UNIVERSITY OF ICELAND

Who Is Playing?

A Study of Chess Behavior

Introduction to Deep Neural Networks ∼ TÖL506M

Arnar Ágúst Kristjánsson (aak18@hi.is)
Jason Andri Gíslason (jag28@hi.is)
Kári Rögnvaldsson (kar26@hi.is)

Instructor: Steinn Guðmundsson
Thursday 24th November, 2022

SCHOOL OF ENGINEERING AND NATURAL SCIENCES

Contents

Abstract

In recent years, machine learning models have been able to match or surpass human abilities in some specific tasks, such as image recognition. In chess, we can recognize that different players have different playing styles. This gives rise to the question: "Can we predict who is playing a chess game from their moves alone?".

In this project, we develop a deep neural network that tackles this task. We use a convolutional-LSTM model that learns patterns in expert chess player decisions with the goal of predicting who the player is. We use the few-shot classification framework to generalize our model to new players with few reference examples. Our model achieves 13.4% results when distinguishing between the 400 players it was trained on. This is 53 times better than guessing randomly. The model was also 51 times more accurate than a random guesser on previously unseen players when using only 50 games from each of them as a reference.

1 Introduction

The game of chess has been among the most popular intellectual games for hundreds of years. More recently, it has been the subject of many pioneering experiments related to artificial intelligence. In the beginning, research was heavily focused on making systems that play chess well. Recently, more research has focused on how humans play chess. In this project, we take that to an extreme. We want to develop a model that can accurately determine who is playing a given chess game, given only the moves that were played. We are repeating a previous experiment exploring this, but we have altered some methods. We explore the topic of distinguishing between expert chess players, while the previous experiment focuses mainly on amateur players. We train a convolutional-LSTM model that generates embeddings for both players playing a chess game. We use prototypical learning to generalize the model to new players whose data was not used in the training of the model.

1.1 Previous Work

Our approach is based on previous work by (McIlroy-Young et al., 2022), where a transformer model was created from the Lichess dataset to determine the playing style of the player that is playing. They have adapted methods from speaker verification in the form of the Generalized end-to-end contrastive loss function introduced in (Wan et al., 2017). Their move-level feature extraction is also based on previous developments in human-like chess-playing systems. Maia is an artificial intelligence system that is aimed towards playing chess like a human (McIlroy-Young et al., 2020). However, due to ethical concerns, they do not disclose their code. Therefore, we wanted to validate their results.

1.2 Ethical Concerns

The problem at hand is a subproblem of the more general task of identifying a decision maker from their decisions alone. Solutions to these problems reduce the possibilities of remaining anonymous online, which can be a large concern in some instances. In this particular case, a good solution would prevent players from being able to play chess online anonymously. We think this is not a substantial ethical concern, though we understand that this might trouble some people.

2 Methods

In this project, two different types of models were implemented and looked at. The baseline model pools the moves in each game together and predicts what the player was playing simply from the moves played in the game (without using information about the game position when the move was played or the order in which the moves were played). The main model of this study is a convolutional-LSTM model that captures the time element in the game, trying to learn patterns in the play to classify who is playing each game.

2.1 Data Processing

The raw data (see section 2.5) is in Portable Game Notation (.pgn) format, which is good for storing game metadata and the moves each player made during the game. When preprocessing the data, we stored the first 20 moves of each game, excluding shorter games, and stored it in a `numpy` array. For each game, we put together another array that included an ID of both players playing the game.

The state of a chessboard can be represented in multiple ways. The way that was chosen for this project was to represent each move as a $24 \times 8 \times 8$ tensor of binary values. This is a similar representation of the moves as used in previous work (McIlroy-Young et al., 2022), but we omit some of the excess information not related to the chess board. Rhe first 12 channels represent what the board looked like before the move was made, and the later 12 channels represent the board after the move was made. Each board state is represented with 6 channels for the white pieces, one for the pawns, one for the rooks, one for the knights, one for the bishops, one for the queen, and one for the king. Similarly, the black pieces are represented with 6 channels. Figure 1 shows an example of a board representation on the tensor form.

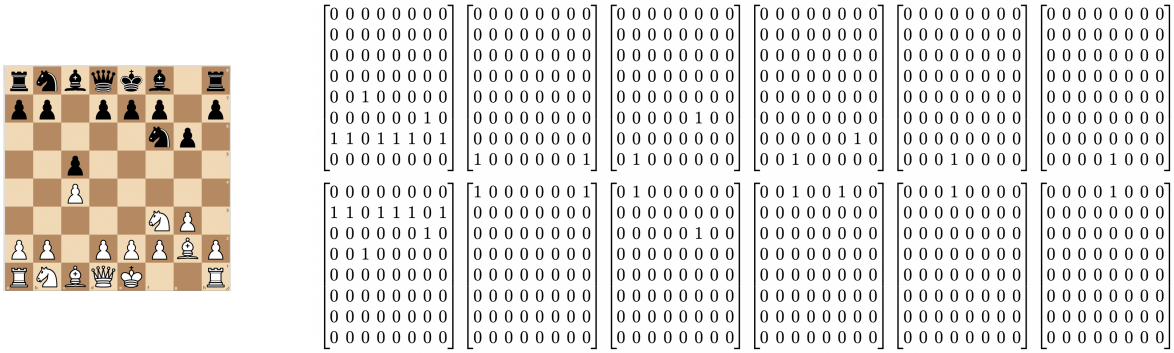


Figure 1: An example of a chess board representation. The channels of the tensor represent the board state of the image on top. The first 6 channels are for the white pieces, and channels 7-12 are for the black pieces. In the board representation that is put into the model, the state of the board before and after the move is used.

To create the tensor representation of the chess board we slightly modified code from Trevor Gaffa, in his lczero tools package.

In this way, each game is represented by a $n \times 24 \times 8 \times 8$ tensor, where n represents the number of moves made in the game. For this project, $n = 20$ was chosen. We split each played game into games made with white pieces and games made with black pieces.

2.2 Convolutional LSTM Model

2.2.1 Model Structure

The game-level data in each chess game is time-dependent, and the moves are played sequentially. Therefore, one thinks that the sequential nature of the data will help in predicting what player is playing. Our model is inspired by the work of McIlroy-Young et al. (2022), slightly simplified, using an LSTM model instead of a transformer to capture the sequential nature of the data. The embedding sizes and most hyperparameters are taken straight from their model. The model is composed of three main components. The first component is a CNN that each move is passed through, which changes the $24 \times 8 \times 8$ tensor into a 64×1 vector, followed by a fully connected layer to transform the result into a 320×1 vector. The second component is an LSTM that takes in the representations created by the CNN model and outputs a 1024×1 vector that is finally passed through to a fully connected layer that results in a 512×1 representation of each game. The model learns these embeddings of chess games using a contrastive loss function called Generalized end-to-end loss (GE2E, see Section 2.2.3). The idea of the G2E2 loss function comes from speaker recognition and rewards the model for keeping samples from the same player close together in the embedding space and samples from different players distant from each other.

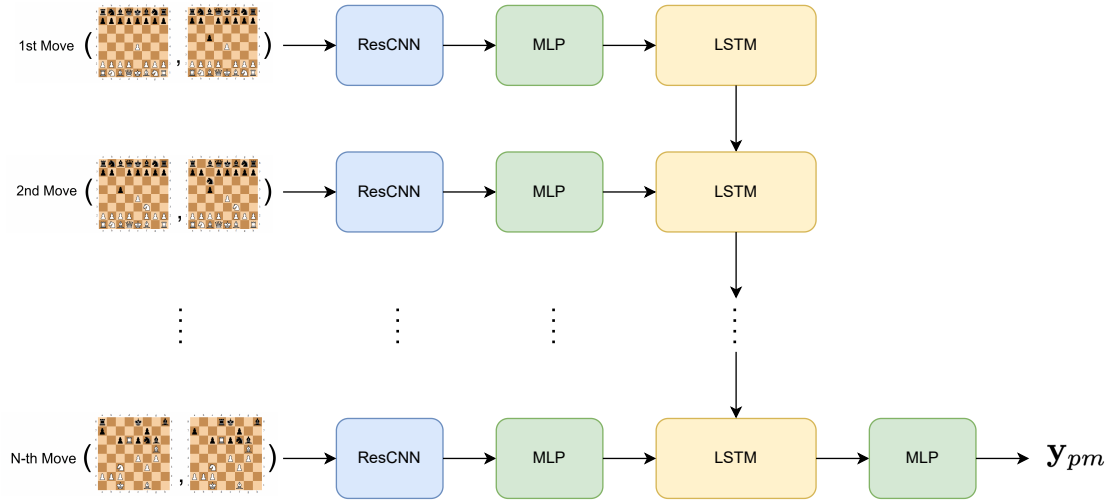


Figure 2: Visual representation of the convolutional LSTM model

2.2.2 Residual Network Architecture

We used a residual convolutional network to get a representation of a single move. The architecture was mainly inspired by the architecture used for this purpose in Maia, which is a chess engine designed to play like a human (McIlroy-Young et al., 2020). In our main model, we used one convolutional layer to convert from 24 channels to 64. Then there followed 6 residual blocks that conserve the dimension of the representation. The residual blocks consist of two convolutional layers with batch normalization after each one, and ReLU activation. We used kernels of size 3×3, a stride and zero padding of size one for all of the layers. Finally,

we used global average pooling to convert this to the shape of $64 \times 1 \times 1$. This was then fed through a fully connected layer whose output was the input to the LSTM. Figure 3 provides a visualization of this.

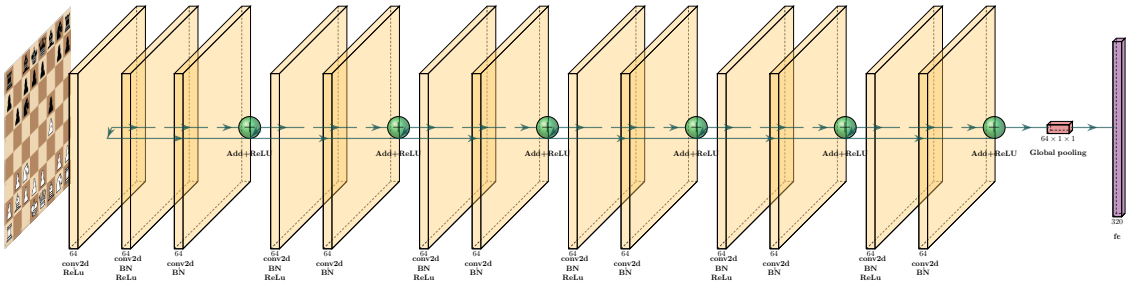


Figure 3: Visualization of the residual network used for processing the moves

2.2.3 Generalized End-to-End Loss

Since the model is an embedding model, a classic softmax/MSE optimization does not suffice. A contrastive loss function must be implemented (Wan et al., 2017). In each training batch, M games are sampled from each of N random players from the pool of players. Following the notation from McIlroy-Young et al. (2022), we denote the embedding of game m of player p with $\mathbf{y}_{pm}$. When the loss function is calculated, the centroids $\mathbf{c}_p = \frac{1}{M}\sum_{m=1}^{M} \mathbf{y}_{pm}$ are calculated. Further, the modified centroids

$$\mathbf{c}_p^{(-i)} = \frac{1}{M-1} \sum_{\substack{m=1 \\ m \neq i}}^{M} \mathbf{y}_{pm}$$

are calculated. These modified centroids are used to compare the distance from a single game to the other games that the player played. If we included the game we are looking at in that centroid, that game would pull the centroid closer to itself. Figure 4 visualizes the concepts of $\mathbf{c}_p$, $\mathbf{c}_p^{(-m)}$ and $\mathbf{y}_{pm}$.

We define a similarity matrix $\mathbf{S}_{pm,q}$, where p represents the player that played the game and q means that we are comparing this game to the games played by player q, as

$$\mathbf{S}_{pm,q} = \begin{cases} w \cos\left(\mathbf{y}_{pm}, \mathbf{c}_p^{(-m)}\right) + b, & \text{if } p = q \\ w \cos(\mathbf{y}_{pm}, \mathbf{c}_q) + b, & \text{otherwise,} \end{cases}$$

where w and b are learned parameters and cos is the cosine similarity function. This similarity matrix measures the similarity between a game and the centroids of the players in the batch. As we want to maximize the distance to other players and minimize the distance to the player that played the game, the loss of the sample is defined as

$$L(\mathbf{y}_{pm}) = -S_{pm,p} + \log \sum_{q=1}^{N} \exp(\mathbf{S}_{pm,q}).$$

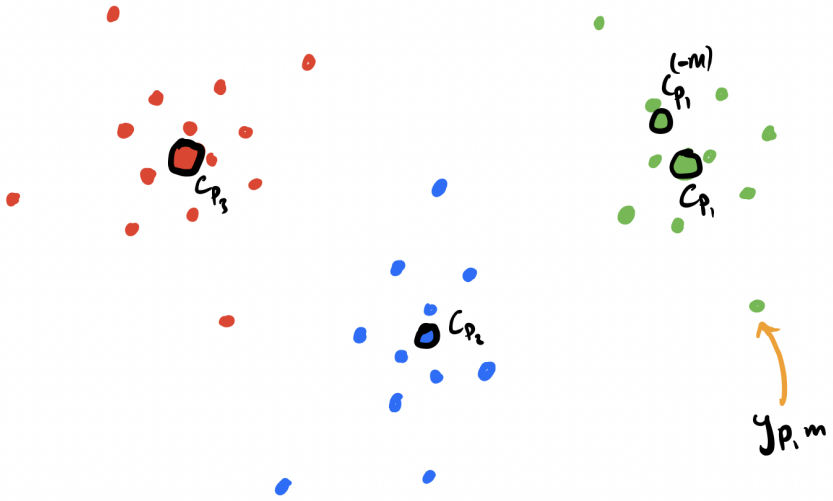


Figure 4: Visual representation of the embeddings and their centroids. The points with black border represent centroids, the upper green centroid is the modified centroid w.r.t. the sample the arrow points at.

Finally, the loss of the batch is defined as

$$L = \sum_{p=1}^{N} \sum_{m=1}^{M} L(\mathbf{y}_{pm}).$$

The generalized end-to-end code was taken straight from Harry Volek's PyTorch Speaker Verification project on GitHub.

2.3 k-shot Learning Routine

To achieve the goal of making the model generalize well to unseen players, we use prototypical learning (Snell et al., 2017). The model learns a mapping from the moves played in a game to a high-dimensional embedding space. The desirable mapping is one that clusters the samples from each player together. At test time, we calculate the embeddings of all of the samples in the reference sets (see section 2.5.1). We create a prototype for each player by taking the centroid of their vectors. To predict who is playing in a sample from the query set, the model calculates the embedding of that sample and outputs the player whose prototype is closest to the embedding.

2.4 Baseline Model

To validate the results of our convolutional LSTM model, we created a simple baseline model inspired by McIlroy-Young et al. (2022). To do that, we used an ad hoc approach in creating the embedding vectors. The model calculates the embedding for each sample by n-hot encoding the first n moves, i.e., the moves that the player played in the first n moves are encoded with 1, and all other possible moves are encoded with 0. Note that we do not take into account the position in which the move was played. For example, if white played the knight from b8 to c6, we do not take into account whether this move was played in the first

move or the n-th move. Here n is a hyperparameter. The prototypes for each player and predictions are then calculated as described in section 2.3. This method takes advantage of the fact that in chess, players often use the same memorized opening moves.

2.5 Data

The data that was used in this project comes from chess games played on Lichess. It was possible to download every game played on Lichess, but the files for that data were around 30 Gb for each month, which would take too long to parse. After some searching, we found the Lichess Elite Database, a database that has been gathered since 2013 of games where players with a Lichess rating of 2400 or higher played against players with a Lichess rating of 2200 or higher, excluding bullet games. For this project, at most 100,000 games per month in the year 2019 were used, totaling around 800,000 games in the dataset. After creating the train/test split (see Section 2.5.1), the training data comprised of just under 400,000 games.

2.5.1 Train/Test Split

To measure the quality of the model, we use two measures. The first goal was to be able to predict who is playing a given chess game out of the players the model was trained on. For the model to be generalizable, we also want to measure the accuracy of the model on previously unseen players, using only a few of their games as points of reference.

First off, the players were split into *seen* and *unseen* players. The *seen* players are the 400 players with the most amount of games in the dataset, ranging from 950 to 9000 games played in the year 2019. The *unseen* players were the players that were not in the top 400 in games played but had played over 100 games in the year 2019.

To create the final splits for the dataset, 100 games from each *seen* player were reserved for test time, and the model was trained on the rest of their data. Out of these 100 games, 50 games were reserved for a reference set, a set to calculate the centroids of the players, and a query set, a real test set to measure the accuracy of the model. For the unseen players, the same split was created for 100 games that they played at random.

2.5.2 Training Time Sampling

Since the training dataset consists of almost 400,000 games, we could not create the tensor representation for every game in the dataset beforehand (that would result in having to store $400000 \cdot 2 \cdot 20 \times 24 \times 8 \times 8 = 2.5 \cdot 10^9$ binary values in memory). Therefore, the tensor representations must be created during training time.

Each batch is sampled at training time as well. The input into the generalized end-to-end loss function (see section 2.2.3) must be on the format $N \times M \times D$, where N represents the number of players looked at in the batch, M the number of games per player in each batch, and D the embedding size of each game. This means that we have to pass in a tensor of size $N \times M \times n \times 24 \times 8 \times 8$ into the model, where n represents the number of moves looked at per game. To create this tensor, N players at random were chosen to compare with each other, and M games from each of these players were sampled for comparison. For each of these games, the tensor representation was created and fed into the model.

3 Results and discussions

3.1 Baseline model

The model was fitted on the reference sets and tested on the query sets for both the seen and unseen players. Note that this model does not need any training, so we do not use the training set. The model's performance is therefore not expected to be better on the seen players when we ignore the fact there are fewer players to predict from there. We ran the tests for four different settings of the hyperparameter N (the number of moves considered). The results are summarized in table 1.

Table 1: Baseline model results

n	**Seen players**			**Unseen players**		
	Random guess	Baseline accuracy	$\frac{\text{Model}}{\text{Random}}$	Random guess	Baseline accuracy	$\frac{\text{Model}}{\text{Random}}$
5	0.25%	3.80%	15.18	0.0535%	1.54%	28.88
10	0.25%	3.29%	13.16	0.0535%	1.67%	31.2
15	0.25%	3.03%	12.12	0.0535%	1.58%	29.48
20	0.25%	2.86%	11.42	0.0535%	1.44%	26.94

3.2 Convolutional LSTM

The model was trained for 40 epochs (100 iterations per epoch) using $n = 10$ and $n = 20$ (i.e., using the first 10 and 20 moves) and inferred on both the *seen* and *unseen* players. They all had $N = 40$ and $M = 20$. Additionally, the model was trained on 5 random moves from the first 20 and 10 random moves from the first 20 (for 15 and 40 epochs, respectively). One of the trained models was used using $N = 10$ and $M = 10$, meaning that in the training sampling, 10 players were sampled along with 10 games from each of the players. That model was trained for 100 epochs since it sees fewer players per batch.

Due to extreme training time and limited project time, the model training could not be fully finalized and tuned, as one epoch took 12-16 minutes. The long training times also made it neccesary to split the training up into multiple instances. Therefore, the loss plots were lost in the process. It would be possible to reconstruct them, but since there were multiple models trained, it would be a hard task. Due to limited resources, the training loss was still decreasing when the training was stopped. Table 2 summarizes the results from each of the different models. We note that our best model (the one with $n = 10, N = 40, M = 20$) has 13.4% accuracy, over 53 times better than a random guess, of predicting who is playing out of a 400-player pool of *seen* players, compared to 3.8% accuracy at best from the baseline model. The unseen player's accuracy of the LSTM model was 2.73%, over 51 times better than a random guess, compared to 1.67% accuracy of the best baseline model. These results show that we were able to obtain some signal from the data.

Table 2: Convolutional LSTM results

n	Type	N	M	Epochs	**Seen players**			**Unseen players**		
					Random	Accuracy	$\frac{\text{Model}}{\text{Random}}$	Random	Accuracy	$\frac{\text{Model}}{\text{Random}}$
5	random	40	20	15	0.25%	2.1%	8.4	0.0535%	1.54%	8.92
10	first	40	20	40	0.25%	13.4%	53.40	0.0535%	2.73%	51.14
10	first	10	10	100	0.25%	8.82%	35.28	0.0535%	1.89%	35.46
10	random	40	20	40	0.25%	8.58%	34.34	0.0535%	1.93%	36.12
20	first	40	20	40	0.25%	12.9%	50.83	0.0535%	2.24%	42.06

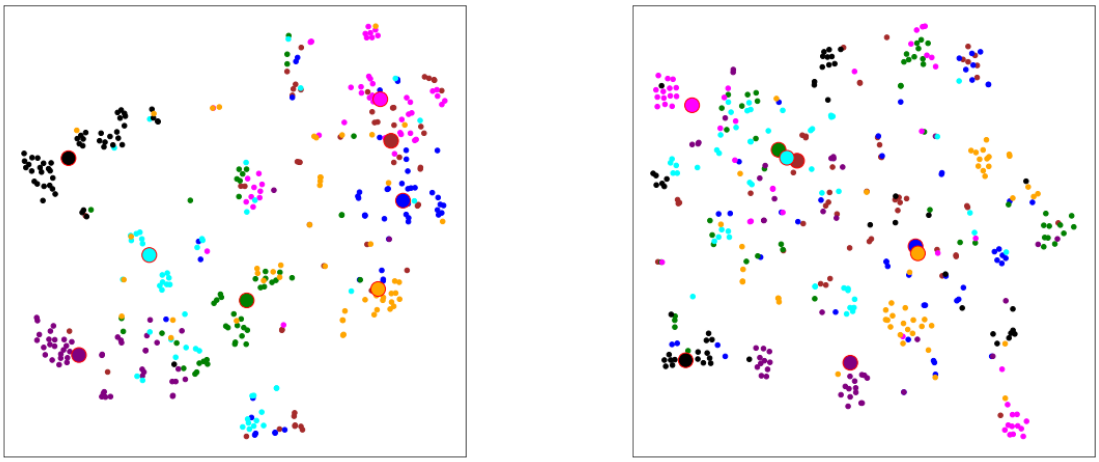
3.2.1 Comparison to Previous Work

In their work, McIlroy-Young et al. (2022) used similar hyperparameters to those that were used in this project, i.e. $N = 40$ and $M = 20$ for the training sampling. They used architecture similar to the one used in this project as well. However, their model was transformer-based. Their main model was trained on Lichess data from players between 1000-2000 Lichess points, each having played over 10000 games, while in this project, only elite players were used (players with 2400 points or more). They however also trained a model on elite players, which can be found in the appendix of their article. It is quite hard to read what their results were from their experiments with a model for the high ranked players. From what can be seen in the appendix, they seem to have obtained a baseline accuracy of over 60% in a pool of around 1600 players, while their high ranking player model obtained accuracy of only 29.8% for *seen* players (1600 player pool) and 29.5% for their *unseen* players (408 player pool). These results are quite different to ours, our baseline model, which was constructed similarily to their baseline, was performing much worse than our LSTM model, while it is the other way around in their article. This might point to some discrepancy in their studies or the result of different data being used. This difference is, however, quite large.

Judging from these results, it seems that elite players are likely to be playing in a similar manner since they most likely use computers when they are training, especially for the early moves. This classification task may therefore be harder than classifying games from worse players. McIlroy-Young et al. (2022) also seem to have had access to considerable computing power, since our model had quite extreme running times and we were not able to train for as long nor with as much data as they did.

3.3 Examples of Embeddings

To visualize the inference of the model, the reference set for 8 players was passed through the trained model for $n = 10, N = 40, M = 20$, and the centroids calculated. Then, the query set for these same 8 players was passed through the model and the embeddings calculated. Finally, t-SNE projections were visualized as figure 5 shows. We see that the *seen* player centroids are spaced apart and the embeddings of a player's game are quite close to the corresponding centroid. For the *unseen* players, the clusters seem to follow this trend as well.



(a) t-SNE representation of games from *seen* players. (b) t-SNE representation of games from *unseen* players.

Figure 5: t-SNE representation of chess games. The larger points with the red border represent the centroids for the reference set and the smaller points represent the embeddings of the games in the query set.

4 Conclusions

The goal of this project was to repeat the experiments made by McIlroy-Young et al. (2022), to accurately predict, from a sequence of chess moves, who is playing with the white pieces and who is playing with the black pieces. With the limited available computing power a student has, we were able to predict with around 13% accuracy who is playing out of a 400 player pool of players the model had seen before. It achieved almost 3% accuracy on unseen players out of a just over 2000 player pool of players the model has not seen at training time, resulting to accuracy over 50 times better than guessing at random and a considerably better accuracy than a baseline model. In comparison to the results in the appendix of McIlroy-Young et al. (2022), our baseline accuracy was significantly lower, and our LSTM model slightly worse as well. However, they seem to have had access to a considerable amount of computing power.

It is important to validate research performed by professionals. After performing this study, we question some of the results achieved in McIlroy-Young et al. (2022). Most notably the results stated in chapter 7.7 in the appendix. They achieve above 60% accuracy in classifying between over 1600 players, with their baseline model. This is on a dataset that is similar to ours. Our baseline model uses the excact same principles, fitting methods and evaluation methods but only achieves at most 4% accuracy.

4.1 Future work

4.1.1 Computational Power

Our work was limited by our computational power. The amount of data we had was more than enough to be able to create models with very high accuracy. However our limited computational resources severly limited our ability to train these models quickly. We trained our final models for over 6 hours, the loss was still decreasing when we stopped the training. We believe that training for a longer time and/or training faster with better resources would results in a better overall model.

4.1.2 Cheat Detection

Our secondary goal was to use these embeddings to detect cheaters in chess purely based on how they play. Due to time limitations we were unable to complete this secondary goal. However, we have a strategy of how to achieve this.

We would get data of chess engines playing many chess games and train the model with the engines and regular players. Having players that are known to have cheated will also help. Then we can compare the unseen players' embeddings and how close they are to the chess engines or known cheaters. Then a simple threshold hyper-parameter to the closeness of the embeddings can be tuned and a model is created.

4.1.3 ELO Rating Prediction

We have data on each players' ELO rating at the time they played each game. An idea we had was to be able to predict the players' ELO rating from the moves. We then switch from a classification problem to an regression problem on the outputs. We could still use the convolutional model for move embeddings as a starting point, We would however need to change the final stages of the model and to get this to work.

References

McIlroy-Young, R., Sen, S., Kleinberg, J. M., and Anderson, A. (2020). Aligning superhuman AI and human behavior: Chess as a model system. *CoRR*, abs/2006.01855.

McIlroy-Young, R., Wang, R., Sen, S., Kleinberg, J., and Anderson, A. (2022). Detecting individual decision-making style: Exploring behavioral stylometry in chess.

Snell, J., Swersky, K., and Zemel, R. (2017). Prototypical networks for few-shot learning. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Wan, L., Wang, Q., Papir, A., and Moreno, I. L. (2017). Generalized end-to-end loss for speaker verification.

Appendix - Failed Attempts

During the process of creating the model, there were a few failed attempts. While we were waiting on the dataset, we trained a binary classification convolutional LSTM model similar to the final model, on data from Magnus Carlsen. We were not able to get accuracy much higher than 50% on classifying if a chess game is played by Carlsen or not. This is probably since there were many thousands of players in the dataset, and we only labeled them as „Not Carlsen", so there was not much pattern to the players that were not Carlsen.

Another failed attempt was to introduce so-called squeeze-and-excitation blocks to the residual CNN in the convolutional LSTM model. These blocks are used in the architecture we were inspired by, Maia (McIlroy-Young et al., 2020). When we introduced them to the model, the model performed worse.

One final mistake that we realized the day before we had the presentation was that we in the beginning, we used Euclidean distance for model inference, but the model was trained on cosine similarity. Using cosine similarity gave a slight improvement over the Euclidean distance, but the gain was not large.