A PROJECT REPORT ON

Chess AI using CNN

SUBMITTED BY

Fenil Mehta	Roll No. 203050054
Aditya Jain	Roll No. 203050003
Amit Hari	Roll No. 203050115
Munna Kumar Paswan	Roll No. 203050120

CHAPTER 1: INTRODUCTION

1.1 MOTIVATION

Using Artificial Intelligence to teach programs to play games has grabbed the attention of many researchers over the past decades. Games that are in focus are particularly board games such as Chess, Go and Shogi. Chess is popular and the most played board game and it is interesting from an Artificial Intelligence perspective.

Board game engines developed so far are based on lookahead algorithms. For any board game engine it is difficult to make optimal moves with few computations and limited computation power. In this project we are investigating Deep Learning algorithms that can make computer programs play chess without using lookahead algorithms and play at par with highly ranked players.

Basic idea behind not using lookahead algorithms to generate optimal moves is, for a given board position, the program should give the next optimal move by exploring current board state. This means the program should be able to maximize the winning chances without evaluating lots of future board states.

1.2 PROBLEM DEFINITION

Develop a Convolutional neural network which learns to classify chess board positions and predicts the optimal move without any look ahead.

1.2.1 Objectives

1. Using preprocessed dataset of chess, convert the FEN notation of the chess boards to encoded bits representation for the board and do one-hot encoding of the centi-pawn score which is real-valued.
2. To train neural network models to predict the next move.
3. To measure the accuracy of convolutional neural networks.

1.2.2 Input

Preprocessing Module

1. Processed kingbase dataset has been used for training which is a CSV file having FEN (i.e. Forsyth–Edwards Notation) string of chess board and the normalized centi-pawn score generated using Stockfish 10.

Training Module

1. Preprocessed CSV files containing chess board states as a matrix of binary numbers along with its one-hot encoded centi-pawn score.
2. Error from the comparison of the result of the neural network prediction and one-hot encoded centi-pawn score generated by the preprocessing module for weight adjustment.

Game Play Module

1. Current chess board positions

1.2.3 Output

From the preprocessing module:

1. Matrix of zeros and ones

After Training:

1. Trained neural network.

While playing the game:

1. Predicted optimal move by the agent.

CHAPTER 2: PROJECT DETAILS

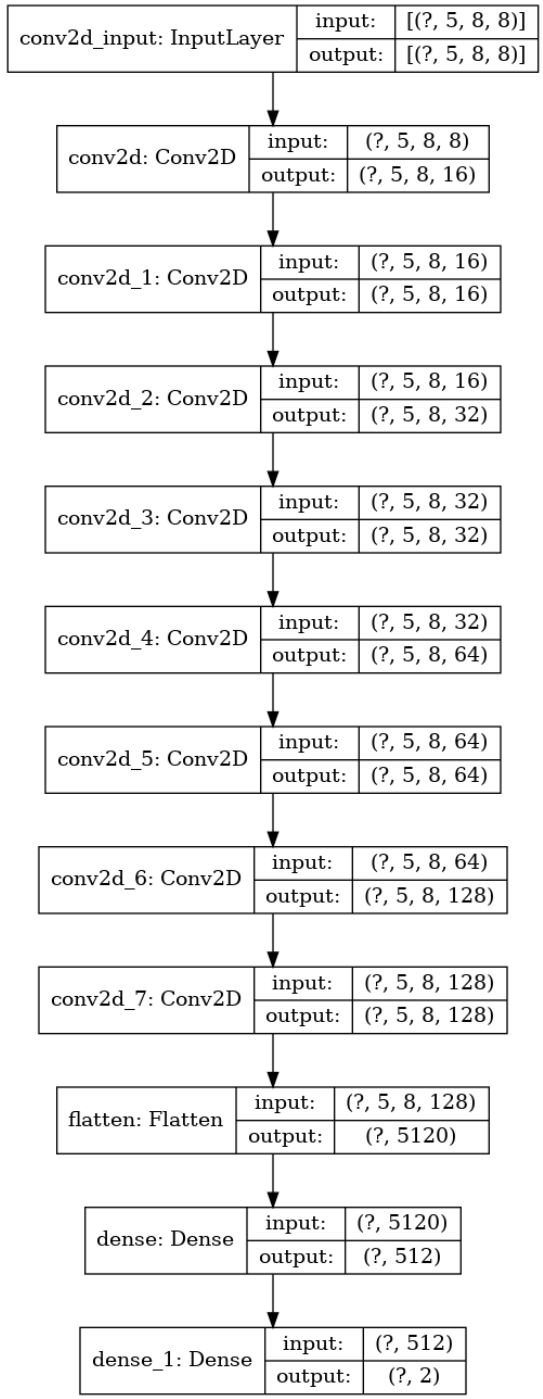
2.1 Preprocessing Module

- The primary role of this module is to make the input dataset ready for giving them as input to the CNN.
- It takes the processed CSV files as input and converts them to a matrix of zeros and ones using the below technique:
    - In a matrix of 8*8, For each piece in the chess board, assign a unique number to it starting from 1 using the following mapping: {"P": 1, "N": 2, "B": 3, "R": 4, "Q": 5, "K": 6, "p": 9, "n": 10, "b": 11, "r": 12, "q": 13, "k": 14} where uppercase letters denote the pieces of white side and the lowercase pieces denote the black side pieces. “P” denotes pawn, “N” denotes knight, “B” denotes bishop, “R” denotes rook, “Q” denotes queen, “K” denotes king. Similarly for the black side as well.
    - If king or queen side castling is possible by white player, then use 7 and for the black player we use 15.
    - For en passant moves we use 8.
    - We convert the 8*8 matrix to a 5*8*8 matrix where each number from 1 to 15 is converted to binary and the bits are distributed in the first 4 layers.
    - The 5th layer is all “1” if it is white players turn, otherwise “0”
- Two different Score Normalizers (SN) have been used:
    - SN4,5 and SN6 convert the range [-1, 0) to category “-1” and [0, 1] to category “1”. Later these categories are converted to one-hot encoding and then used for training. However, the only difference between SN4,5 and SN6 is that the latter creates two unused categories (i.e. in total, SN6 will have 4 target categories) to be compatible with the CNN model v7 architecture.
    - The second normalizer converts the range [-1, -0.0001) to category “-2”, [-0.0001, 0) to category “-1”, [0, 0.0001] to category “1” and (0.0001, 1] to category “2”. Later these categories are converted to one-hot encoding and then used for training.

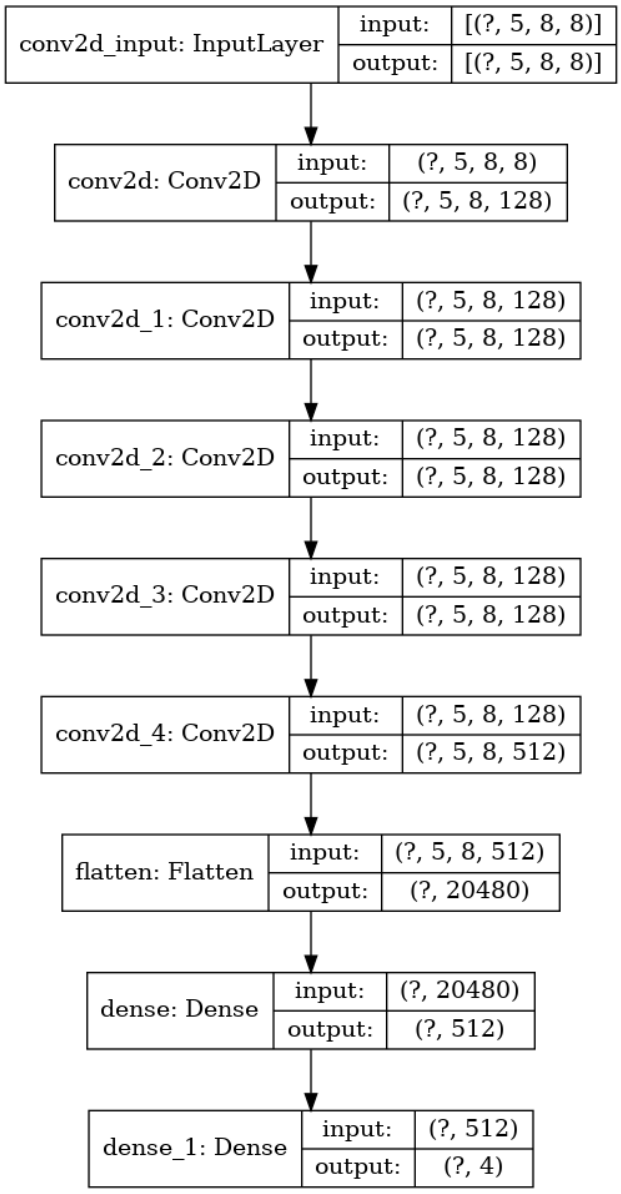
2.2 Training Module / Convolutional Neural Network Architecture

- Multiple architectures (i.e. MG = Model Generator) were tested but two were more effective probably due to their large size. Note that each MG denotes a unique model architecture.

CNN model v6 - MG6



CNN model v7 - MG7



2.3 Game Play Module

- It is made up of ChessPredict and ChessPlayCLI
- ChessPredict class provides an abstraction to the ChessPlayCLI class to predict an optimal move using a neural network or a game engine based on the parameters with which the ChessPredict object was constructed.
- ChessPlayCLI interacts with ChessPredict and the user to provide a game play interface.

CHAPTER 3: RESULTS, CONCLUSION, FUTURE WORK

3.1 RESULTS

- All the training data is from white's perspective hence the model tends to play better as white (1st player) as compared to black (2nd player).
- When the game starts, the model is able to play decently. However, considering the sparseness when chess boards are converted to binary form and the humongous possibilities of the different board positions for input to the network, exploratory search is necessary as the game proceeds.
- In order to evaluate the performance of the network, we used the model to predict the chess moves on the Kaufman special dataset of 25 complicated board positions, and the best model was able to predict the expected move for only one board position.
- Accuracy scores for Model:

MG6, SN4,5	**-ve**	**+ve**
-ve	0.387	0.009
+ve	0.591	0.013

MG7, SN6	**-ve**	**+ve**
-ve	0.000323	0.366553
+ve	0.000418	0.632607

MG7, SN7	**-ve**	**+ve**
-ve	0.349	0.017
+ve	0.605	0.027

3.2 ADVANTAGES

- The proposed system will provide an efficient way of selecting a possible best move for the current board position, as compared to the traditional approaches like state space or lookahead algorithms which take a lot of computational time and power.

- The proposed system makes use of deep neural networks, which enables the system to continuously learn through each game played. This improves the system to play games better. And, as the model gets trained on more games, the decisions improve.

3.3 LIMITATIONS

- The preprocessing involves the training of the model on a very large number (millions) of games. This training of the deep neural network will require huge computational power and a lot of time for training. Thus, there is a need for a high performance GPU as well as time for training the model.
- The accuracy of the system is dependent on the reliability of the Stockfish engine's accuracy.
- The strength of the system depends on the quality of the games it is trained on. Thus, more high-quality data results in better predictions.

3.4 CONCLUSION

- In this work, we discussed a chess engine, which avoids the use of state space search to find the next optimal move.
- It only relies on the trained network to select the most favourable move after being trained on millions of games played between players. This work tries to show that for board games like chess, there should be an optimal move for each board state.
- However, we cannot say that there is no need to perform exploratory search for every request to the model or the engine to play or predict an optimal move.

3.5 FUTURE WORK

- The system is developed by using only the processed CSV files for the game of chess which contains the game's board state's evaluated scores generated using Stockfish 10. Thus, the model can be ported to other board games as well, requiring only their processed CSV files generated using a board evaluation method.
- As the system is partially aware of good moves and bad moves. Hence, using an optimized tree search and self-play can help enhance and fine tune the board evaluations. The system can also be extended to learn from games played against human players, or against any other game engines.

Code link: https://github.com/fenilgmehta/Chess-Force-CNN
(This is a private repo as of 6th Dec 2020)