CNN-Based Chess Engine

Kevin Thomas

Abstract

This project applies deep learning to evaluate chess board positions using a convolutional neural network. The model takes FEN (Forsyth–Edwards Notation) strings as input and is trained to predict evaluations of positions using supervised learning. This report outlines the full pipeline, including data preprocessing, model architecture, training procedure, and evaluation. Basic gameplay against the trained model is supported and and the board states can be visualized using the python-chess library.

1 Introduction

Chess is a classic strategic game that has been a standard problem in artificial intelligence. Traditional engines used to use brute-force search and handcrafted evaluation functions. Deep learning has revolutionized chess via AI, allowing chess engines to learn to play chess well from scratch using reinforcement learning and self-play.

This project trains a CNN to evaluate chess positions using supervised learning on labeled game data. Chess is a 2D game where local patterns—such as pawn structures, attacks, and piece positioning—are critical to strategic evaluation. CNNs are well-suited to this domain.

2 Dataset and Preprocessing

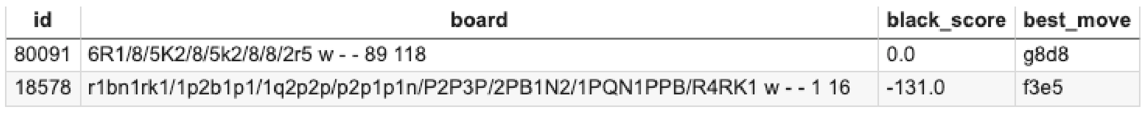
The dataset consists of chess games encoded in FEN strings. This data comes from [2]. Each FEN string is parsed and converted into a tensor representation suitable for neural network input. Preprocessing steps include:

- Encoding piece types and positions: Each FEN string is parsed to identify the pieces on the board and their colors. This information is mapped into a structured format suitable for input to the neural network.

- Normalizing board state: the parsed board representation is standardized to ensure consistent input formatting, regardless of how the original FEN string varies.

2.1 FEN Notation

Forsyth–Edwards Notation (FEN - [1]) is a standard for describing the state of a chess game. It includes Piece placement, Active player, Castling rights, En passant target square, Halfmove clock and Fullmove number. An example of the notation is shown in figure 1.



id	board	black_score	best_move
80091	6R1/8/5K2/8/5k2/8/8/2r5 w - - 89 118	0.0	g8d8
18578	r1bn1rk1/1p2b1p1/1q2p2p/p2p1p1n/P2P3P/2PB1N2/1PQN1PPB/R4RK1 w - - 1 16	-131.0	f3e5

Figure 1: train.csv data

2.2 Library *python-chess*

The ‘python-chess‘ library is used in the code for:
- Parsing FEN strings: It converts FEN strings into board objects, making it easier to extract piece positions and game state.
- Legal move generation: The library provides functions to list all legal moves from a position, which supports evaluation or interaction logic.
- Game state handling: It manages information like turn, castling rights, en passant squares, and more, simplifying input processing.
- Visualization: Parses FEN strings to generate a board object and renders it as an SVG image.

3 Model Architecture

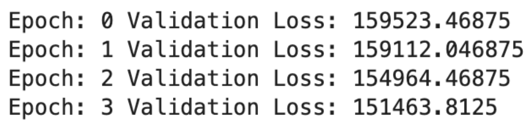
The neural network is implemented using PyTorch. The input layer processes an encoded 8x8 board. The architecture includes:
- Convolutional layers for spatial pattern recognition: The model starts with layers that scan the board to detect useful patterns, like piece groupings and threats.
- Fully connected layers for decision making: After detecting patterns, the model uses regular layers to combine this information and form a final judgment.

- Loss function for training: Mean-squared error loss is used to measure the error between the predicted and actual position evaluations.
- Optimizer used for learning: The training uses Adam optimizer. It adjusts the model step-by-step.
- Output layer: The final layer gives one number that represents how good or bad a chess position is, based on the model's judgment.

4 Training and Evaluation

The model is trained to predict a numeric evaluation (black score) from a FEN board state. This is a regression problem, and the accuracy metric is the Mean Squared Error (MSE) loss between the predicted score and the actual score in the dataset.

The first 55,000 entries in the dataset are used for training. The remaining entries are used for validation to evaluate model performance during training. The model is trained over several epochs with a defined learning rate and batch size. The model tracks and prints the validation loss at the end of each epoch to monitor how training is progressing. An example of the logging during training is shown in figure 2.



```
Epoch: 0 Validation Loss: 159523.46875
Epoch: 1 Validation Loss: 159112.046875
Epoch: 2 Validation Loss: 154964.46875
Epoch: 3 Validation Loss: 151463.8125
```

Figure 2: Example of logging during training

The network was trained for around 100 epochs. Initially the validation loss was going down with the training loss. Around 20 epochs, the validation loss plateaued while the training loss still showed a downward trend as can be seen in figure 3.

5 Game Play

After the network is trained, the following sequence of actions help us play a game

- The user first enters the move to be made.
- All legal moves are generated from this position.
- Each move is simulated to see the new board.
- A score is predicted for each resulting board using the trained model.

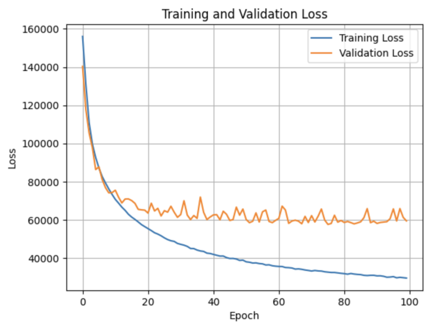


Figure 3: Training and Validation Loss

- The move that gives the best evaluation is chosen and a move is made. The computer in this case is always assumed to play black
- The user can make the next move at this point. The game continues till there is a winner or a draw.

An example of the visualization of the game move is shown in figure 4.

6 Future Work

Here are some more things that can be tried for this application

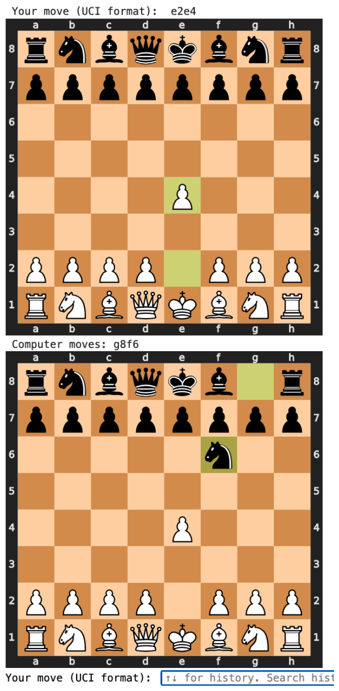
- Hyperparameter tuning: Experimenting with different learning rates, batch sizes, layer sizes, and activation functions could help improve model performance.
- Data augmentation: Adding transformations such as board flipping or rotation (with label adjustment) could increase dataset diversity and generalization.

7 Codebase

The code for this project is available at https://github.com/kvnjthms/chess-nn.

8 Conclusion

This project demonstrates the use of convolutional neural networks to evaluate chess positions from FEN strings using supervised learning. The model achieves reasonable accuracy for static evaluation tasks and supports basic gameplay interaction.


Your move (UCI format): e2e4

Computer moves: g8f6

Your move (UCI format):

Figure 4: Game Visualization

Visualization and legal move generation are handled through the ‘python-chess‘ library. While the current model is limited to evaluation, it provides a foundation for extending to move prediction or reinforcement learning. Future improvements could include optimizer changes, hyperparameter tuning, and richer training data.

References

[1] Steven J. Edwards. Fen – forsyth-edwards notation. `https://www.chessprogramming.org/Forsyth-Edwards_Notation`, 1994. Accessed: 2025-03-23.

[2] Kaggle. Train an ai to play chess. `https://www.kaggle.com/competitions/train-an-ai-to-play-chess`, 2022. Accessed: 2025-03-26.