Chess Blunder Prediction: Final Report

Introduction/Background

Literature Review

- Acher and Esnault (2016) [1] conducted large-scale chess analysis using Stockfish, highlighting the importance of chess engines in understanding human decision-making. Their work identified that computation time of chess engines is a crucial bottleneck in wide-scale interaction between humans and chess engines.
- Silver et al. (2017) [2] introduced AlphaZero, an AI that achieved superhuman performance in chess through self-play, combining deep reinforcement learning with Monte Carlo Tree Search. Their work showed deep learning models can be integrated into chess analysis and beat other state-of-the-art methods.
- McIlroy-Young et al. (2020) [3] developed Maia, an AI model fine-tuned to mimic human chess play, including mistakes, for more relatable and educational purposes. Theirs was one of the first work to use machine learning methods to understand human chess play.

Dataset Description:

A standard Lichess dataset contains chess game information such as player IDs, ratings, game results (win, loss, or draw), time control, and termination reasons (e.g., resignation or checkmate). Each game is identified by a unique ID and includes metadata like the opening name and ECO code, date and time, and the move list in Portable Game Notation (PGN). Some datasets may also include movetimestamps, annotations for notable moves, or "optimal" engine evaluations at certain moves. Optional player metadata, such as titles (GM, IM) may also be available.

Dataset Link: Link to Dataset

Problem Definition

Problem Statement:

Since chess engines achieved superhuman performance, they have been extensively used as analytical tools in human chess play. Current chess tools excel at suggesting the best moves but lack the capability to warn players about the likelihood of making blunders in specific positions. There is a need for a such a specialized chess tool that can assess and communicate the "blunder risk" of any given position, taking into account position complexity and common mistake patterns at different skill levels. This tool would create crucial awareness for players before making critical decisions, helping them approach risky positions with appropriate caution.

More formally, we want to develop a model, say M, that can accept a chess position $p \in P$ and predict a label $M(p) \in \{0, 1\}$ where 0 indicates that a human player is unlikely to make a blunder in position p and 1 indicates that a human player is likely to make a blunder in position p. The model M will be trained on a dataset of chess positions with labels indicating whether a blunder was made in that position or not.

Motivation:

The recent chess boom, accelerated by the pandemic, has highlighted a critical gap in chess tools for casual players. While traditional chess engines focus on finding perfect moves, they don't help players understand when they're in dangerous positions likely to cause blunders. This is because making a blunder is an inherently "human" trait and chess engines are typically tuned to achieve perfect (superhuman) play. Further, Maia [3] shows that artificially restricting superhuman chess engines (like capping the depth in tree-search algorithms) takes the models even farther from human behavior. A blunder prediction system would serve as a crucial learning tool, especially for newer players who often make game-losing mistakes without recognizing the risks. This addresses the growing need for practical learning tools that help prevent catastrophic errors rather than just suggesting optimal moves.

Methods

Data Preprocessing: We have access to a lot of chess games, but it needs to be systematically converted and labelled into chess positions and blunder labels. The following steps exhaustively detail the data preprocessing pipeline:

- **Download dataset from Lichess and extract PGN:** The system downloads large datasets of chess games from Lichess's public database. These datasets contain standard rated chess games played during specific months and are initially compressed to save storage space. The

data comes in a special compressed format (.zst) which needs to be unzipped before it can be used. The downloading and unzipping processes happen simultaneously for multiple files to save time, with visual indicators showing the progress of both operations. Once uncompressed, the data becomes available in PGN format, which is the standard format for recording chess games, containing all the moves and relevant game information.

- **Stockfish Engine Integration:** To determine what constitutes a blunder, the system leverages Stockfish, one of the strongest chess engines available. For each position, Stockfish evaluates both the current position and the position after a move is made. The difference in these evaluations, measured in centipawns (1/100th of a pawn's value), reveals how much the move weakened the position. If this difference exceeds a certain threshold, the move is classified as a blunder, providing our ground truth for training the model.
- **Position Extraction:** Convert PGN data to 8x8x17 grid representations using `python-chess`. 17 channels are represent the following: Pieces (12): One channel per piece type/color (6 white, 6 black); Castling (4): Legal castling options for both players; Turn (1): Active player indicator (white/black)
- **Feature Extraction:** Classical ML algorithms struggle to obtain good representations from raw data. Hence, we manually extract useful features from board states to pass it to ML algorithms. The full list of features includes:
  - Number of legal moves for current player
  - Time remaining for current player and time delta against opponent
  - Elo of current player and elo difference against opponent
  - Mismatch in pieces (knights against bishop)
  - Number of pieces on the board
  - Opening type
  - Whether the queens are on board
  - Number of pieces attacking the king
  - Current move number
  - Time control
  - Stockfish evaluation
- **Data Filtering:** We process games where both players are rated 1000-2000 Elo but skip bullet games and moves with insufficient clock time. For each position we evaluate it with Stockfish before and after move and mark it as blunder if eval difference > 3 centipawns. To ensure roughly equal blunders and non-blunders, we sample blunders with 1/4 probability and non-blunders with 1/32 probability. Thus the output is board state (8x8x17), extracted features, and

blunder label (0/1), and player Elo (a single number measuring the strength of a player). This creates a balanced dataset of ~2.8M blunders and ~4.2M non-blunders.

Machine Learning Models

- **PCA**

  As noted above, the input dimension of raw board data is 8x8x17 = 1088, which is quite high. Therefore, we use PCA to reduce the dimensionality of the data. We retain 95% of the variance in the data, which experimentally results in a reduced dimensionality of 185. We implement PCA using `sklearn` library.

- **Classical ML**

  We experiment with various classical machine learning algorithms, namely Random Forest and Logistic Regression. We tried SVM but given the size of the dataset, it was computationally expensive. We train our models on the raw board state, extracted features, and PCA-transformed data. We perform hyperparameter tuning on all models and obtain hyperparameters that give best results on the test set. `sklearn` library is used for implementation.

- **Deep Learning**

  We experiment with two deep learning models, Fully Connected Network (FCN) and Residual Convolutional Neural Network (R-CNN) emulating Maia [3]. FCN has 2 hidden layers with 128 and 64 units respectively. R-CNN has 64 convolutional layers with residual connections and 1 fully connected layer at the end. We train these models only on raw board state data because deep learning models typically work well when given raw data. We use `PyTorch` for implementation.

Results and Discussion

Visualizations

We will visualise a few chess positions from our dataset and show our best model's prediction for those. We will also show feature importance for classical ML models trained on extracted features. This will help us understand which features are most important in predicting blunders. Finally, we also include ROC curve for our best classical ML model.

Quantitative Metrics

After fitting each model, we evaluate them for accuracy on a held out test set. We run two experiments - one where we train the model on the entire dataset (1000-2000 Elo) and another where we train the model on a subset of the dataset (1600-1700 Elo). The purpose of this experiment is to understand whether there is a significant distribution shift of blunders as Elo changes. We also compare our results with Maia [3] to understand how well our model performs.

Model Comparisons

Model	1000-2000 Elo	1600-1700 Elo
Logistic Regression with Raw Board State	66.3%	66.8%
Random Forest with Raw Board State	61.9%	63.9%
Logistic Regression with PCA	62.7%	64.1%
Random Forest with PCA	61.8%	64.0%
Logistic Regression with Extracted Features	63.5%	64.2%
Random Forest with Extracted Features	68.3%	68.0%
FCN	67.6%	67.3%
R-CNN	66.5%	65.8%
Maia (Random Forest)	56.4%	---
Maia (R-CNN)	67.7%	---

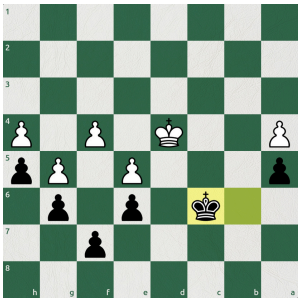
Accuracy of models when trained on data from two ELO range: 1000-2000 and 1600-1700

Confusion Matrix		Predicted	
		0	1
Actual	0	TN = 4936	FP = 583
	1	FN = 2289	TP = 1255

Confusion Matrix on test data across all ELO for Random Forest with Feature Extraction (Best Model)

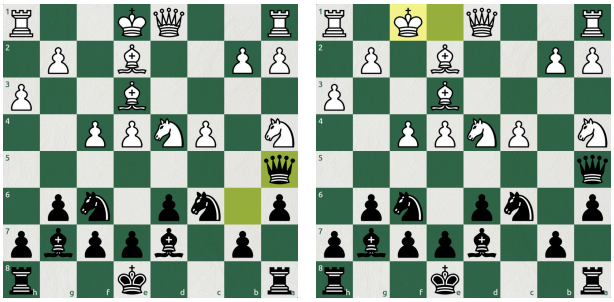
Analysis

Based on the quantitative results in the table, all our models achieve good performance compared to Maia baselines. Specifically, random forest with extracted features performs the best on the entire dataset. This is expected because extracted features are designed to capture human-like behavior. However, the performance of deep learning models is also quite good, indicating that raw board state data is sufficient for predicting blunders. Further, the model performs slightly better when trained on the smaller dataset. This is to be expected as the variations in blunders are less pronounced in the smaller dataset. Players of similar strength tend to perform the same kind of blunders, which makes it easier for the model to learn the patterns.



Before Move **After Move**

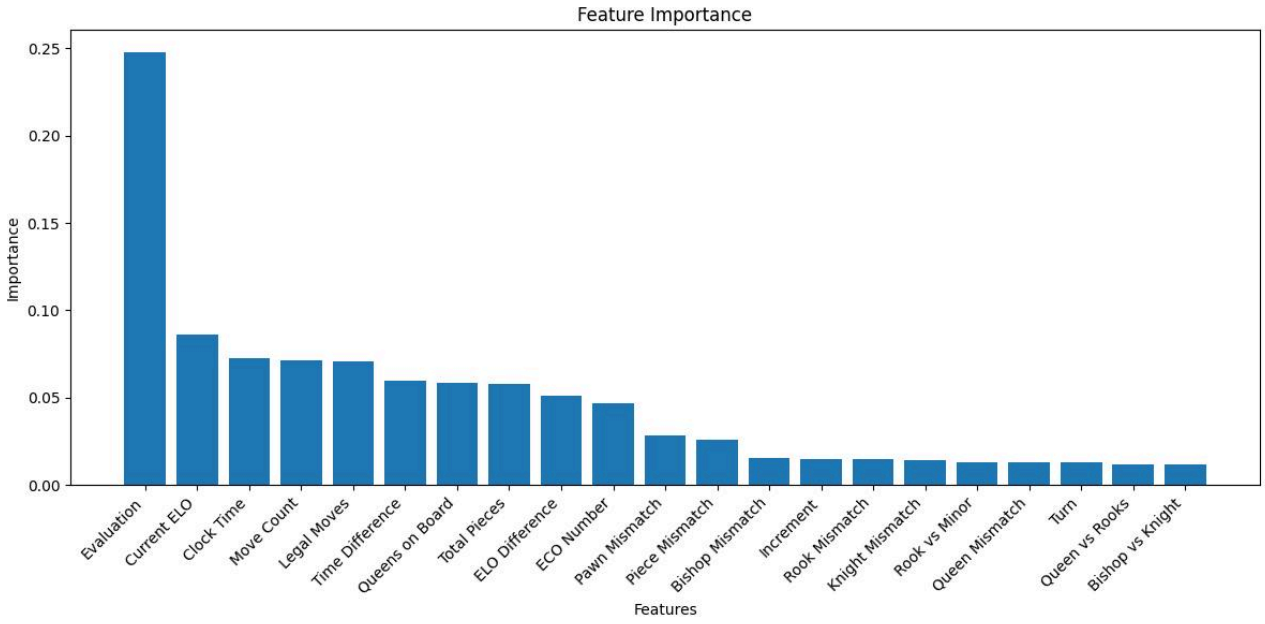
Non-Blunder Position



Before Move **After Move**

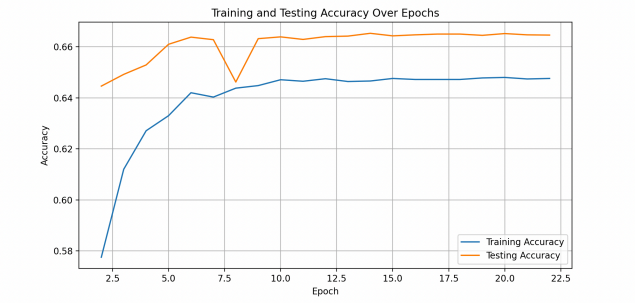
Blunder Position

The visualizations above show the board positions before and after a move for a non-blunder and a blunder position. The non-blunder position is during a drawn-out endgame where the best moves are quite obvious (shuffling the king side-by-side). Here a human is quite unlikely to make a blunder. The blunder position is from a middle game where the best move (Kc3) is not so obvious. Weaker Elo humans are prone to make blunders here. Our best model is able to label each position correctly.

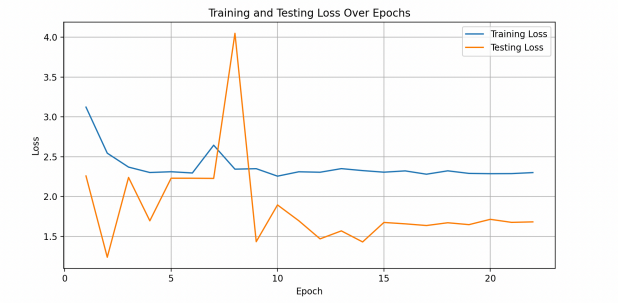


Feature Importance for Random Forest with Extracted Features (Best Model)

The above plot shows the importance of extracted features in the trained Random Forest model. Empirically, the model puts a lot of weightage to the stockfish evaluation of a position to determine blunder probability.



Accuracy trends during R-CNN training



Loss trends during R-CNN training

Training trends for R-CNN model

The training trends for the R-CNN model show that the model converges after 10 epochs. The accuracy increases and the loss decreases with each epoch. This indicates that the model is learning the patterns in the data and is able to predict blunders effectively.

Next Steps

In this project, we created a model that predicts whether a human player is likely to make a blunder in a given chess position. Our models achieved good accuracy when compared to baselines, but

there is still room for improvement. Some potential next steps include:

- Collecting more data: Our dataset was limited to players with Elo ratings between 1000 and 2000. Even within this range, we were only able to process a subset due to computational limitations. Future work can work on large amounts on data and train larger models.
- Human evaluation: Our work essentially tries to mimic human behavior. Therefore, it is natural to conduct surveys on diverse populations to gauge our model's performance.

References

1. Acher, Mathieu, and Gilles Esnault. "Large-scale Analysis of Chess Games with Chess Engines: A Preliminary Report." arXiv preprint arXiv:1607.04186 (2016).
2. Silver, David, et al. "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm." arXiv preprint arXiv:1712.01815 (2017).
3. McIlroy-Young, Reid, et al. "Aligning Superhuman AI with Human Behavior: Chess as a Model System." arXiv preprint arXiv:2006.01855v3 (2020).
4. Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." nature 529.7587 (2016): 484-489.
5. Pgn-extract [https://www.cs.kent.ac.uk/people/staff/djb/pgn-extract/], which can be processed in parallel.

Gantt Chart

Gantt Chart Link

Contribution Table

Name	Proposal Contributions
Shreyas Kanjalkar	Data Preprocessing, Meetings, Code Review
Harshil Vagadia	Result Analysis, Report, Code Review, Meetings
Miloni Mittal	PCA, Report, Meetings
Jineet Hemalkumar Desai	Classical ML Algorithms, Meetings
Lohith Kariyapla Siddalingappa	Deep Learning Algorithms, Meetings

Video Presentation

Watch the proposal video here: Proposal Video

GitHub Repository

Repository: GitHub Repo Link