Evaluating Chess Positions using Convolutional Neural Networks

Fabian Krause
Department of Computer Science
Universität Freiburg
`krausefa@mail.utoronto.ca`

Eric Wang
Department of Electrical and Computer Engineering
University of Toronto
`ericdamon.wang@mail.utoronto.ca`

Abstract

This paper discusses the usage of a convolutional neural network (CNN) to evaluate chess positions by training on a heuristic function, here Stockfish evaluations. While the model shows some ability to analyze the board in a general sense, its performance is limited. Nonetheless, this work demonstrates that CNNs can approximate a hand-crafted heuristic function on multidimensional data. Future work can investigate ways to refine the training data and architecture to improve the model's performance and explore multiple data processing enhancement methods.

1 Introduction

Even though chess only comprises of an 8×8 board, its search space is intractable. There have been many artificial intelligence approaches, for example AlphaZero [1] and Stockfish [2]. We intend to use similar techniques to AlphaZero, i.e. convolutional neural networks, but instead of using reinforcement learning, perform supervised learning using Stockfish evaluations as our labels. Stockfish evaluations are calculated through handcrafted algorithms [3], thus they can be used as labels describing the actual relative advantage. By using convolutional neural networks we hope that our model will be able to recognize specific patterns, for example advantageous pawn structure or king safety. Based on evaluations it would then be possible to use algorithms like alpha-beta pruning to create a playable chess AI.

2 Related Works

Stockfish [2] uses alpha-beta pruning on a hard-coded evaluation algorithm on board configurations. Until the release of AlphaZero, it was the best-performing chess AI in existence. Newer versions of Stockfish also incorporate neural networks [4].

AlphaZero [1] uses Monte Carlo tree search, deep convolutional neural networks and reinforcement learning. It is the best performing chess AI to date.

There are several student projects that attempt to use CNNs to create a chess AI, for example [5] and [6]. The project [5] does not use alpha-beta pruning and board evaluations, but instead splits the problem into two separate problems: Which piece to move and where to move it to. The board configurations used in [6] were randomly generated, which leads to unsatisfactory results, whereas [5] learns from games played by humans. In another student project [7], a CNN was trained on games played by humans where positions were evaluated using Stockfish. However, they were limited in their data (13M vs. 300k) and did not normalize the evaluation scores.

Preprint. Under review.

3 Method

3.1 Dataset

Our data is comprised of 13 million chess positions that have been evaluated using Stockfish 11 at a depth of 22 moves, retrieved from the website Kaggle [8]. Chess positions were provided in Forsyth-Edwards Notation (FEN): The whole board is represented as a string, where rows are separated by `/`, and pieces are represented by their first letter, in lower-case for black pieces and upper-case for white pieces, apart from knights, which are represented as `n` and `N`, respectively. Empty squares are represented by an integer denoting the number of consecutive empty squares in a row. For example, the board shown in Fig. 1 is represented in FEN notation as `r1bqkb1r/pppp1ppp/2n2n2/1B2p3/4P3/5N2/PPPP1PPP/RNBQK2R`.

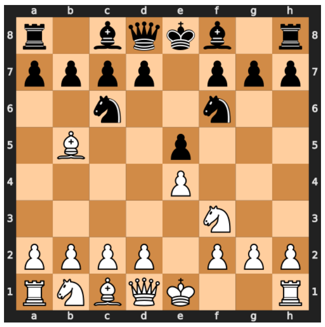

Figure 1: Example chess position, rendered using the `python-chess` library. The FEN representation of this position is `r1bqkb1r/pppp1ppp/2n2n2/1B2p3/4P3/5N2/PPPP1PPP/RNBQK2R`.

Evalutions were given in centipawns. 100 centipawns can be interpeted as having a one-pawn advantage. Evaluations of mate in n where transformed to the centipawn value of 10000. To solve the issue of some evaluations being very high or very low compared to most of the evaluations, we normalized evaluations using the sigmoid function $\sigma(x) = 1/\left(1 + e^{-(x/1000)}\right)$. This essentially transforms centipawn scores to the probability of winning for the white player. The factor of $1/1000$ was chosen since we have observed that most of the evaluations live in the range $[-5000, 5000]$: Using this factor most of the data is on an almost linear part of the sigmoid function, see Figure 2. This factor can be seen as a hyperparameter, as it is a tradeoff between accuracy at lower and higher absolute evaluations.

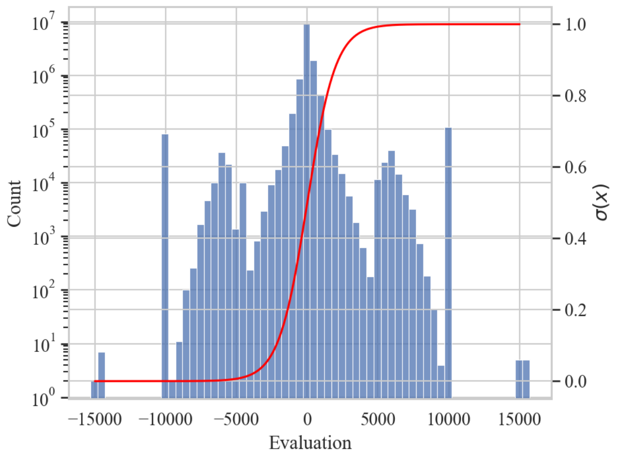

Figure 2: Histogram of the evaluations of whole dataset. The red line shows $\sigma(x) = 1/\left(1 + e^{-(x/1000)}\right)$.

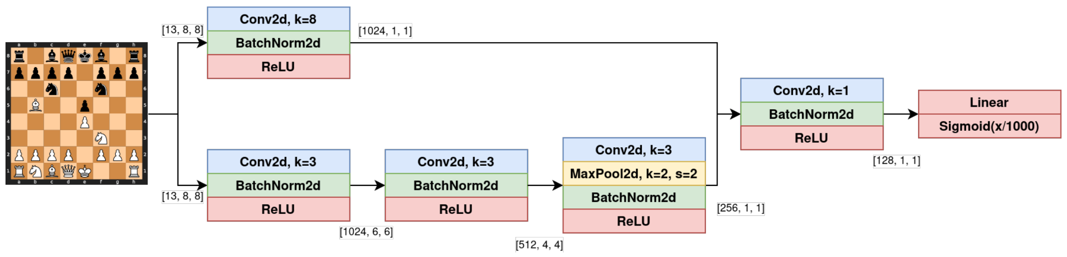

Figure 3: Model architecture.

3.2 Data Processing Pipeline

The input data for the CNN model needs to effectively represent the chess position, taking into account key features such as the position of each piece, the relative values of the pieces, and the scope of the pieces. To accomplish this, a proposed $13 \times 8 \times 8$ dimension input tensor is used, where each 8×8 slice represents the aforementioned features for each piece (6 types of pieces for each side plus empty squares).

To transform the FEN string into the desired input tensor, several steps are taken. First, the FEN string is converted into a board notation represented by an 8×8 matrix with pieces denoted by letters. Then, the board matrix is expanded into 13 board layers using one-hot encodings to represent the locations of each type of piece. This step provides the tensor skeleton for the following transformation.

The next step is to add the values of each piece to the corresponding piece layer. These values are based on convention, with pawns worth 1, knights and bishops worth 3, rooks worth 5, queens worth 9, and kings assigned an estimated value of 2 for their attacking abilities. While this estimation is useful, it should be noted that the king's true value is essentially priceless, and this assumption may need to be revised in future implementations.

Finally, the scope of each piece is included by giving legal moves a value of 1 using the `python-chess` library [9]. All values in the input tensor are treated as positive for white and negative for black. With these data processing procedures in place, the input data for the chess machine learning model is well-structured and captures the key features necessary for effective training. As a last step, data was binarized and stored on disk, since the whole processed dataset occupies about 50 GB, which exceeds the VRAM and RAM available to us.

The implementation of the full pipeline can be found in the Github repository `https://github.com/kcaliban/deep-learning-final-project/`.

3.3 Model

The architecture of our model is shown in Figure 3. Vectorized and processed chess boards pass through two separate pipelines, one containing three convolutional layers, the other a single convolutional layer, before being concatenated and put through another convolutional layer, with finally being flattened and sent through the sigmoid function after being divided by 1000.

Experiments revealed that while increasing the model depth could enhance performance to some extent, the gains were limited compared to increasing the number of filters. It appears that there are no significant abstractions beyond the original chess board required for approximating Stockfish evaluations. This is also in line with how humans improve their intuition by analyzing patterns, both local and global, but not at excessive layers of abstraction.

The implementation of the model can be found in the Github repository `https://github.com/kcaliban/deep-learning-final-project/`.

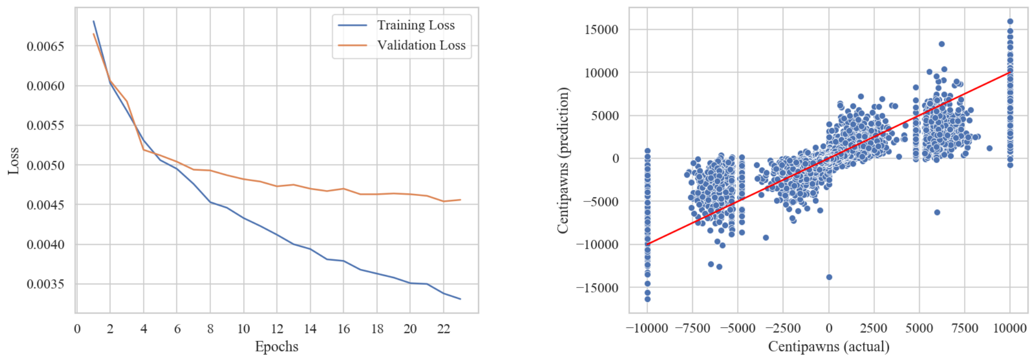

Figure 4: Training results. The left part shows loss curves, the right part shows actual evaluations plotted against predicted evaluations for a sample of $n = 100000$ of the dataset. The red line shows the ideal line $y = x$.

4 Evaluation

Data was split into 80% for training and 20% for validation. We ran 23 epochs with batch size 256 and learning rate 0.01 using the Adam [10] optimizer. MSE was used as the loss function. Training and validation loss are shown in Figure 4. The speed of decrease in validation loss slowed considerably around the 6-th epoch. Final training loss achieved was 0.00331, whilst final validation loss was 0.00456. Predicted and actual evaluations for a sample of $n = 100000$ positions are also shown in Figure 4. Though most evaluations are correct with respect to which player has an advantage, the accuracy of predictions leaves room for improvement. We postulate that better results could be achieved by using a larger dataset, and running the training for more epochs.

The implemenation of the training as well as the final model can be found in the Github repository `https://github.com/kcaliban/deep-learning-final-project/`.

5 Discussion

First of all, it must be said that by learning Stockfish evaluations, our model is learning a heuristic function. These evaluations are not the ground truth, they are a hand-crafted approximation. Our method is therefore limited compared to approaches that are able to discern patterns unknown to humans, for example reinforcement learning, where more data can be processed than humanly possible. Still, we have shown that CNNs are able to approximate a hand-crafted heuristic function on multidimensional data. Furthermore, we have achieved much higher performance than a similar project [7] by using a bigger dataset, more complex model, and normalizing the data.

Moving forward, we can potentially investigate ways to interpret the model and identify specific patterns or features that it is particularly adept or inept at recognizing. This could help us refine the training data or architecture to improve the model's performance on a broader range of positions. There are also multiple ways to enhance the data that could be tried out, such as:

- Incorporate whose turn it is in the position for improved accuracy.
- Handle repetition draws and stalemates, either by assigning a score or removing them from the dataset.
- Reduce the sparsity of the input tensor, such as by combining matrices of the same pieces for both sides.
- Represent the king with a better value that captures its value while maintaining the balance of value distribution in the tensor.

Finally, the trained model could be paired with classical tree-search algorithms such as alpha-beta pruning to create a playable chess AI.

References

[1] David Silver et al. *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*. 2017. DOI: 10.48550/ARXIV.1712.01815. URL: https://arxiv.org/abs/1712.01815.

[2] *Stockfish*. URL: https://stockfishchess.org/.

[3] *Source code of Stockfish evaluation*. URL: https://github.com/official-stockfish/Stockfish/blob/master/src/evaluate.cpp (visited on 04/15/2023).

[4] *Stockfish 12 Release Notes*. URL: https://stockfishchess.org/blog/2020/stockfish-12/ (visited on 04/13/2023).

[5] *ConvChess*. URL: http://cs231n.stanford.edu/reports/2015/pdfs/ConvChess.pdf.

[6] *Creating a Chess AI Using CNNs and Stockfish*. URL: https://www.nbi.dk/~petersen/Teaching/ML2022/Week8/FinalProject_16_CooperAsgerAnirudhAtlasAugustPetroula_ChessAI.pdf.

[7] *Training a Convolutional Neural Network to Evaluate Chess Positions*. URL: http://www.diva-portal.se/smash/get/diva2:1366229/FULLTEXT01.pdf.

[8] Ronak Badhe. *Chess Evaluations*. URL: https://www.kaggle.com/datasets/ronakbadhe/chess-evaluations (visited on 04/08/2023).

[9] *python-chess: a chess library for Python*. URL: https://python-chess.readthedocs.io/en/latest/ (visited on 04/13/2023).

[10] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).