Chess Recognition

Andrew Annestrand
University of Washington
atannes@uw.edu

Shriraj Gandhi
University of Washington
ssgandhi@uw.edu

Abstract

Chess is one of the oldest and most popular board games in human history that is played between two players. The internet has created plentiful opportunity to play opponents on a digital chess board from around the world. However, most important chess competitions are played over-the-board (OTB) rather than in a digital format. In the absence of a real-life opponent, a chess player can only play against themselves. In this project, we build a system that enables a user to play OTB chess against a virtual opponent. Specifically, it allows a player to play chess on an actual board, and converts the image of the chess board into a machine readable representation of the state of the chess board.

1 Problem Statement

Capturing the state of a chess game comes down to two important components: board recognition and piece recognition. Board recognition refers to the ability to look at an image of a chess board and recognize the four corners of each of the 64 squares on the chess board. Piece recognition, on the other hand, refers to the ability to look at the chess board image and recognize where the pieces are, and what kind of pieces they are. Board recognition is mostly a geometric problem, and so we applied traditional image processing algorithms and techniques. Piece recognition is ultimately a classification problem, which led us to use machine learning techniques to solve it.

1.1 Challenges

Although this sounds like a straightforward problem, there are several variations in the input that make it more difficult than it seems at first glance.

1. Angle: The image could be taken from above the board, from the side, or at a diagonal angle.
2. Board Color: Although in theory a chess board is made of black and white squares, the most common chess boards are white and green. Blue, brown, and some other colors are also often used instead of black.
3. Piece Color: Similarly, black pieces are often blue, brown, or other colors in practice.
4. Lighting: White pieces are fairly reflective, to the extent that the features defining the piece get overshadowed by the reflection of light.

1.2 Data Capture Setup

In order to standardize the process, image was capture at a fixed top-down angle. The reason this was done was to minimize occlusion of squares on the chess board. When taken at angles, the problem of board recognition becomes increasingly more challenging and less accurate.

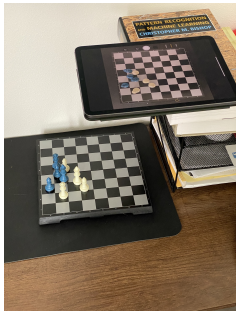

Figure 1: Setup for image capturing.

2 Board Recognition

2.1 Early Approaches

In order to take advantage of the naturally contrasting colors of a chess board squares, some of the first attempts used thresholding and morphology. Specifically, we applied an adaptive thresholding (Otsu's Method [1]) in order to account for glare and variable lightning conditions. The adaptive thresholding finds the optimal threshold value based on local pixel intensities. After applying the thresholding to our image, we then applied morphology to smooth squares and reduce noise. In particular we chose to use the Open method which is simply an erosion followed by dilation. Lastly, we applied a blob detector in order to locate the chess squares in the image. This solution was quite effective in finding empty squares on the board, however suffered severely when pieces were present (see 2). The biggest reason why this failed was the morphology of the piece would perturb the structure of the square if the colors were inverse (white piece on black square, vice versa). In addition, this method provides no information on the specific id of the square on the board. All of these issues led us to finding a more robust way to dissect a chess board even when pieces were present.

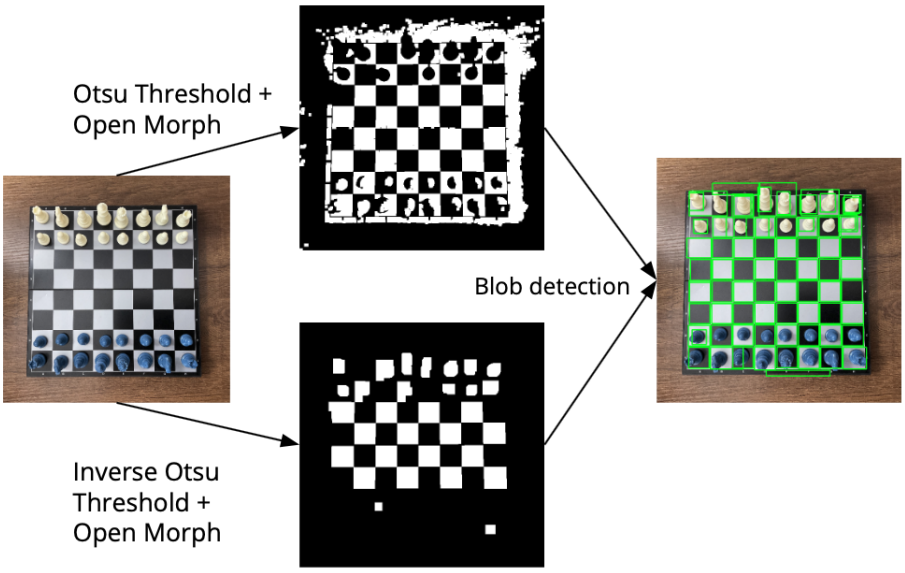

Figure 2: Adaptive Thresholding, Morphology, and Blob Detection.

2.2 Final Detection Algorithm

Since we wanted to have a pipeline that detected location of squares regardless of if there was a piece present, we decided to focus on the lines that make up the squares in the chess board. After converting the image to gray scale, we apply the Canny edge detector. The motivation behind this is that we know that abrupt pixel values between differently colored scales will lead to a large gradient. These large gradients end up making a perfect outline of all the squares on a chess board. Next we applied a Hough line transform to the Canny image in order to detect the lines that separate the squares. This gave us many great results as we had lines that were separating the chess board. However, many times there would be a bunch of collinear lines next to each other (representing the same chess line). This was frustrating since we the system now had a good idea of were the correct lines were, but they weren't exactly fitting the chess board. The solution that we implemented was a clustering algorithm that would group lines if they were of the same orientation (horizontal or vertical) and within a small distance of one another (we used an empirically effective distance threshold, but this would need to change if the angle of the image drastically changes). After clustering, the mean line for each cluster was calculated. Lastly, it was important that we found the intersection of all of the lines to properly track the chess square. To achieve this, we simply projected the mean lines to the ends of the image (see 3).

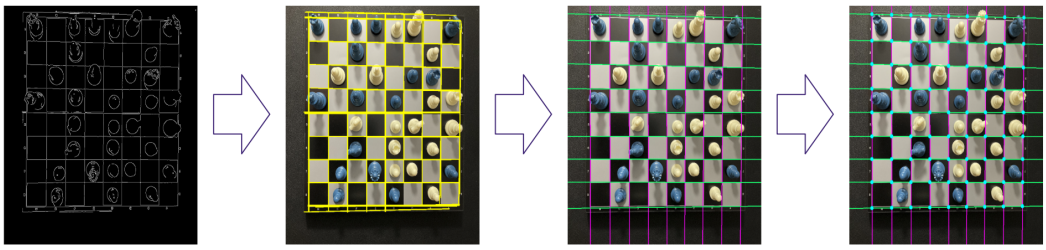

Figure 3: Canny Edge Detection, Hough Transform, Clustering & Projection, Intersection Points

If the lines are properly sorted prior to finding intersection points, then square identification and iteration now becomes trivial. The last task to full board recognition is to find the specific IDs of the squares (A1, B1, ..., H8). This must be done when the chess pieces are placed in their proper starting position before the game begins. A1 is always the square containing the left rook of the white players position. To find this square, we simply compare the bottom-left and top-right black squares (the only legal starting squares) and the square with the higher mean value is deemed A1.

The final structure of the board recognition was a simple 2D python array that was the size of the chessboard in pixels. Each value was assigned to a certain square based on the previous A1 findings. After creating a digital map of a chess board, it was time to start recognizing pieces.

One important note is that although this algorithm was evaluated mainly on our top-down viewpoint, it also performs well at slight angles. However, if the angle becomes too large, the clustering algorithm will struggle due to line distances being non-normalized (due to depth). A solution to this would possibly be to have an intelligent way to calculate this distance heuristic based on camera angle (see 4).

3 Piece Recognition

3.1 CNNs

Our first approach was to take the output of the board recognition algorith - 64 squares - and feed it to a CNN for piece recognition. We padded a few extra pixels to each square we generated earlier, since some pieces spilled over outside their square boundaries. We built two models - one that took images as input and outputted the type of piece (e.g. rook, king, queen), and one that outputted the color. We figured that the fewer the output classes we kept per model, the easier time it would have training itself, especially with limited training data.

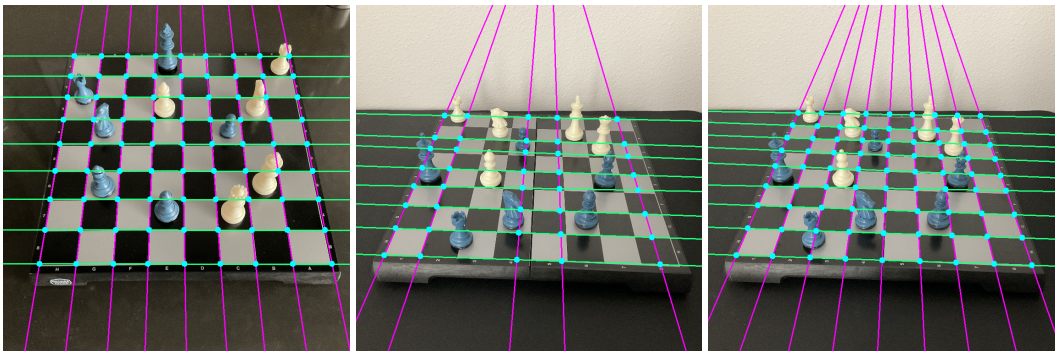

Figure 4: Good Detection at slight angle, Bad Detection at larger angle, Correction to distance metric for good detection at larger angle

After training a simple CNN, we achieved nearly 80% validation accuracy on piece type and over 99% on the color. However, the model outputted ”pawn” and ”blank square” almost exclusively. In

3.2 YOLO

3.2.1 Transfer Learning

To avoid some of the challenges faced when training the CNN models from scratch, we decided to try some SOTA models. We opted to transfer learn on the YOLO network since it offers bounding boxes and classification for objects. In order to train a YOLO model, we had to individually annotate the bounding boxes and class for every piece in an image. This was quite a lot of manual work. In total around 60 images were fully annotated and classified, but we were able expand the size of our dataset to 212 images by adding minor perturbations. Some of these perturbations include rotation, noise, and lighting shifts.

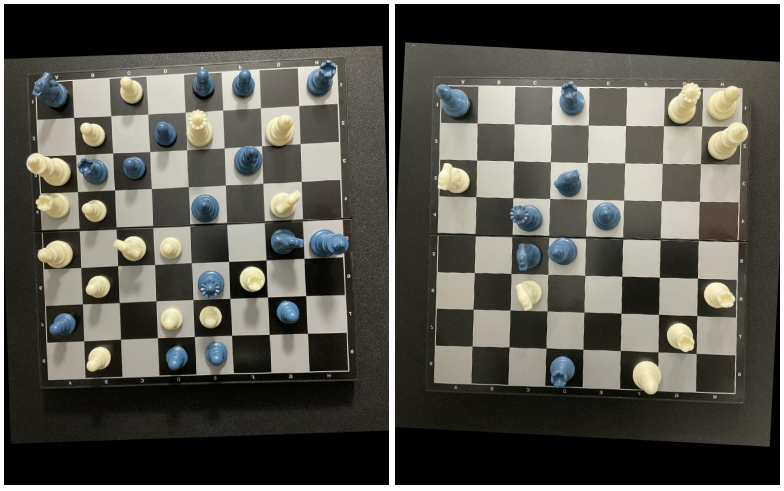

Figure 5: Lighting shift, rotations

Once the dataset was compiled, we trained the YOLO model on 100 epochs. Ideally, the number of epochs shouldn't necessarily be that high but due to the small dataset convergence took a long time. Shown in Table 1, we have the classification a bounding box metrics for the validation set. We arrive at impressive results with most precision/recall above 90% for all classes. Further, the mean average precision (mAP) at IOU of 0.95 is 0.973 for all classes which indicates that the bounding boxes are concise and accurate.

3.2.2 Results

Since the ouputs of the YOLO model are both bounding boxes and class, we had to do a little extra work to determine the location of the pieces in our pixel map. Iterating through the bounding boxes, we indexed our pixel map in the area of the box and assigned the mode (most common) square id the associated class of the bounding box.

Table 1:

Validation Results after 100 Epochs						
Class	Images	Labels	Precision	Recall	mAP@.95	mAP@.5
All	12	207	0.969	0.962	0.973	0.771
black-bishop	12	23	0.952	1	0.979	0.776
black-king	12	12	0.985	0.917	0.951	0.78
black-knight	12	18	0.895	0.952	0.932	0.721
black-pawn	12	22	0.992	1	0.995	0.785
black-queen	12	10	0.983	1	0.995	0.75
black-rook	12	19	0.991	0.947	0.954	0.746
white-bishop	12	22	0.927	0.955	0.956	0.783
white-king	12	13	1	0.923	0.939	0.799
white-knight	12	17	0.991	0.941	0.992	0.768
white-pawn	12	22	1	0.996	0.995	0.78
white-queen	12	12	0.913	1	0.995	0.808
white-rook	12	17	1	0.909	0.99	0.751

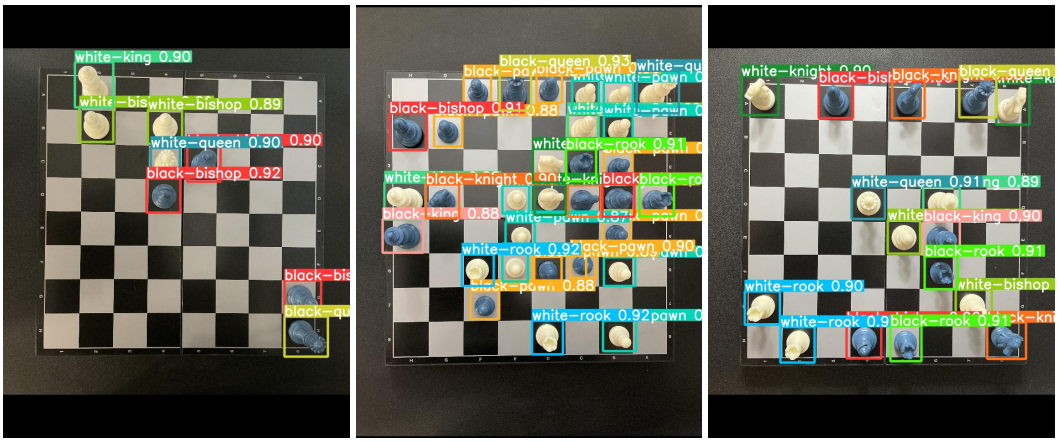

Figure 6: Resulting ouputs of the YOLO model evaluated on our chess dataset.

4 Conclusions

To conclude, we were able to accomplish our goals of building a system that can translate an image of a chessboard into a digital representation of the game state. We relied on a variety of different computer vision techniques ranging from canny edge detection to advanced machine learning models like YOLO. The board recognition is very precise at the right angle, and our piece recognition provides accurate classification of ¿ 90 % for all classes.

5 Future Work

Going forward, the board recognition algorithm could be improved to account for sharper angles using intelligent heuristics. Further, we believe that model performance can be improved even fur-

ther by increasing the size of our dataset. Lastly, an application that can take advantage of our final system and provide an interactive game with a Chess AI would be an excellent final step.