chessboard corners across the white and green, white and brown, and combined datasets. It was observed that the U-Net model yields better chessboard corner detection accuracy on the white/green dataset. This discrepancy is attributed to the presence of partially visible chessboards, coupled with suboptimal contrast and illumination, in the white/brown dataset, which negatively impacts detection accuracy compared to the consistently clear, well-contrasted, and well-lit images in the white/green dataset.

5.3. Final Results

Table 2 shows the final results of our ensemble model. We evaluated the model using the test dataset that comprised of 33 test images of various board types and pieces. We also evaluated each of the steps by checking whether 1. UNet model detected the corners accurately 2. Yolo model detected the pieces accurately 3. The integration algorithm was able to create an accurate 8×8 grid and was able to assign the pieces to their actual squares to generate accurate FEN notation.

As shown in Table 2, UNet corner detection achieved an accuracy of 81.8%, while YOLO piece detection reached 54.5%. Our end-to-end pipeline successfully generated correct FEN notation in 45.5% of test cases. Notably, when both corner detection and piece recognition were successful, the FEN accuracy increased significantly to 83.3% (15 out of 18 cases). This indicates that both accurate corner detection and precise piece recognition are critical components for our system to function properly. No test case resulted in correct FEN notation without both components working correctly, highlighting the sequential dependency in our pipeline architecture. The primary failure mode in our system appears to be YOLO-based piece recognition, which suggests that improving this component would yield

Table 1. Average Euclidean Distance (in pixels) between the ground truth and predicted chessboard corners

Corner Position	White and Green Chessboard Dataset	White and Brown Chessboard Dataset	Combined Dataset
Top Left	9.61	44.69	14.92
Top Right	7.41	4.14	6.91
Bottom Left	4.94	82.62	16.71
Bottom Right	6.32	166.25	30.55

Table 2. Performance metrics of chess position recognition system components

System Component	**Success Rate**	**Accuracy (%)**
Corner detection	28/33	84.8
Piece recognition	24/33	72.7
Square assignment algorithm	22/24	91.7
FEN notation generation	22/33	66.7
Performance by Board Type	**Success/Total**	**Accuracy (%)**
Standard boards	12/16	75.0
Minimal boards (1-2 pieces)	10/10	100.0
Non-standard boards	0/5	0.0
Combined Component Success	**Success/Total**	**Accuracy (%)**
When both corner and piece detection succeed	24/33	72.7
When only corner detection succeeds	28/33	84.8
When only piece detection succeeds	24/33	72.7
When both components fail	5/33	15.15

the greatest overall performance gains for future iterations.

6. Conclusion

In this study, we developed an automated system for analyzing chess board states, integrating a YOLOv8 model for piece detection and classification with a U-Net model for precise chessboard corner localization. Individually, YOLOv8 achieved commendable detection rates, particularly for frequently encountered pieces, and the U-Net effectively generated segmentation masks enabling the crucial perspective correction to an 8x8 grid. However, despite these strong component performances, our complete pipeline correctly generated FEN strings for only about 67% of the test cases. This limitation highlights the cumulative impact of even minor inaccuracies in either piece detection or corner localization, which can lead to erroneous square assignments. Specifically, detection accuracy was notably lower for classes with limited training examples (e.g., knights and bishops), and subtle mask misalignments from the U-Net sometimes caused sufficient corner shifts to misplace pieces.

To enhance system robustness and push performance beyond 67%, several avenues for future work are proposed. Model-centric improvements include increasing the diversity and volume of training images for underrepresented piece types and meticulously fine-tuning class-specific confidence thresholds. For the U-Net, incorporating a direct corner regression head alongside the segmentation mask or leveraging higher-resolution masks could substantially improve corner precision. Beyond model refinements, strengthening the inference pipeline is critical. This could involve adding a lightweight post-processing step to verify piece centers within the transformed grid, potentially snapping misaligned pieces to the nearest valid square. Implementing a feedback loop to reprocess improbable FEN outputs (e.g., overlapping pieces) with adjusted thresholds would also be beneficial. Furthermore, collecting ”hard” negative examples, such as images with background patterns resembling a chessboard, would bolster YOLO's resilience to false positives.

7. Contributions and Acknowledgment

Our team's contributions were distributed as follows: Matt Yang was responsible for training the YOLOv8 model and applying it to chess piece recognition in video. Samruddhi Kahu focused on chessboard corner detection, exploring both GrabCut-based segmentation and fine-tuning a U-Net model. Satya Biswal integrated the YOLOv8 and U-Net model inferences, optimized the Unet model inference, and worked on grid generation for piece assignment, culminating in the generation of FEN notation. Satya also created an app to take a picture of a live chessboard, extract FEN notation, and display the next move. We will demo this app in the poster presentation.

References

[1] IMT Kaggle Team, *Chess Pieces Detection Image Dataset*. Kaggle. https://www.kaggle.com/datasets/imtkaggleteam/chess-pieces-detection-image-dataset

[2] Tanner G., *Chess Piece Object Detection Dataset*. Kaggle. https://www.kaggle.com/datasets/tannergi/chess-piece-detection

[3] M. Czyżewski, A. Laskowski, and S. Wasik, "Chessboard and Chess Piece Recognition with the Support of Neural Networks," *arXiv preprint arXiv:1708.03898*, 2017. https://arxiv.org/abs/1708.03898

[4] D. Yang, "Building Chess ID," *Medium*, Jan. 17, 2016. https://medium.com/@daylenyang/building-chess-id-99afa57326cd

[5] Rizo-R, *Chess-CV: Computer Vision Project to Recognize Chess Positions*. GitHub repository, 2025. https://github.com/Rizo-R/chess-cv

[6] A. Mehta, "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*, 2020. https://arxiv.org/abs/2009.01649

[7] G. Wölflein and O. Arandjelović, "Determining Chess Game State From an Image," *arXiv preprint arXiv:2104.14963*, 2021. https://arxiv.org/abs/2104.14963

[8] M. Conner and R. Talvi, "YOLO for Chess Detection: An Implementation of the YOLO Algorithm," Project report, Mar. 2023. https://dothereading.github.io/projects/YOLO_Chess.pdf

[9] Maia, "Depth-based Segmentation with YOLOv4 for Chessboard and Piece Detection," Roboflow Universe dataset, 2022. (See Roboflow search for "Maia chess YOLOv4.")

[10] J. Chen, "Chess Piece Classification and Localization Using Depth Camera and CNN," Technical report, 2022.

[11] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," in *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, vol. 9351, pp. 234–241, 2015. https://link.springer.com/chapter/10.1007/978-3-319-24574-4_28

[12] Roboflow, "Roboflow Annotation Tool," https://roboflow.com (accessed Jun. 4, 2025).