



# BoardMind: A Deep learning Chess Model

Matthew Borkowski and Philip Virdo

201588010

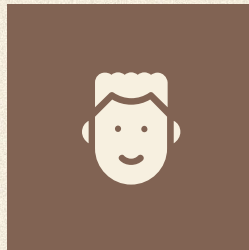
201749430



# OUR TEAM

## Matthew Borkowski

- Model optimization
- Decision Tree Enhancing
- Data collection and formatting



## Philip Virdo

- Model development and foundations
- Model optimization
- Model training management





# Introduction to Project

- Chess is a benchmark for AI research, pushing the limits of computational power and machine learning algorithms.
- Our research project aims to develop a chess engine that can play at a high level and make human-like decisions.
- We developed our own engine from scratch in Python, using a convolutional neural network, and move analysis helper methods such as Monte Carlo Tree Search









# Abstract





- The aim of the project is to design and develop a chess engine called BoardMind, that can play chess effectively by leveraging machine learning techniques.
  - We will gather historical chess games, preprocess the data to extract relevant features, and explore deep learning architectures such as convolutional neural networks, alongside decision tree methods such as the Monte Carlo Tree Search method
  - BoardMind's performance will be evaluated against existing chess engines and human players, using metrics like the ELO rating system, win-loss ratios, and game quality
- 
- 





# Objectives

- Develop a chess AI in Python that uses deep learning techniques
  - Optimize the performance of the AI using various decision models such as Monte Carlo Tree Search or Alpha Beta Pruning
  - Create a framework to allow the model to continually improve by expanding the data sets when required
  - Gain a foundational understanding convolutional neural networks and how they work in a practical setting such as chess
- 
- 



# Pre-Processing and Data Analysis

- The dataset for this project was collected from chess databases such as the lichess.org open database and the pgnmentor database
  - Both sites had data that could be downloaded as or easily formatted into the pgn file format, allowing for easy training for the chess AI
- we parsed through the data using some helper functions to organize the games into smaller sub-files based on the opening theory used in the games
  - This allowed us to train the model with a batch of games that use the same opening theory but have varying middle and late game moves

```
[Event "TCh-BEL 2022-23"]  
[Site "Belgium BEL"]  
[Date "2022.10.16"]  
[Round "2.5"]  
[White "Weiermann,A"]  
[Black "Maltezeanu,Stefan"]  
[Result "1-0"]  
[WhiteElo "2237"]  
[BlackElo "2139"]  
[ECO "A03"]
```

```
1.f4 d5 2.Nf3 Nf6 3.g3 g6 4.Bg2 Bg7 5.O-O O-O 6.d3 b6 7.e4 dxe4 8.dxe4 Bb7  
9.Qxd8 Rxd8 10.e5 Ne8 11.Nc3 Na6 12.Be3 Nb4 13.Rac1 c5 14.a3 Nd5 15.Nxd5 Bxd5  
16.Rfd1 Nc7 17.c4 Bc6 18.b3 f6 19.exf6 Bxf6 20.a4 Na6 21.Ne1 Bxg2 22.Kxg2 Nb4  
23.Kf3 Bb2 24.Rb1 Bc3 25.Ke2 Bxe1 26.Kxe1 Kf7 27.Ke2 Ke6 28.Bd2 Nc6 29.Bc3 g5  
30.fxg5 Kf5 31.Rf1+ Kxg5 32.Rf7 e5 33.Rbf1 Nd4+ 34.Bxd4 exd4 35.R1f5+ Kg6  
36.R5f6+ Kg5 37.Kf3 Rh8 38.h4+ Kh5 39.Rf5+ Kh6 40.R5f6+ Kh5 41.Rg7 Rag8 42.g4+ Kxh4  
43.Rh6+ 1-0
```





# Platform and Configurations



- PyTorch to set up and train the convolutional neural network and the Python Chess library to give context to the pieces, actions and rules of chess to the neural network
  - PyTorch is one of the default libraries for neural networks because it contains a dynamic computational graph feature
- the AI was trained on a desktop running Windows 11 with a AMD Ryzen 5 5600 6-core processor
  - On this hardware, the average training time with approximately 6 epochs was around 5 hour









# Data Split

- The training set for the AI were the pgn's from open chess databases
    - This allowed for the AI to build a general understanding of movements during various stages of a game of chess(early, middle and end game)
  - To validate the model, the AI played itself on both sides of the board
    - Using existing chess analysis programs, we analyzed the moves that the AI made for each piece and looked for inaccuracies
  - The model was tested by playing against human players to see how it matches up against the natural move selection and to see how accurate its responses were
- 
- 







# Model Planning

- The initial algorithms we looked at were:
    - Minimax Algorithms
    - Alpha-Beta Pruning
    - Monte Carlo Tree Search (MCTS)
  - Through the trial and error of each algorithm, we ended up choosing the MCTS because it can be engineered to be more efficient than Minimax and Alpha-Beta, especially when using a CNN
    - MCTS is particularly good because it mitigates the Horizon Effect
- 
- 





# Model Training

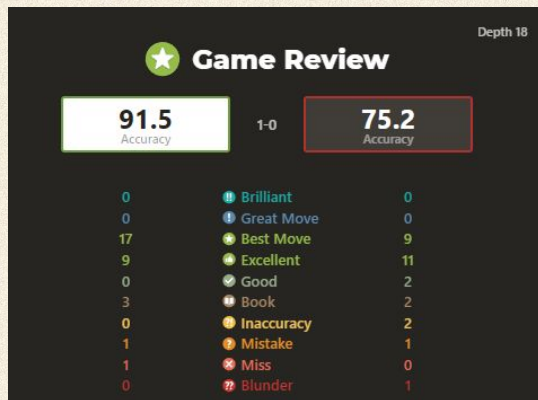
- Due to the large amounts of data used to train the model we used a combination of chunk loading and batch sizing to manage the data more efficiently and train the model more accurately
  - This trains the model on a batch of games for a given number of epochs, saving the weights from the epochs and then moving onto the next batch
- We also utilize random shuffling when creating batches to prevent the model from overfitting
  - This increases its adaptability to respond accurately to opponent's moves





# Model Evaluation and Optimization

- Using the chess analysis programs, we were able to determine which moves the AI made that resulted in weak play and tune our hyperparameters accordingly
  - In the early stages, our model was opening the game by pushing the 'a' or 'h' pawns which is an incredibly weak opening move







# Performance Metrics

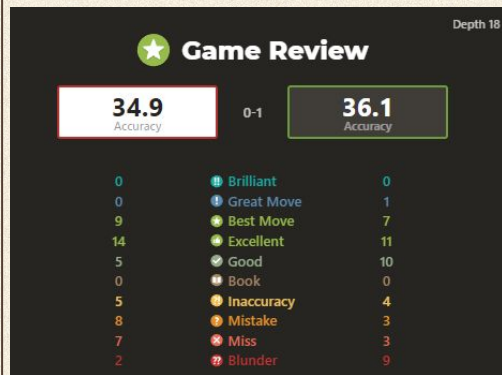


- We utilize mean squared error (MSE) as a performance metric to measure the difference between the predicted outputs and the actual target values during the training process
- This calculation computes the average squared difference between the predicted and target values, resulting in a scalar value that represents the model's performance for the current batch
- The training process goes on for the specified number of epochs, with the goal of minimizing the MSE loss to improve the model's performance over time
  - The running loss is updated and displayed in the progress bar, allowing for the monitoring of the model's training progress



# Model Progress



- Improvements to the AI (black) by tuning hyperparameters and training







# Interpretation of Results

- In this early stage of training we can see that BoardMind has an extremely low range of accuracy and it is evident that the model requires more time training
  - In this second round of testing we can see that the model is making significant improvements playing at a range of accuracy somewhere between 58% and 75% which is a great improvement
  - After concluding these results we can see that BoardMind now plays at a confident range within 70% - 80%
- 
- 





A decorative border surrounds the central text. It consists of a thin dark line forming a rectangle. At each of the four corners, there is a small dark diamond. Additionally, at each corner, there is a large chess piece: a green king in the top-left, a brown king in the top-right, a brown king in the bottom-left, and a green king in the bottom-right.

# Project Demo





# Conclusion

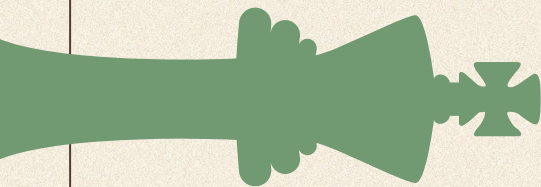
- By employing convolutional neural networks, the AI was able to learn from experience and adapt to various game scenarios, a feat that conventional chess engines with their complex algorithms do not necessarily utilize
  - By combining cutting-edge deep learning techniques, meticulous data pre-processing, and thoughtful model optimization, BoardMind has evolved into a powerful chess AI capable of challenging human players
- 
- 





♦ ♦ ♦

# Thank you



# CP322

♦ ♦ ♦

