Chess Game Tracking Using Computer Vision and Deep Learning

Jerry Liu May 29, 2019

Abstract

This project involves the tracking of a chess game using a high angle camera. It is divided into two phases: board detection and piece classification. A convolutional neural network (CNN) is used to classify pieces according the categories of empty, pawn, knight, bishop, rook, queen, and king. When the camera detects the board, a frame is extracted, and each one of the 64 squares are cropped out. From here, a convolutional neural network processes each square to classify each piece in the image, and then the program predicts the board. This process will allow the program to monitor a chess game move by move. This program is intended to allow the automatic recording of chess games during tournaments.

Contents

1. Introduction

During chess tournaments, players have to manually record each move played. This process is a tedious and time consuming task. There are specialized chess sets that automatically record each move played, but these chess sets are expensive. A program that tracks each game using computer vision is an inexpensive solution.

Previous work on chess game tracking has used an overhead view of the chessboard. This, however, creates a problem when distinguishing between similar pieces, such as between pawns and bishop. For my project, I will use the camera to view the chessboard at an angle so that pieces are more easily distinguished from one another. In addition, previous projects on chess game tracking use chess sets with specialized color schemes (e.g. red pieces and green pieces. In my project, I generalized to generic white and black chess sets.

My program is meant to track a chess game from start to finish using computer vision and deep learning. Given a picture of a chess game, the program should be able to tell on which square each piece is located, and it should distinguish what type each piece is. The piece categories I employed were empty square, pawn, knight, bishop, rook,

queen, and king. In my project, I used the OpenCV and Keras Libraries and the Pickle Module. In the future, chess tracking programs will allow the creation of autonomous chess-playing robots equipped with computer vision.

In my project, I viewed the board from a 45 degree angle. My project has two parts: board detection and piece classification. For board detection, I used the OpenCV library. To classify each piece, I trained a convolutional neural network (CNN) to distinguish pieces from one another using the Keras library. I collected images of chess pieces from Google Images and later used manual collection to construct my database. I used the Pickle Module to store my data into pickle files so that I could train my CNN.

Given an image of the chessboard at an angle, I used the far corners to create a perspective transform of the picture so that the board becomes a square, centered in the image. After this, I cropped out the 64 squares and ran these images through my classifier to predict each square. Finally, my program predicted the current position on the chessboard. This project is in a field I have had no experience in before, so I had to learn many new skills during my time working on this project.

2. Methods

Libraries

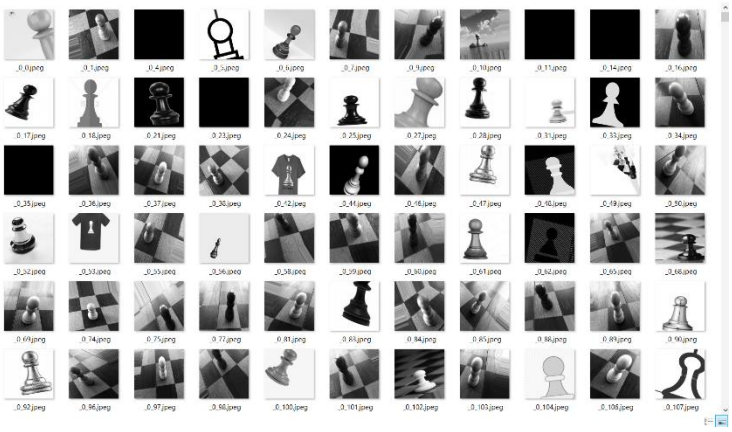
In my project I used the OpenCV and Keras libraries. I used OpenCV for board detection. I used Keras for training my piece classification convolutional neural network and saving it for use by my classifier. Furthermore, I used the Pickle Module to compact my data from folders of images into two pickle files: one for the images, the other for the labels.

Data Collection

For the collection of online pictures in my project, I searched phrases such as “pawn chess” or “knight chess” in Google Images and scrolled down the page until I reached the cap of around 500 images. Then, I used a JavaScript program to scrape the URLs of all the images on the page and composed a text file of the urls for each piece. Next, I used a Python script using each text file to download the images into folders corresponding to each piece. From here, I filtered the bad images from the data, resulting in cutting down the number of pictures in each category to 150-250 images per folder.

Partway through my project, I realized my pictures from Google Images were not representative of the pieces my camera would detect because I intended to view the board at a 45 degree angle. Thus, I manually collected 200 sample images of each piece at angles from between 30 and 60 degrees. After that, I standardized each picture to 100 x 100 grayscale jpeg files. From here, I used Keras to augment my data using random distortions and rotations to around 5000 to 6000 pictures for each piece.

Figure 1 (Sample of Training Data)



Data Compaction

Once I had collected my data and saved it into folders, I used the Python Pickle Module to compact my images into a single Pickle file, which I named x.pickle. Also, I assigned a label to each category of piece (empty = 0, pawn = 1, knight = 2, bishop = 3, rook = 4, queen = 5, king = 6). I created a y.pickle file of these labels corresponding to each respective image in x.pickle. To clarify, the nth element of y.pickle is the label of the nth element of x.pickle.

Convolutional Neural Network

In my project, I used a convolutional neural network (CNN) to distinguish the pieces from each other. A convolutional neural network is a type of deep neural network, most commonly applied to analyzing pictures and visual data. It uses hidden layers of convolution filters, or kernels, to extract features from an image, which are edges and various patterns.

For my convolutional neural network, I used 6 convolution layers with 5 dropout layers in between. My first dropout layer was at 20%, with each successive one increasing by either 5% or 10% until it reached 50% for the last one. Each successive convolution layer had twice the neurons of the one before it since the max pooling in each convolution step reduces the length of each side of the picture by half. This is followed by a fully-connected layer with 1000 neurons and then another fully-connected layer of 7 neurons to categorize each piece. I used the RELU activation function for each convolutional layer and the first fully-connected layer, and I compiled with the categorical cross-entropy loss and the Adam optimizer. For the final classification layer, I used the soft-max activation function.

I trained my CNN remotely to the Duke Computer because it has a strong GPU and would shorten the training time.

Once I finished training my CNN I saved it to a model called chess.h5. Then, I used this model to create a piece classifier program.

Figure 2 (CNN layers)

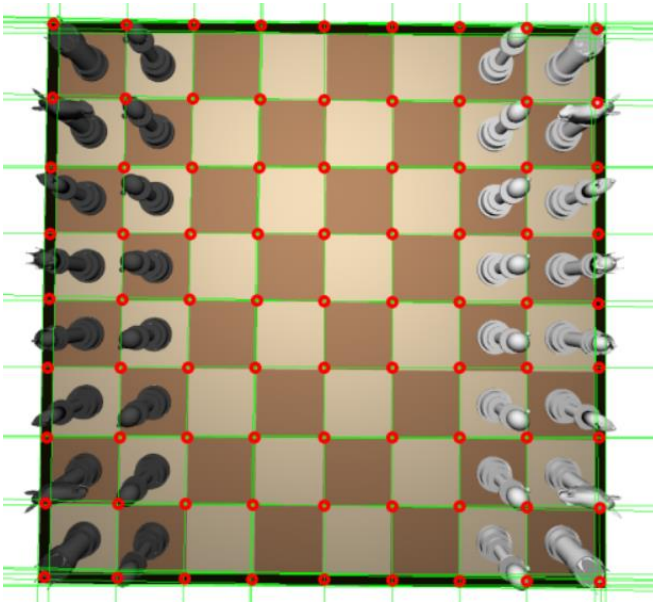
Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 100, 100, 32)	896
max_pooling2d_1 (MaxPooling2	(None, 50, 50, 32)	0
dropout_1 (Dropout)	(None, 50, 50, 32)	0
conv2d_2 (Conv2D)	(None, 50, 50, 64)	18496
max_pooling2d_2 (MaxPooling2	(None, 25, 25, 64)	0
dropout_2 (Dropout)	(None, 25, 25, 64)	0
conv2d_3 (Conv2D)	(None, 25, 25, 128)	73856
max_pooling2d_3 (MaxPooling2	(None, 12, 12, 128)	0
dropout_3 (Dropout)	(None, 12, 12, 128)	0
conv2d_4 (Conv2D)	(None, 12, 12, 256)	295168
max_pooling2d_4 (MaxPooling2	(None, 6, 6, 256)	0
conv2d_5 (Conv2D)	(None, 6, 6, 512)	1180160
max_pooling2d_5 (MaxPooling2	(None, 3, 3, 512)	0
dropout_4 (Dropout)	(None, 3, 3, 512)	0
conv2d_6 (Conv2D)	(None, 3, 3, 4024)	18546616
max_pooling2d_6 (MaxPooling2	(None, 1, 1, 4024)	0
dropout_5 (Dropout)	(None, 1, 1, 4024)	0
flatten_1 (Flatten)	(None, 4024)	0
dense_1 (Dense)	(None, 1000)	4025000
dense_2 (Dense)	(None, 7)	7007

Total params: 24,147,199
Trainable params: 24,147,199
Non-trainable params: 0

Board Detection

I used the canny edge detector and Hough transform algorithms in the OpenCV library to detect the edges and lines of the chessboard. For this process, I used the OpenCV library functions cv2.canny and cv2.houghLines. Next, I separated the lines between horizontal and vertical lines, and I calculated the intersections of these lines, finding the corners. I then took the average of corner points that were counted more than once to make sure there is one corner detected for each intersection. This method of corner detection allows me to find corners obstructed by pieces, which is something the OpenCV function cv2.findChessboardCorners cannot do.

Figure 3 (Line and corner detection)



Perspective Transformation

Since my camera is angled relative to the board, I used a perspective transformation to reorient the board as a square centered in the image. To do this, I first found the four far corners of the chessboard, and then I calculated the lines and corners of the new image again. Now, I can crop out each square of the board.

Figure 4 (Perspective Transformation)

Piece Isolation and Classification

Once I had cropped out the 64 squares of the chessboard, I ran each through my classifier. Next, I used my classifier's predictions to construct the predicted board.

Figure 5 (Cropped squares and a prediction)

Setup

I put the chessboard on a table, with my camera mounted at a 45 degree angle from the normal to detect the board.

Figure 6 (The angle at which the camera will view the board)

3. Results and Discussion

Piece classification

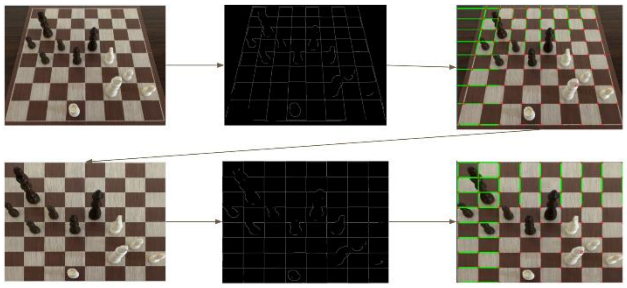
So far, my convolutional neural network has a 92% accuracy of classifying pieces between each other (Keras validation accuracy). Given pictures of isolated chess pieces, it generally correctly categorizes the images. However, given cropped images of chess pieces, it has trouble classifying do to obstruction. As a result, it has a low accuracy when classifying pieces from actual pictures of chess games.

Figure 7 (Sample output)

Board Detection

I am able to find the lines and corners of a chessboard. My method works even if a corner is obstructed, unlike the OpenCV function cv2.findChessBoardCorners (). However, I have to adjust my thresholds for the canny edge detector and Hough transform when given different pictures. I used the far corners to calculate the perspective transformation, and after transforming the picture, I calculated the lines and corners again. This allows me to crop out each square of the chessboard.

Figure 8 (Visualization of perspective transformation)



Problems

In my project I had issues with piece obstruction and having multiple pieces being shown in a single square. This is a logical consequence to viewing the chessboard at an angle instead of directly overhead. This problem has resulted in inaccurate classification for the piece in each square.

Figure 9 (Piece obstruction)

Furthermore, my neural network is rather inaccurate when being tested on actual pictures of chess games. Thus, it requires fine tuning and perhaps an increase in training data.

Unfinished Work

I was not able to distinguish pieces by their color because I ran out of time. However, this idea should be simple as I can check the intensity of the pixels of each piece to distinguish them by color.

4. Conclusions and Open Questions

My project successfully classifies isolated pieces in images. It is also able to detect the corners of chessboard pictures, despite obstruction by pieces. For future projects, maybe it will be better to compose a database of pieces from an overhead camera and track chess games from directly overhead so that pieces will not obstruct each other.

5. Acknowledgments

I would like to acknowledge my Senior Research Lab director, Dr. Zacharias.

References

- OpenCV Website
- Keras Website