CSCI 1430 Final Project Report: Chess Recognition

L.A.R.D: Long Do, Andrew Cooke, Richard Hill, Danielle Rozenblit.
Brown University
10th May 2020

1. Introduction

Our goal was to recognize the state of a chessboard from an input image and to recommend a move for each player based on the current board state. Recognizing the board state is a difficult vision problem to solve because it involves correctly labeling chess board pieces as well as knowing the location of those pieces on a chessboard.
Our approach to this vision problem involved separating it into two sub-problems. First, we wanted to recognize the structure of the board. This involves recognizing the bounds of the board in the image as well as the coordinates of the individual board squares. We then planned to identify the pieces on these individual board sections using a trained CNN. These steps together would allow us to both label the pieces and determine their position on the boards.
Once we have the board state recognized, we planned to pass this information into a chess API in order to return a move recommendation.

2. Related Work

Line Detection Process: For finding the 9x9 grid of a standard chess board we followed loosely Buchner's implementation of Gaussian, Canny, and Hough Lines. Proceeded by a calculation of intersections and conversion from polar to euclidean coordinates. [1] VGG16 Architecture: For the piece recognition aspect of our project, we used the VGG16 architecture [2] with weights trained on ImageNet.
Training Images: In order to train our model to classify chess pieces, we used a collection of training and testing images by D. Yang on GitHub. [4] These were labelled images of individual chessboard slices, which was exactly what we needed in order to train our model. Stockfish Chess Engine: In our attempt to use stockfish [3], we downloaded the current version but did not end up using the engine. We did not think our results were consistent enough to pass through an engine and as an end goal, we decided to focus elsewhere. In the future, we hope to implement this stockfish API in order to predict the best move.

3. Method

In order to recognize the state of a chess board, we split the problem into two parts:board recognition and piece recognition. Board recognition involves recognizing the location of the board as well as the location of the individual board squares. Piece recognition involves taking in an individual board square and recognizing what piece is on that square (ie: white queen, empty, etc.)

3.1. Board Recognition Method

We assume that all images are going to be taken from a birds-eye view. We take in an image of the chess board and then convert the image to grey scale. We blur by convolving with a Gaussian filter and input the resulting image into canny line detector. We then take this image of just lines and input that into Hough Lines, which returns all lines of a certain length in polar coordinates. [1]

As this method both misses lines and detects lines that are not a part of our board, we process our lines to have two groups of 9 roughly parallel lines that outline each square of our board. We first use KMeans to cluster our lines into two groups by angle. Then, we find the median of distances between lines in each group and look for spaces that are multiples of the median. If found, we add a line in the gap with an angle between that of its neighbors. Next, we look to remove a line that is not spaced out by the median, and if we do, we repeat this process. Lastly, and this is the step where our code sometimes messes up on more difficult boards, if we are missing an edge we fill in the last edge based on a few factors. This code is very consistent in recognizing boards from birds-eye views with all lines vertical and horizontal. The farther we deviate from this camera angle, the more likely it is that the system makes a mistake, but it is still very often successful.

Taking all the lines we solved simple systems of equations to find each intersection between the lines. This process results in an array of 81 (x,y) coordinates representing each corner of the board.We use these coordinates to cut up our image into 64 images of board squares to pass into our piece

recognition code.

3.2. Piece Recognition Method

In order to recognize board pieces, we decided to use a CNN classification model in order to recognize individual board squares. We used a data set created by Daylen Yang found on GitHub [4] in order to train and test our model. Initially, we experimented with designing and training our own various architectures with fewer than 15mil parameters. However, the accuracy achieved with these architectures was below 0.8000. This accuracy is not terrible for individual board sections, however, this corresponds to a best-case scenario $6.277 * 10^{-7}$ board accuracy (the probability of guessing all 64 sections correctly), which would result in a practically non-functional board-recognizer.
We then decided to experiment with transfer learning in order to achieve higher accuracy. We used the VGG16 model architecture [2] with weights that were pre-trained on ImageNet. This ended up having far better accuracy than our original models. After experimenting with different classification head architectures, we decided on the following design :

Layer (type)	Output Shape	Param #
block1_conv1 (Conv2D)	multiple	1792
block1_conv2 (Conv2D)	multiple	36928
block1_pool (MaxPooling2D)	multiple	0
block2_conv1 (Conv2D)	multiple	73856
block2_conv2 (Conv2D)	multiple	147584
block2_pool (MaxPooling2D)	multiple	0
block3_conv1 (Conv2D)	multiple	295168
block3_conv2 (Conv2D)	multiple	590080
block3_conv3 (Conv2D)	multiple	590080
block3_pool (MaxPooling2D)	multiple	0
block4_conv1 (Conv2D)	multiple	1180160
block4_conv2 (Conv2D)	multiple	2359808
block4_conv3 (Conv2D)	multiple	2359808
block4_pool (MaxPooling2D)	multiple	0
block5_conv1 (Conv2D)	multiple	2359808
block5_conv2 (Conv2D)	multiple	2359808
block5_conv3 (Conv2D)	multiple	2359808
block5_pool (MaxPooling2D)	multiple	0
flatten (Flatten)	multiple	0
dense (Dense)	multiple	1605696
dense_1 (Dense)	multiple	975

Figure 1. Model Architecture

We additionally experimented with various hyperparameters and had the best results with 20 epochs, a batch size of 10, a learning rate of 1e-4, and some image preprocessing / data augmentation.

3.3. Finalization Process

We then put all the steps together. We used the board recognition process on an input image and used this method in order to split up the original image into 64 individual squares, each one representing a different square on the chess board. We then preprocessed these images in order to properly prepare them for the vgg16 model. We used the saved model from training in order to quickly load the model weights and results and predict a classifier for each of the 64 individual squares. Finally, using the predictions, we compiled the array into a string representing a chess board which we print to the terminal.

4. Results

4.1. Board Recognition Results

Here is an example of our detected lines without processing. It outlines the board well, but it misses a few lines, has some overlapping lines, and detects the edges of the board as well. Our results with preprocessing are significantly better.

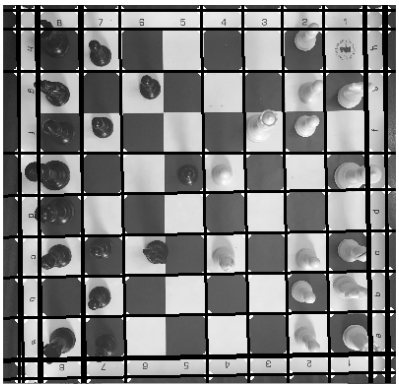

Figure 2. Line Detection Without Preprocessing

In this fully recognized angled board, we take our 18 lines and draw circles at each of their intersections.

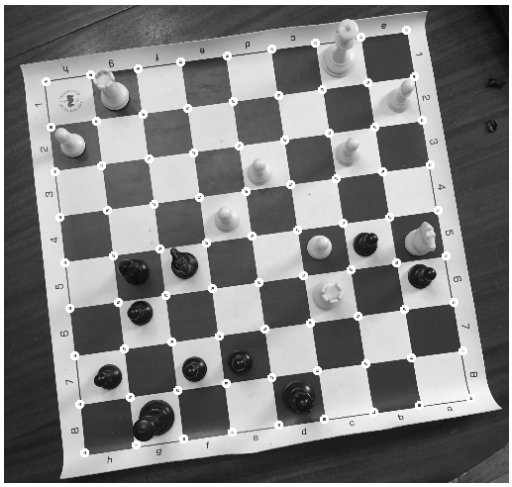

Figure 3. Line Detection on Angled Board

4.2. Piece Recognition Results

Our final piece recognition model was able to achieve a very high accuracy of 0.9946 on the test images, which translates to around a 0.7071 expected accuracy for board detection / for correctly classifying all 64 board sections. This accuracy was achieved using the VGG16 model architecture and weights that were pre-trained on ImageNet. This accuracy is much higher than the 0.8000 accuracy achieved when we attempted to design / train our own architecture.

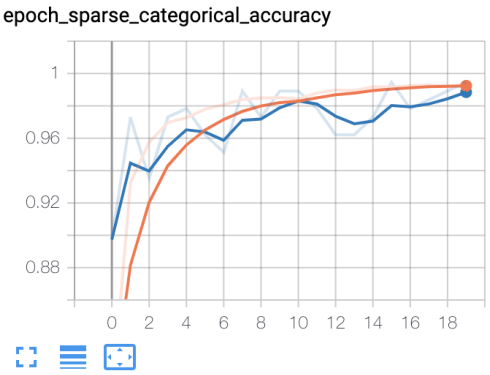

Figure 4. Piece Recognition Accuracy

With the following image as input, this was our predicted board representation and the correct board representation:

Correct Board Representation:
"7q/KP6/1Q3P2/3p4/p1p5/6p1/4Rpp1/r4rk1/"
Our Board Representation:
"7k/KB6/1B3B2/3p4/pNp5/6b1/4Rpp1/r4rq1/"
Our representation is fairly close, but there are several differences.

4.3. Discussion

First, we will discuss the representation of the board. Each representation consists of 8 different strings, each representing a rank of the chess board and divided by a forward slash. For each rank of the chess board, the representation is as follows:
A "p" represents a black pawn
A "q" represents a black queen
A "k" represents a black king
A "b" represents a black bishop
A "r" represents a black rook
A "n" represents a black knight
A "P" represents a white pawn
A "Q" represents a white queen
A "K" represents a white king
A "B" represents a white bishop
A "R" represents a white rook
A "N" represents a white knight
An integer represents that number of empty squares. For example, an empty rank will be represented as /8/

Second, we will discuss the issues with our results that we would like to address. As seen in the piece recognition results, there is a trend of getting specific pieces messed up. Often times, bishops and pawns are confused for one another and the same is true for kings and queens. Additionally, there are errors with other pieces getting confused for one anther but this are not as common as these usual errors. We believe that the reasoning behind this is similar structure between the pieces. From a birds eye view, pawns and bishops are near indistinguishable from one another and the same goes for kings and queens.
We also believe that much of the confusion came from the board itself. We found that if the image had too large or cluttered of a background, the line detection function we used could have errors. In addition to this, if lighting on the board was stronger in one area than another, it was much harder to detect the correct pieces in glare. Finally, the board we used was slightly different from the board used in the training images. Thus, when predicting images from our board, we did not get as consistent results as when testing with the date found online. These bugs could be fixed by improving the line recognition method and taking the time to create our own data instead of using images found online.

We found that the orientation of the image mattered immensely, and our first implementation of using only vertical and horizontal lines often created errors when slicing the image up into 64 different squares. Additionally, the validity of the result vastly depended on the centering of each piece on its respective squares and the shadows created by the lighting in the image. We fixed this by using images with well centered pieces and good lighting, but

again, using our own data would possibly improve these results as well as adding more heavy image preprocessing before predicting results.

Currently, our board representation result is usually around 70-80 percent accurate. This is a pretty good result and though there is room for improvement, we are hoping we can improve this in the future.

5. Conclusion

Overall, we were able to create a program with a fairly high accuracy to predict the board representation of a given chess position. We think this is an applicable project and eventually, anyone who played chess could use this as a method of finding the best move from a chess game being played in person.

Goals for the future:
We were all excited about this idea, and wish to improve it in the future. First, we would like to create a data set more viable for the board that we used and work to make the result more consistent with the input image. If possible, when deciding on an idea we hoped to be able to find a board representation from an input image not only from a birds eye view but also from other angles. Though this now seems like a much more difficult goal after completing this project, we think it would still be a cool project. Additionally, we would like to eventually get a consistent enough board representation to pass it into the stockfish [3] API without error or confusion.

References

[1] M. Buchner. Chessboard picture recognition project, 2011. http://codebazaar.blogspot.com/2011/08/chess-board-recognition-project-part-1.html. 1

[2] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition, 2015. 1, 2

[3] M. C. Tord Romstad and J. Kiiski. Stockfish chess engine, 2020. 1, 4

[4] D. Yang. chess-id, 2016. https://github.com/daylen/chess-id. 1, 2

Appendix

Team contributions

All of the team members worked to finish the report.

Danielle Rozenblit Danielle was mostly involved with the piece recognition aspect of the project. This involved training a CNN to classify individual board squares. She designed multiple different CNN architectures, experimented with parameters and eventually decided to use the vgg-16 with pre-trained weights. She experimented with various classification heads for this model.

Long Do Long focused on the board recognition part of the project. He experimented with various filters, algorithms, and parameters for detecting the 9x9 grid that makes up a chess board. Some attempts include Otsu's method and square detection through color contrasting. He settled on a process that involved turning the image to grey-scale, blurring it with a Gaussian filter, running Canny line detector, and then Hough lines. He plotted all lines that were near vertical or horizontal and implemented a method to find all intersections. These corners are then used for splitting up the image into 64 squares.

Richard Hill Richard worked on the second half of board recognition. He processed the lines to find those missing and remove lines not on the chessboard and overlapping lines. Our initial version of the code only detected horizontally and vertically aligned boards, so he also used KMeans to group the lines so that angled boards could also be detected.

Andrew Cooke Andrew was involved with the finalization process and figuring out how to split the original image into 64 separate squares in order to predict a result. This involved working with Richard and Long in making sure that the intersection points matched up with the original image and that each individual image was valid for predicting. He also worked with Danielle in creating a valid model to use with predicting. Together they were able to save and load a model which was used for predicting a final board representation. Next, he preprocessed each of the 64 different squares and created a valid data set to be passed into the model. After passing these images into the model, he using the predicted values in order to format the board representation string.