Mini-Project Report

Chess Board Analysis Ai with Computer Vision

Team 17

Adam Hall, Bhargav Panchal, Zhiyang Cheng

§ 1 Problem statement and analysis

Chess, a strategic board game, demands keen perception and foresight. However, players, regardless of their skill level, often struggle to recognize the full potential or pitfalls of their current board positions. Our project, leveraging the capabilities of computer vision, aims to assist chess players by providing a tool that analyzes chessboard images to recommend the best strategic moves. By uploading an image of their setup, players receive instantaneous feedback, helping them understand their position better and make informed decisions. This technology calculates a numerical score that reflects the position strength and suggests optimal moves based on the game state.

§ 2 Use-case scenarios

Our solution is crafted for everyone from hobbyists to professional chess players. It primarily involves a player snapping a photo of their chessboard at any point during their game and uploading it to our app. The system then analyzes this image to give a comprehensive review of the current game situation, which includes scoring the position and recommending potential moves. This helps players make better decisions and also acts as an educational tool, enhancing their understanding of chess strategies.

§ 3 Literature review

In our project on analyzing chessboard images, we incorporate several sophisticated image processing technologies to boost both the precision and efficiency of our computer vision system. We use the Hough Transform to accurately identify the lines of the chessboard grid, which is crucial for pinpointing the exact squares where chess pieces are placed. Before this step, we apply Canny Edge Detection to clearly outline the edges of the chessboard and the pieces, ensuring that the Hough Transform has accurate data for line detection. Additionally, we use Roboflow, an advanced platform that helps manage and enhance our image dataset. Roboflow improves the variety and quality of our data through preprocessing and augmentation, which in turn boosts the effectiveness and reliability of our machine learning models. These technologies are integrated smoothly into our software, collectively enhancing our capability to accurately analyze chess games and provide timely strategic advice.

§ 4 AI algorithm and model

The section will go over how the chess board was analyzed and how we extracted the board state to analyze further. For the chess board analyzed the algorithm will be talked about along with the efficient improvements and board state handling. Computer Vision demonstrates how we classified the pieces on the board and places at locations with edge detection feeding the results into the analyzer.

§ 4.1 Chess Board Analysis

Chess board analysis was done with the understanding that resources on the board like rooks, queens, pawn, and rook have different value to the player. And, that the players have different interests in mind. Combining this information we gather a way to predict the moves of openents based on the best moves of each player. However, the full understanding of the game is very complex and impossible to fully calculate.

§ 4.1.1 The Algorithm:

The algorithm behind the board analysis was chosen to be a min max algorithm. This best fits the understanding of the best interests where if it's the players turn we want to maximize our turn and minimize the other person's turn. This best fits chess, since it's a turn-based games against only two players.

§ 4.1.2 Efficient Improvements:

The min max algorithm for chess is very expensive to compute, in fact it will take so long it would easily take over a thousand years and no one has that time. However, we can improve the performance of the algorithm and limit the computation power that is required to know the approximate best move/score of the board. First we use alpha beta pruning, since it's a rather obvious improvement that prunes unnecessary branches that will not affect the end result. Also, for this project along with alpha beta pruning we reordered the output of getting the states to help promote states that are obviously better.

§ 4.1.3 Board State:

The board state itself is very important to create because this goes hand and hand with the min max algorithm. For this project we used python chess to help analyze the board states. Originally we planned on developing the board state from scratch. However, this stood with some challenges such as special cases that may occur during the game like en passant and the king can't move into check or we can choose the promotion. With all these reasons in mind along with efficiently we decided to go with python chess.

§ 4.2 Computer Vision(Cheng):

Computer Vision has two parts. One is chess classification, and our program will classify the type of each chess piece in the image and use a box to select them. The next part of computer vision is to assign the location of each chess piece from the image to the chess board.

§ 4.2.1 Chess Classification:

We want to train our model with a deep learning algorithm. Thus, we choose the YOLOv8(You Only Look Once) to train our model. YOLO is a famous object detection algorithm with some pre-trained models for object detection, like cats, dogs, humans, etc. However, chess piece detection is not a built-in function of YOLO. So, we need to train the model ourselves.

§ 4.2.1.1 More about YOLO:

Basically, the YOLO is based on a convolutional neural network(CNN). There are three important layers that I want to mention. First is Conv2D layers, which are used to extract the features from the image. Second is the Batch Normalization layers, which are used to reduce the number of iterations that need to converge.

§ 4.2.1.2 Data Sets

The datasets that we used for training is downloaded from Roboflow. Here is the link(https://public.roboflow.com/object-detection/chess-full/24). There are 693 pictures inside the dataset. 606 are used for training, 58 are used for validation, and 29 are used for testing. Meanwhile, all chess pieces are labeled with the location on the image and class. But, the size of the dataset is not enough. Thus, we decided to use data augmenting. Figure 2, shows the augmenting parameter that we used. Pictures are augmented by changing the hue, saturation, brightness, degree, and size of pictures. Also, we will flip each image left and right, and combine four pictures together.

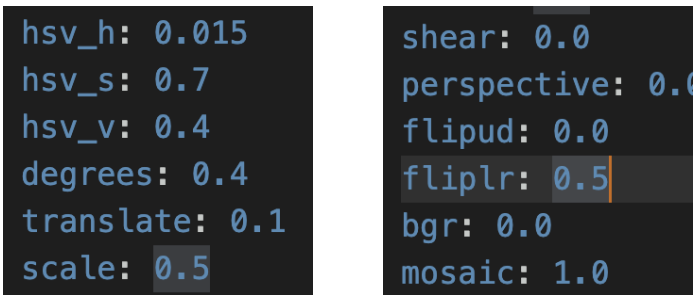


Figure 2. Augmenting Parameter

§ 4.2.1.3 Chess Pieces Classification Result

Figure 3 compared the f1 score between 10 epochs and 50 epochs, which shows that 50 epochs have better results than 10 epochs. Figure4. Shows the training result of 50 epochs. Which plots the error after each epoch. Which has the error of framing the chess, the error of classification, etc. I will show the classification demo in the demo section.

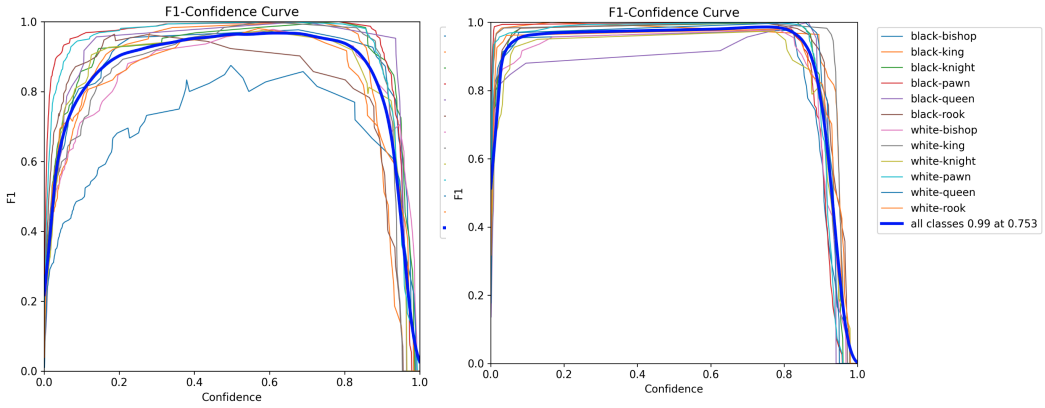


Figure3. F1 Score After 10 epoch vs. 50 epoch

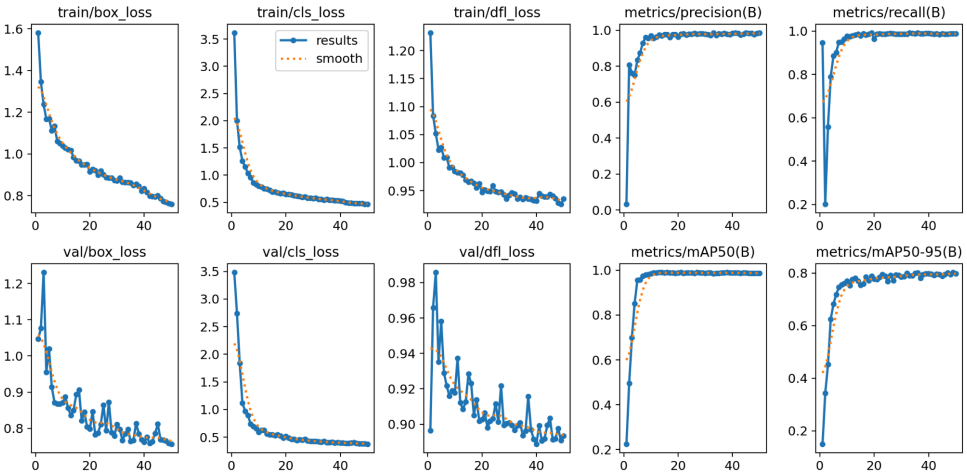


Figure4. Training Result of 50 Epoch

§ 4.2.2 Chess Location Detection:

After we have the classification information of each chess piece, we need to transfer their location on the image to the location on the 8 x 8 chess board. The basic process uses canny edge detection to detect the edge on the graph, and then uses the Hough transform to find the line on the image based on the edge we detected. After that, we found the corners by using the intersection of the line. Then we use K-means to filter out the noise points.

§ 4.2.2.1 Canny Edge Dection:

When we perform the Canny Edge Detection, we must first transfer the image to the grayscale. After that, we detect the edge based on the gradient intensity. The point with a high gradient intensity can be considered an edge point

§ 4.2.2.2 Hough Transform:

After we found the edge from canny edge detection, we performed the hough transform based on the edge. Then we let each edge point vote for the line, which means the line has enough edge points lying on it, then it will be considered as a valid line

§ 4.2.2.3 Filtering out the noise points:

Figure5. Shows the noise corners. Because there are only 81 corners on the chessboard, thus we use the K-Means to cluster them in 81 clusters. Figure6 shows the True corners that we get after using K-Means. The white dots are 81 true corners.

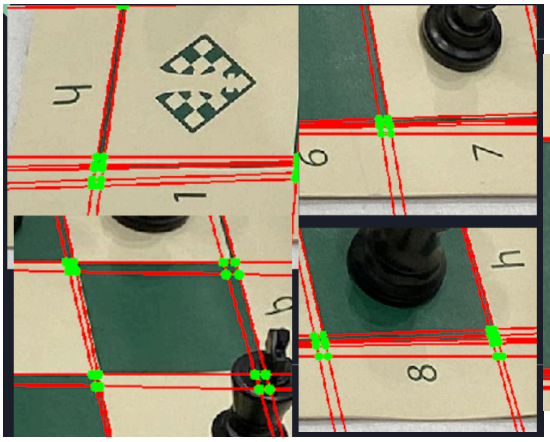


Figure5. The noise corner

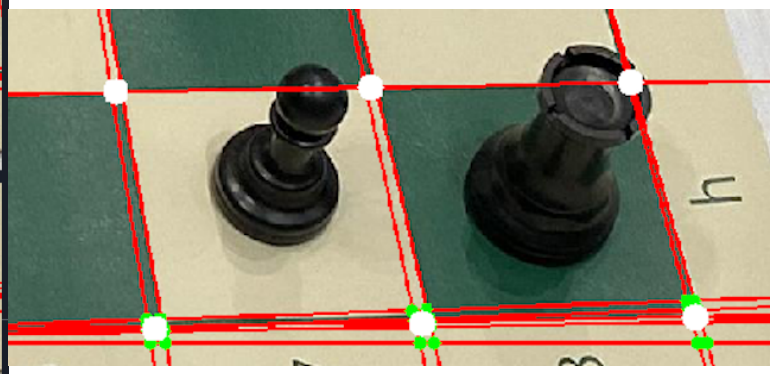
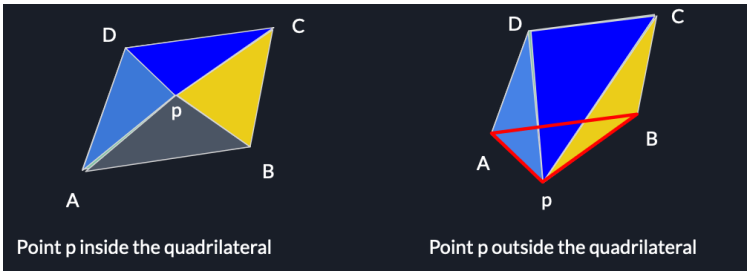


Figure6. True Corners after Perform K-means

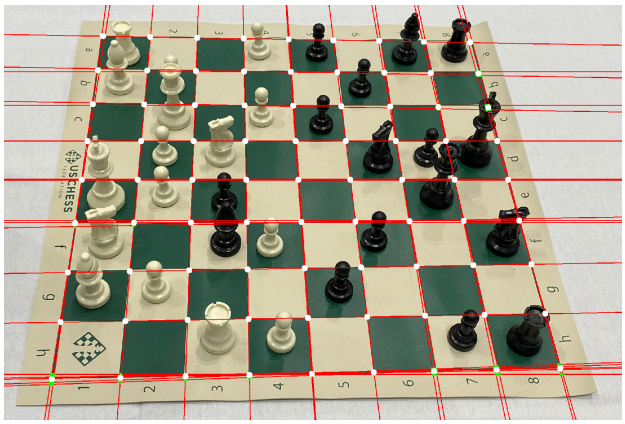
§ 4.2.2.4 Assign Chess Pieces on Chessboard:

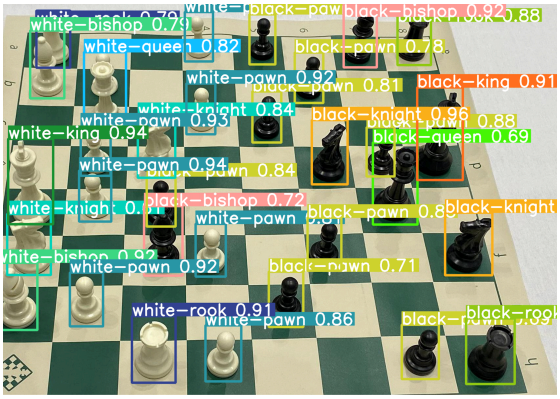
First, we sort 81 corners into a 9 x 9 array, each corner are stored as a tuple(x, y). Then we iterate through all quadrilaterals in a 9 x 9 array to assign the coordination of chess pieces on the chessboard. The way that we check if a point is inside the quadrilateral is to compare the quadrilateral area and the sum of the area of four triangles that are connected to the point. Figure7 is a demonstration.



§ 5 Results and demonstration

Below is the image of line detection, and the white point is the true corners. Then is the image with classification,





Next is the assigned location of each chess piece and its class. Then is the best move we found for the white side.

```
. . R P . . p r
B P . . p . . .
N . b P . p . n
K P p . . . q .
. P N . . n p k
. Q . P p . . .
B . . . . p . .
R . . P p . b r
A B C D E F G H
Chess Board score: 7
Chess Board move: c8c6
```

§ 6 Code and documentation

Run “WrapUp.py” to run our project, first it will pop up an image of line detection, then press “ESC” to see an image of the true corner on the image. Then press “ESC” again, and the classification result will pop up. Then the program will ask you to choose a side, if you are white then type yes, if you are black type no. Then you will see the best move.

In this section, I will talk about the folder structure of our project, and what is the purpose of each file.” dataset/” has three folders, which are train, test, and valid. All of the images and labels are in YOLOv8 Format. “runs/” has all results from training, but the best one we put in the “TrainedModels/”. “YOLO/” includes the data set arrangement file “data.yaml”. “ChessBoard.py” includes all code for chessboard analysis. “ChessCls.py” includes all codes for chess pieces

classification. “GridLocatior.py” includes all codes of allocation of chess pieces on the chess board. “TestModel.py” is used to test the model we train. “TrainMode.py” is used to train the chess classification mode. The “Until.py” includes some helper functions for calculation

§ 7 Lessons learned

Computer Vision:

1. Increasing the size of the batch will not always speed up the training.
2. Sometimes a simple algorithm can solve huge problems, like the K-means.
3. Using Hough transform and Canny edge detection algorithm
4. Training the classification model

Algorithm/Board State

1. The chess board state are very difficult to emulate
2. Algorithms that fully cover chess entirely are impossible
3. Alpha beta pruning can be sped up by using move reordering

§ 8 Future work

1. Using more advanced algorithms to predict the best move such as reinforcement deep learning
2. For analysis of the board state their other indicator of a stronger positions and can be explored
3. Using more datasets to train our classification model to make our program can fit with a different type of chess board
4. Making Computer vision part can detect the orientation of the chess board

§ 9 Link To Code

https://github.com/Yang-NPC/ChessAiFinal