Chessboard Recognition with Hough Line Transform and Neural Networks

Rain Juhl
rjuhl@stanford.edu

Praneeth Kolichala
pkolich@stanford.edu

Eric Feng
ericfeng@stanford.edu

Abstract

The goal of this paper is demonstrate a method for computers to recognize the state of chess board given an image. The method involves first finding the location of each piece in an image by identifying the chess grid using Canny Edge Detection and Hough Line Transform. The second half of the models takes each piece or empty square on the board and gives a prediction for its type using a ResNet18. Working together both parts are able to parse the state of board from an image.

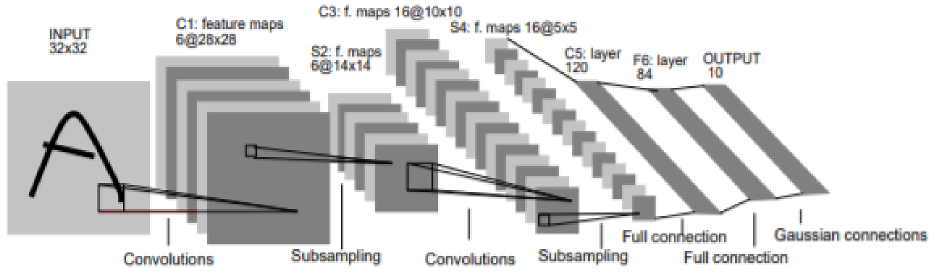
1 Introduction

Chess is one of the oldest recorded board games in world history. Originating from India in the 5th century, the game of chess has been passed down from generation to generation. It is played on a square chessboard with 64 squares arranged in an eight-by-eight grid. At the start, each player (one controlling the white pieces, the other controlling the black pieces) controls sixteen pieces: one king, one queen, two rooks, two knights, two bishops, and eight pawns. In aggregate there exists over 10^50 possible setups for chess games. Setting up these games on digital devices for analysis takes time and resources to get the correct set up. In this project we will be exploring the use of convolutional neural networks to recognize images of chessboards, and automatically generate game states based on the image. To this end, we train a model to classify pictures of individual pieces, and we detect chessboards and the grid using edge detection. Combining these results in a program which can view an image of a chessboard and automatically detect where each piece is.

2 Literature Review

Yann LeCun [1999] Object Recognition with Gradient Based Learning: This paper first proposes the utilization of Convolutional Neural Networks. The original idea for Convolutional neural networks was based on 3 main concepts: Local receptive fields, shared weights and spatial subsampling. For the Local Receptive Fields, the idea was that the inputs of a layer came from a neighborhood of inputs in a previous layer. By utilizing Local Receptive Fields of convolutional neural networks, the network is able to pick up on elementary structural features such as edges, endpoints, and corners. The idea of weight sharing is utilizing the same parameters for weights for elements in the same plane. This greatly decreases the needed number of parameters needed to train the convolutional neural network, making training easier and decreasing the possibility for overfitting. Lastly the Convolutional Neural network utilizes spatial subsampling, which is a technique utilized for mapping previous feature maps to smaller feature maps. This is utilized for down sampling, so the final layer can produce a reasonable output.

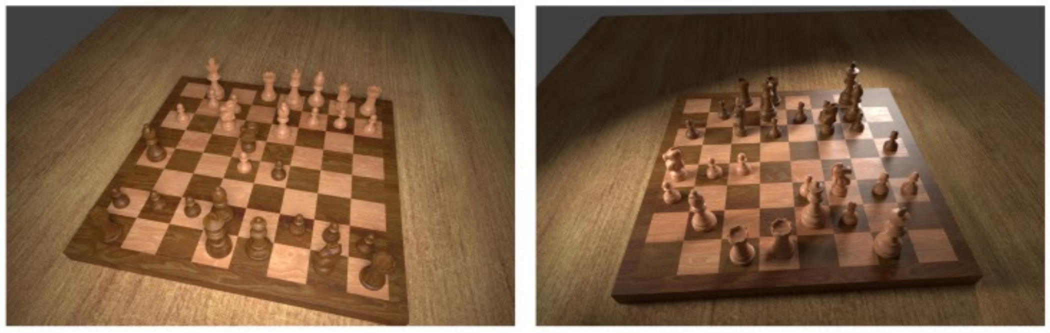
Preprint. Under review.



One reason CNNs might have done so well in image recognition is due to their translation invariance – that is, if we shift all the input pixels by a certain amount, the output of each convolution will also be translated by that amount. This means CNNs have an "architectural prior" (a prior embedded in the architecture itself) towards detecting objects regardless of their position in the image, which is a desirable feature for image recognition.

2.1 Determining Chess Game State from an Image

Wölflein [2021] In this paper, they address the issue of creating game states utilizing computer vision. The paper claims that previous models in the past have been unsuccessful due to a lack of large datasets. In this paper, 2 percent of all positions from the total 2851 games that Magnus Carlson has played were samples. From those 4888 positions, the researchers utilized a 3D modelling system to model those positions in a realistic image (see below)/. This was to increase the amount of valid data points in the dataset by magnitudes greater than all previous available chess datasets. The paper first utilizes a Random sample consensus that computes a projective transformation of a board onto a normal grid. Then using two convolutional neural networks the first network predicts which parts of the grid are occupied, then the second convolutional neural network predicts what pieces are in the predicted spots. Utilizing this technique the team was able to reach a per-square accuracy of 99.83% accuracy on datasets that have not been seen.



(**a**) Camera flash (**b**) Spotlights

2.2 Dataset

We combined two datasets for this task: one dataset containing raw chess pieces, and one dataset containing chessboards with some pieces on them. The dataset containing raw chess pieces comes from Geoffrey Kao [2018], and contains close up images of chess pieces as well as empty squares on a chessboard.

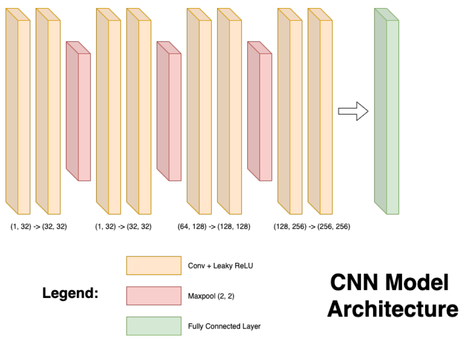
The second dataset came from Roboflow [2021]. We divided these chessboards into training and testing. The chess pieces from the chessboards of the training dataset were cropped out and added to the total training data, after resizing. The chessboards themselves in the training section were used to test our chessboard square detection code (this code simply used an edge detector, and did not require statistical machine learning). The chessboards were also pushed to grayscale before being fed through the edge detector to detect chessboard squares.

The training dataset that was fed into the machine learning model underwent the following preprocessing: first everything was scaled to a uniform size. Next, we normalized the image pixel data to have mean 0 and standard deviation 1. Each piece of training data was randomly flipped horizontally with 50% probability in each epoch, and a random section of it was cropped and resized to the full size, in order to make the model robust to reflections and translations of pieces.

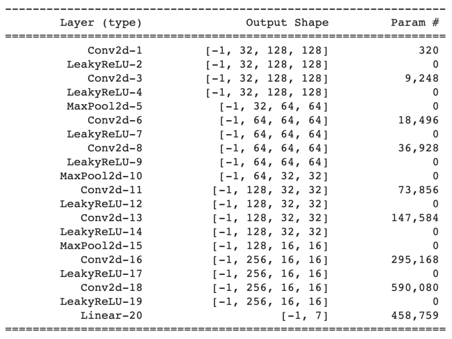
The test section of the Roboflow dataset was used to determine the final test accuracy of the model.

2.3 Baseline

For our baseline we decided to create our own rudimentary CNN model using limited computing resources. We preprocessed the data by splitting the data into a training dataset, a validation dataset, and a test dataset. Then for each of the images we grayscale the images, normalize them and rescalled the images down to the constant size of 128 x 128. The layers for our CNN. The goal of the baseline was to be able to identify individual grayscale chess pieces. Ideally, the classifier would have a high accuracy but for the baseline the goal was to do better than random. The classifier consisted of four double convolutional layers separated by max pooling layers. The specifics are shown below:



(a) Model Layers (Image)

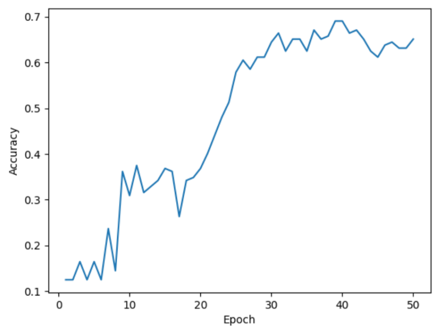


```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 32, 128, 128]             320
         LeakyReLU-2         [-1, 32, 128, 128]               0
            Conv2d-3         [-1, 32, 128, 128]           9,248
         LeakyReLU-4         [-1, 32, 128, 128]               0
         MaxPool2d-5           [-1, 32, 64, 64]               0
            Conv2d-6           [-1, 64, 64, 64]          18,496
         LeakyReLU-7           [-1, 64, 64, 64]               0
            Conv2d-8           [-1, 64, 64, 64]          36,928
         LeakyReLU-9           [-1, 64, 64, 64]               0
        MaxPool2d-10           [-1, 64, 32, 32]               0
           Conv2d-11          [-1, 128, 32, 32]          73,856
        LeakyReLU-12          [-1, 128, 32, 32]               0
           Conv2d-13          [-1, 128, 32, 32]         147,584
        LeakyReLU-14          [-1, 128, 32, 32]               0
        MaxPool2d-15          [-1, 128, 16, 16]               0
           Conv2d-16          [-1, 256, 16, 16]         295,168
        LeakyReLU-17          [-1, 256, 16, 16]               0
           Conv2d-18          [-1, 256, 16, 16]         590,080
        LeakyReLU-19          [-1, 256, 16, 16]               0
           Linear-20                    [-1, 7]         458,759
================================================================
```

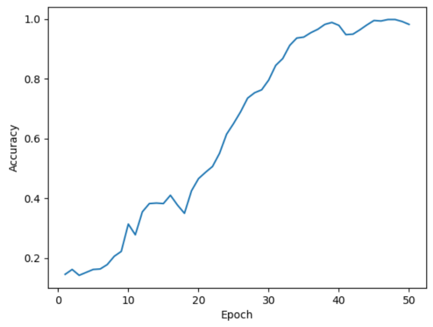
(b) Model Layers (Text)

Figure 1

The model was then trained for 50 epochs. To keep computing resources low for the baseline the images were converted to size 128x128. This resulted in a final accuracy of 67% on the test data.



(a) Test



(b) Train

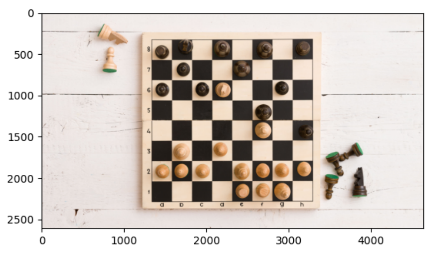
Figure 2

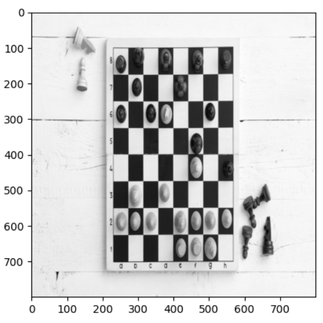
3 Main Approach

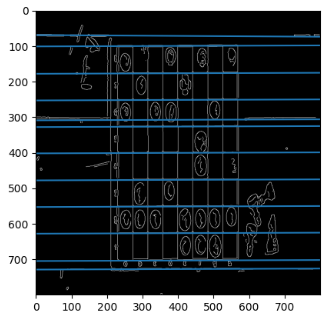
The main approach consists of 3 steps. First the board is located in the picture. Then the grid is identified and the location of each square is provided. Finally, the area identified for each square is feed into a pretrained ResNet18 that is fine tuned on chess piece images.

3.1 Locating the Board and Squares

The first step in processing the image is to find the board. To do this images were first resized and greyscaled so that they could be processed quicker. Additionally, Canny Edge Detection was applied to them so that Hough Lines Transform could identify lines better. To find the board Hough Lines Transform is applied twice to the image. First it tries to find the best 16 horizontal lines in the image and then picks the lines with largest y-value average and lowest y-value average. It then does the same same but with only vertical lines. These lines are then put together to find the board outline.


(1) Original Image


(2) Gray Scaled Image


(3) Horizontal Lines

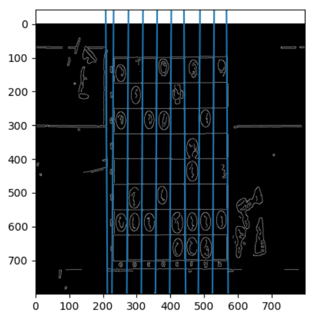

(4) Vertical Lines

Figure 3: Board Finding Process

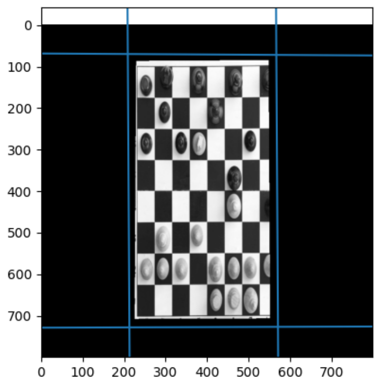
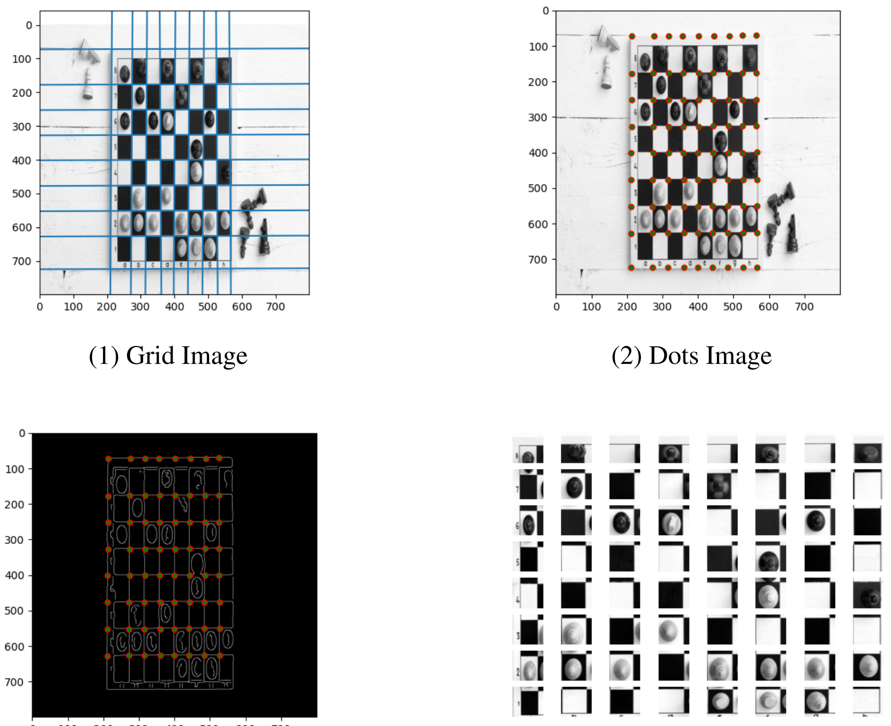


Figure 4: Board Finding Process Cont.

Now that the board is extracted, the rest of pipeline deals with just the board. A similar process is now repeated to find the grid lines. This time the best eight horizontal and the best eight vertical lines found by Hough Line Transform composes the grid. The pairwise intersection of the horizontal and vertical lines is then computed – call these the dots. The intersection of the lines except the final row and final column is then computed, call these the corners. For each corner its closest two neighbors (below it and then to the right of it) are computed from the dots. The closest neighbors are then used to give the coordinated for each piece.



(1) Grid Image

(2) Dots Image

(3) Corners

(4) Separated Piece Images

Figure 5: Piece/Square Finding Process

The images are now ready to be feed into the ResNet18! More precisely, the corresponding portions of the original colored image are fed into the neural network model to identify each piece.

3.2 Training the ResNet18

The model utilized a technique known as Transfer learning, which is the improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned.

The convolutional neural network ResNet18, was pretrained on ImageNet trained on 1000 different classes. For our implementation we utilized feature extraction with this model to modify Resnet to work with our specific needs. The model had its final layer removed and replaced with a linear layer with a input size of 512 and an output size of 13, an output for each possible piece(White knight, white king, white bishop, White pawn, white rook, White Queen, Empty, Black Knight, Black King, Black pawn, Black rook, Black Queen, Black bishop) and finally a softmax is applied to all the outputs. During the training all the ResNet18 parameters were frozen during fine-tuning, and only the last linear/softmax layer was tuned.

We used a batch size of 8. The model was optimized against cross-entropy loss for 50 epochs, with a learning rate of $\alpha = 0.001$ and a momentum of 0.9, using stochastic gradient descent.

The training data was split at a 70%-30% ratio into training and validation sets. The training sets had the inputs resized to 224 by 224, and then was randomly cropped, and normalized. The validation data was similarly normalized to 224, cropped, and then normalized. The test data was used to train, while the validation dataset was used to fine-tune hyperparameters such as the batch size and learning rate, as well as to monitor accuracy. The model which had the highest accuracy on the validation data was selected to be saved.

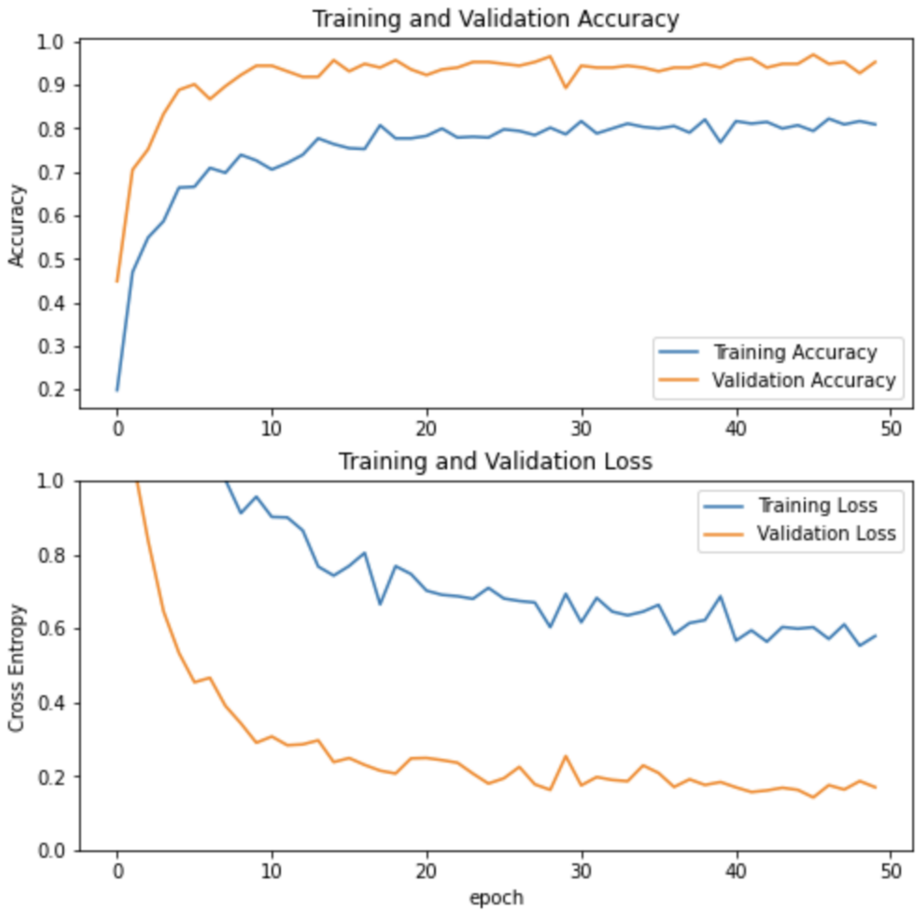


Figure 6: Training and Validation loss, accuracy

The reason the validation accuracy was higher than the training accuracy (see figure 6) was likely because of the harsh regularization methods we used, dropout, as well as random resize crop and random horizontal which makes some training images much harder to predict, since a random portion of the training image is zoomed in.

4 Evaluation Metric

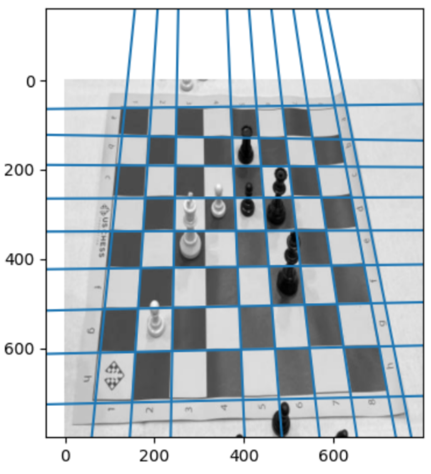
We evaluated the chessboards on the test split of the Roboflow test dataset. First, we calculate the board accuracy, which is the number of chessboards that were correctly identified, along with the fraction of gridlines correctly identified. Most chessboards were correctly identified, but the few that weren't were simply manually picked out (the accuracy for this section was not automated).

After this, for the chessboards which were correctly identified, we ran the model on each square of the chessboard. For each square, the model classifies the square based on the color and type of the chess piece, or classifies it as unoccupied. Our reported normalized accuracy is the total accuracy of the model on all the $64 \times 26 = 1664$ squares of the chessboards that were correctly identified. We also report the accuracy out of $64 \times 28 = 1792$ – the unnormalized accuracy – which represents the fraction of squares correctly predicted, acting as though every square for the chessboards that we didn't see was wrong.

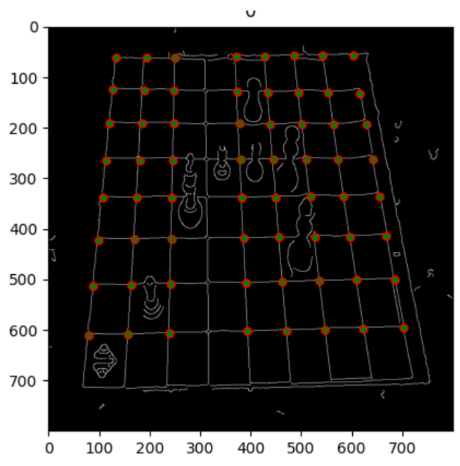
5 Results & Analysis

5.1 Piece Extractions

Piece extraction was tested using 28 images from the Roboflow test dataset. This dataset consisted of full board images with different board positions. The board analysis pipeline identified 26/28 of the boards 100% correctly. Additionally, it correctly identified 502/504 of the grid lines from all 28 images correctly. This gives it a board accuracy of 92.87% and a grid line accuracy of 99.9960%. The picture below shows one of the images it messed up on. To highlight where it failed look at the "corners" and "grid" it produced.



Grid Image (Failure)



Corner Image (Failure)

Figure 7: Failed Image Analysis

The grid picked up on the rightmost edge and this ultimately messed with the final corners output. This is because the board wasn't correctly picked up. This is caused by the outline of the board getting picked up instead of the actual chess grid. To fix this a greater offset from the right edge was needed but this changes with each image. Another option is to require the lines to be similar distance from each other. This would be a feature to add in the future.

5.2 Piece Identification

With a board correctly split into squares, we individually input labels for each detected square, then running the squares through the model, we were able to get the prediction accuracy of the model on our boards. With the boards that we labeled and run through the models we had on a an average accuracy of around 63%

6 Conclusion/Future Work

Conclusion: Although our piece recognition and the line detection was very promising, the predictions of our model was slightly disappointing. This performance could be attributed to a few factors. One issue that we had was that the original model was trained on color images and the images that we trained on and tested on were grey scaled. Furthermore, in order to capture the entirety of images, we cropped squares to be larger than we detected, however those cropping could have been optimized.

Future Work In future iterations of our work, we could utilize a wider database of images. The images that we used in the training mostly came from two different chessboards, and we are unsure how well the model would generalize to other boards. For the grid detection, we could also utilize automated checks to ensure that the spacing between grid points are even. Furthermore, we could have made our model a little more expressive with a more complex final layer, utilizing multiple layers of MLP or adding additional non linearity. Also as previously stated from above, we could try to train the model with color images and also experiment with different margin expansions to achieve better results.

References

Bottou Yoshua Bengio Yann LeCun, Patrick HaffnerLeon. Object recognition with gradient-based learning. `https://doi.org/10.1007/3-540-46805-6_19`, 1999.

Ognjen Arandjelović Wölflein, Georg. Determining chess game state from an image. `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8321354/`, 2021.

Ruiyang Hu Geoffrey Kao. Recognizing chess pieces using an object ensemble approach. `https://github.com/gkao123/Chess-Piece-and-Board-Recognition`, 2018.

Roboflow. Chess pieces dataset. `https://public.roboflow.com/object-detection/chess-full`, 2021.