Pázmány Péter Katolikus Egyetem

Információs Technológiai és Bionikai Kar

Quantifying Complexity in Chess Positions

Guided Individual Study

Dominik Csatári, FV1TW4

Software Engineer Msc

2024

Contents

Chapter 1

Introduction

In this chapter, I will explain how the complexity of a board game is typically measured. I will then discuss why determining the complexity of a chess position is more challenging. I will present an existing method for assessing complexity and highlight its limitations.

1.1. Complexity

1.1.1 Complexity Introduction

When discussing the complexity of a game, we typically refer to **state-space complexity**, which measures the number of legal positions reachable from the initial position. This concept leads to the game tree size, which represents the total number of possible games originating from the initial position. In chess, the game tree is extraordinarily large due to the immense depth and breadth of these trees, as well as the substantial state-space complexity of the game. To determine which side is winning in a given position, it is not feasible to calculate the entire game tree. Therefore, modern models typically analyze up to a certain depth and then evaluate the position using heuristics. Intuitively, these solutions have enabled supercomputers and advanced chessbots to achieve an Elo rating of 3800, which is over 1000 points higher than the level of human grandmasters [1].

1.1.2 Importance of position complexity

As I have mentioned in Subsection 1.1.1, computer evaluation often exceeds human comprehension. This concept is also discussed in chess podcasts and papers, where moves are referred to as "human-like" or "non-human-like." Non-human-like moves are those that are likely difficult for even a grandmaster to find, as they may seem illogical or counter-intuitive in the given position. This is why, when you analyze your game and mistakes with a computer, it cannot fully capture or account for the difficulty of finding a move in a complex position. This computer supremacy is also evident in the accuracy score, which provides a percentage indicating how accurate your moves were during the game. This accuracy score is also used in the chess site Lichess [2]. In their description they state: *"While Stockfish can assess the soundness of our moves, it can't tell us how difficult it is to find them."* [2] It would be great to somehow factor in a position's difficulty or complexity, or how hard it is to find a move. With this kind of measurement, we could weigh mistakes or moves in the game based on the position's complexity. This way, we could better reflect how tough it is to find a good move, making the evaluation feel more human-like.

1.1.3 Elocator

Elocator is a tool that provides a complexity value for a given position. This complexity value is between 0 and 10 [3]. It is supposed to tell how complex a position is. The complexity is calculated as the expected change in the win rate. This is a very similar calculation to that done in Lichess's accuracy score. Elocator neural network was trained on grandmaster moves, and it learned the expected win rate after a move is made. The win rate is based on Stockfish 16 at depth 20. The resulting complexity values are the predictions of this model. I have attached three figures that show how the Elocator defines complexity for low, medium, and high-complexity positions. These figures are 1.1, 1.2, 1.3 [3].

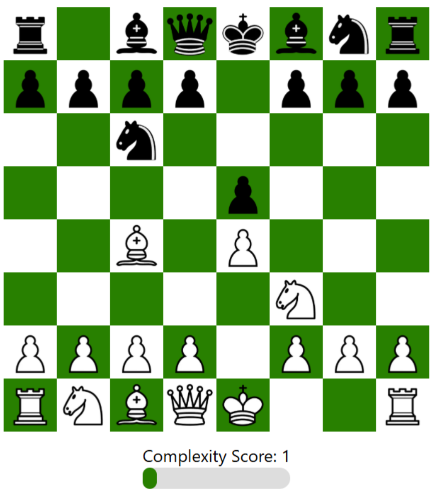


Figure 1.1: Elocator: Chess Complexity Calculator predicting **low complexity**.

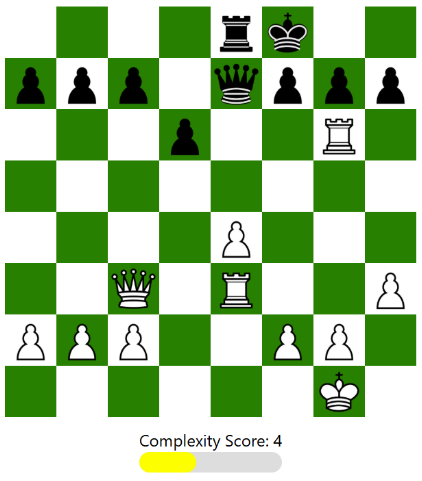


Figure 1.2: Elocator: Chess Complexity Calculator predicting **medium complexity**.

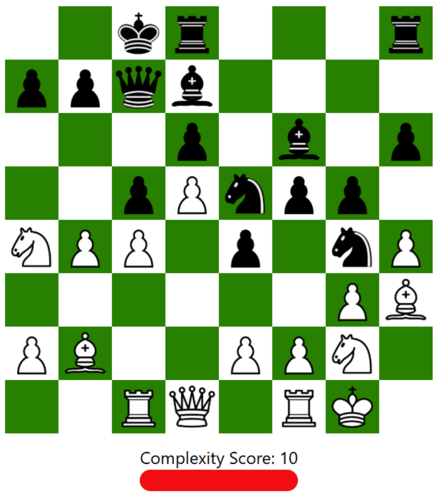


Figure 1.3: Elocator: Chess Complexity Calculator predicting **high complexity**.

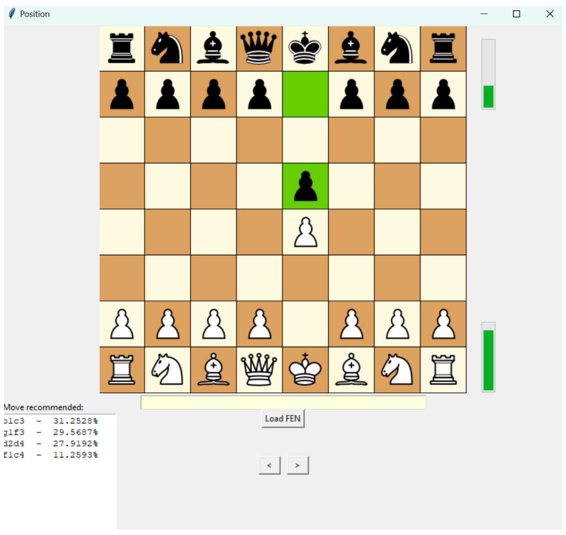
As it is shown in these images, the complexity this tool defines, is not exactly a solution for what was described in Lichess accuracy description. For example, medium complexity is illustrated for Black in Figure 1.2, where Black actually has only two sensible moves, one of which is capturing the rook on *g6*. Also, for the example lower complexity,, shown in Figure 1.1, they show a very low complexity, however there are a lot of possible moves. Since this is at the very beginning of the game - the opening phase - there will not be significant changes in the win rate. However, the diversity of possible moves is still expected to indicate a higher complexity level.

Chapter 2

My implementation

In this chapter, I will describe my implementation of a neural network for evaluating the complexity of chess positions. I will cover the necessary components I encountered and highlight the aspects of the project that proved useful during the development process.

2.1. Visualization



```
blc3  -  31.2528%
glf3  -  29.5687%
d2d4  -  27.9192%
flc4  -  11.2593%
```

Figure 2.1: This figure shows the application I created, allowing users to interact with the game.

As the matrix representation of a chess board is not user-friendly, I have decided to create an application where I am able to interact with the board, change the position, and so on. For this, I have used a library called **Tkinter**, which helped me to provide a great representation of the board [4]. This application can be seen in Figure 2.1. As it is shown on the bottom left corner, there is a table, for recommended moves. The user is also able to read a position in FEN form, than play it on. On the right side of the board,there are two bars, placeholders for the complexity, that will be calculated. In this case, they are random numbers between 0 and 10. Additionally, the user can choose a board style from a set of 8 different styles.

2.2. Predicting Complexity

To predict a game's complexity, I need to define three additional aspects: database, the network I am using, and how the complexity values are aggregated.

2.2.1 Database

I have used Lichess database of 90 million positions. Each position is evaluated with Stocksifh. The position itself is provided in FEN form **??**.

2.2.2 Network

For the network, I have decided to use 3 models. Each called *ChessExpert*, with a number from 1 to 3 indicating which specific model I am referring to. All of them are loosely based on the ConvChess architecture. While my setup is entirely different, the idea of using convolution operations to create feature maps of the board seemed like a great approach.

ChessExpert1

This model was created as a proof of concept to determine whether my structure could generate and predict sensible moves. For this I have used a simple architecture:

- **Input:** It consists of 6 planes for White pieces and 6 planes for Black pieces. Each plane contains zeros in squares without the corresponding piece type and ones in squares occupied by that piece. the last two planes are for places you can move and place you can move to.

- **Convolutional Layers:** I use three convolutional layers, where the number of filters increases with each layer, but the spatial dimensions remain the same due to padding. After these layers, the final output is flattened to create a global feature vector.

- **Fully Connected Layers:** This part consists of fully connected layers. The first layer takes the global feature vector as input, and the output of the second layer is the predicted output vector.

- **Output Vector:** This output vector is sized 4096 and it contains all the possible moves from all the possible squares. It is similar to the AlphaZero structure [1].

- **Target:** The target is created using predicted moves generated by Stockfish. The four best moves are selected, provided they are not worse than the first move by more than 40 centipawns, or if the first move is not a checkmate. In these cases, fewer moves are chosen. The centipawn difference then converted into probabilities, with an exponential transformation applied. This means that the further a good move is from the best move, the lower its probability becomes. I would like to receive a similar probability vector as the predicted output.

I trained this network using Stochastic Gradient Descent (SGD) with a learning rate of 0.1, along with a scheduler that reduces the learning rate by a factor of 0.1 after the validation loss fails to decrease over 6 steps. For loss I have used KL divergence loss. I have tried multiple losses, this performed the best. I have used 30 000 000 positions through 10 epochs. The results were great regarding the loss and the predictions as well. The predictions were great, mostly at the beginning of the game, in the opening. One prediction type can be seen in Figure 2.1. So the results proved, that this model is able to create a sense of features of the positions.

ChessExpert2

This model is similar to the previous network in terms of its core structure. However, there are two main differences. The network uses *Squeeze-and-Excitation blocks* to emphasize each layers importance after a given convolutional layer. Additionally, I modified the first Squeeze-and-Excitation block by removing the squeezing step, allowing it to better identify the important layers of the input data. Also, the output vector's length is only 6. This is because I want to only predict choosing a piece with this model. Creating the target follows a similar process to that of the previous model.

For training, I have chosen to use Adam optimizer with a learning rate of 0.001, dropping to 0.0001 after if the training loss does not decrease for at least 6 steps. I have trained it with more epochs and less positions. I have trained it with 60 epoch having 10 000 000 positions. The results for the validation a training loss is attached in Figure **??**.

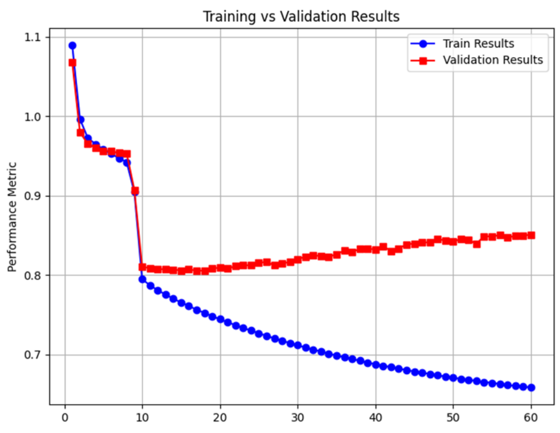


Figure 2.2: This the loss for *ChessExpert2*. The x values are the number of chunks processed and the y values are the corresponding losses.

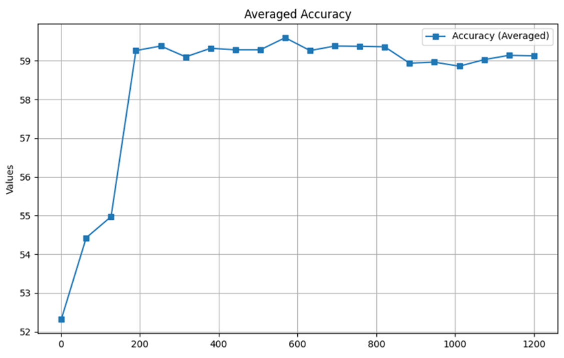


Figure 2.3: This the accuracy for *ChessExpert2*. The x values are the number of chunks processed and the y values are the corresponding accuracies for validation set.

The accuracy it showed during training is shown in Figure 2.3. For the accuracy I have chosen the maximum for both target and predicted output. As it is shown in Figure 2.2, the validation loss stopped decreasing at around epoch 10. After that the model overfitted. In Figure 2.3 we can see, that the accuracy is around 60%, which is a great value overall.

ChessExpert3

This model is used with the same architecture as *ChessExpert2*, however the output is sized 64 not 6.The size of 64 corresponds to the possible types of moves. This includes 8 moves for the Knight and 56 moves for the Queen moves (all piece movements can be covered with queen movements except for the knight). The training went well. I have used more positions and epochs as the loss was still decreasing after 100 epochs. This can be seen in Figure 2.4. The accuracy is attached in Figure 2.5. The accuracy was 40%,

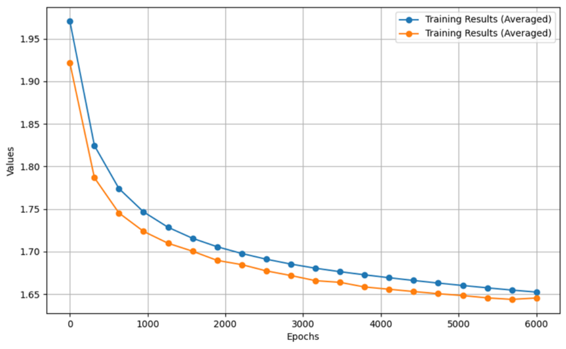


Figure 2.4: This the loss for *ChessExpert3*. The x values are the number of chunks processed and the y values are the corresponding losses.

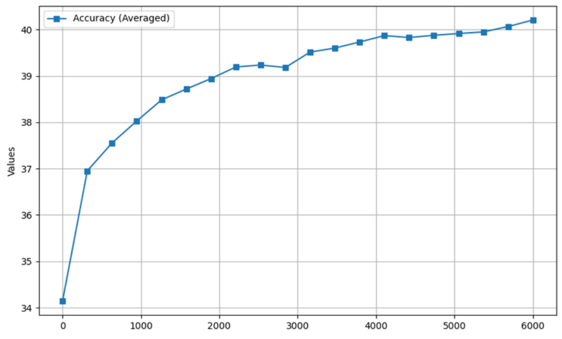


Figure 2.5: This the accuracy for *ChessExpert3*. The x values are the number of chunks processed and the y values are the corresponding accuracies for validation set.

which is a great value for detecting only the best moves. If I had continued training, the loss would have decreased further; however, I decided to stop and accept this value due to time constraints.

2.2.3 Predicting Accuracy

For predicting accuracy, I have decided to try three metrics. First, I wanted to see how the first SE_blocks are correlated for different piece choices. However, as far as I have seen it, most of them have a similar value, all layer is multiplied with 1, or similar so it did not tell that much of a difference that I have expected it to do. After this, I tried to create nearest neighbors for the predicted vectors, labeled with their highest probability piece. However, this approach simply overcomplicated things and was not robust enough.

My last metric for calculating complexity from the output vectors, was entropy. i have calculated the normalized entropy for *ChessExpert2* and *ChessExpert3* and then I have averaged them. It is a great representation because when there are more choices with higher probabilities, the entropy increases, which is the case in complexity level as well. I have created a survey for chess players and enthusiasts. This survey had 5 chess positions attached in Appendix 4 in Figures 4.1, 4.2, 4.3, 4.4, 4.5. The results for it are attached in Table 2.1. I have also calculated complexity values for the position shown in Figures 1.1, 1.2, 1.3. The values for these from my model were **1.64**,**5.02** and **5.95**.

	Elocator	**My model**	**Spectators**
Position 1	**3**	**4.55**	**4.09**
Position 2	**6**	**2.11**	**5.3**
Position 3	**2**	**5.53**	**4.09**
Position 4	**4**	**7.39**	**5.81**
Position 5	**4**	**3.96**	**4.09**

Table 2.1: Table with calculated complexity values.

Chapter 3

Conclusion

For my Guided Individual Study I wanted to create a neural network, which predicts a position's complexity based on only the position. As I have shown the results in Subsection 2.2.3, My model is different than the Elocator's predictions.However, for me, my model's predictions make more sense. In most cases, they matched the values I would have assigned to the example positions. I have managed to quantify complexity based on convolutional models that understand the basic patterns of positions.

In the future, this method could be used to weight the accuracy score of a game based on the complexity of each position. Additionally, it could be applied to predict the complexity of a position, and if the complexity is high, a larger tree search width could be employed, as there are more possibilities to consider.

Chapter 4

Appendix

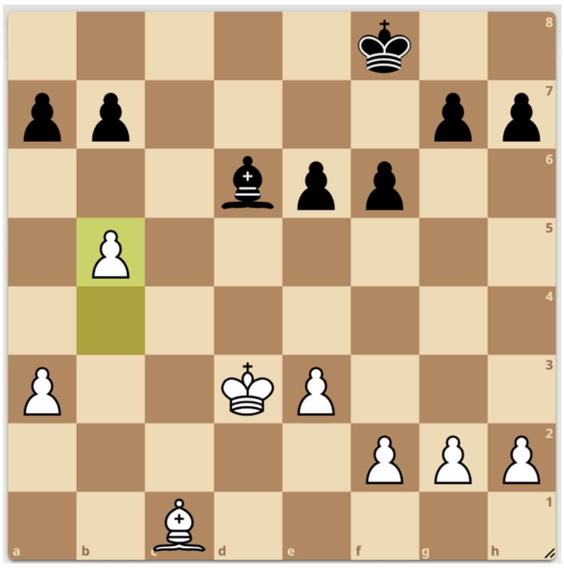


Figure 4.1: **Position 1** from survey.

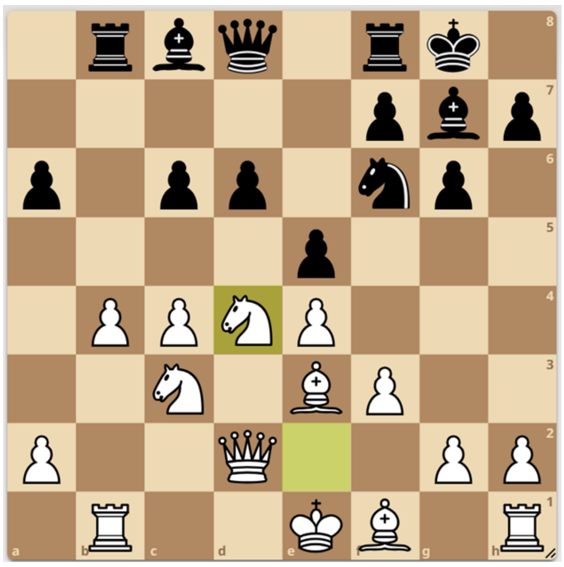


Figure 4.2: **Position 2** from survey.

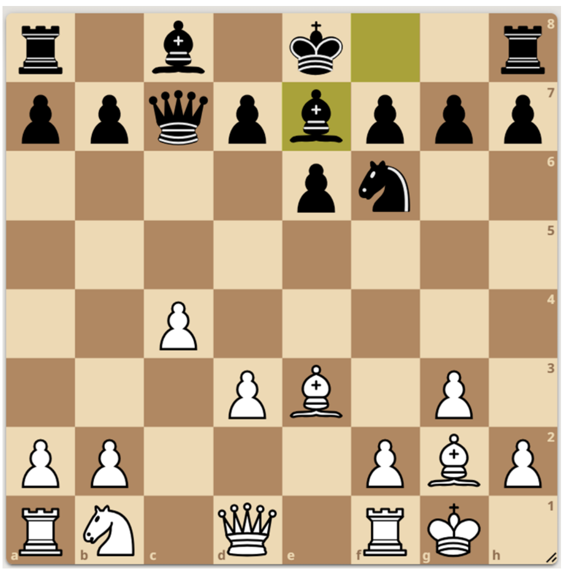


Figure 4.3: **Position 3** from survey.

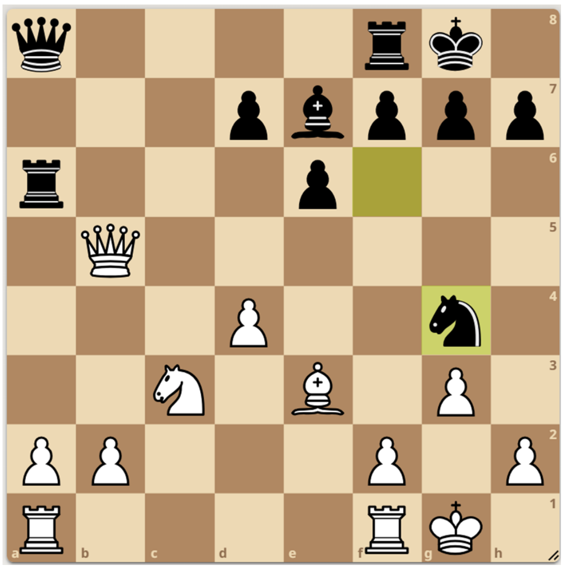


Figure 4.4: **Position 4** from survey.

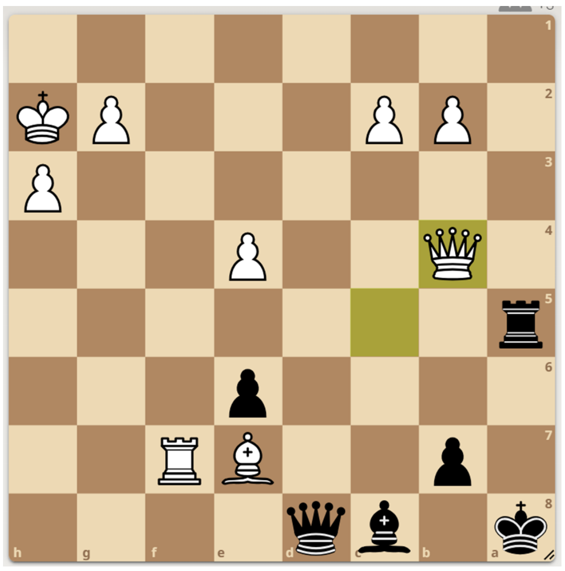


Figure 4.5: **Position 5** from survey.

Bibliography

[1] D. Silver, T. Hubert, J. Schrittwieser, *et al.*, *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*, arXiv:1712.01815 [cs], Dec. 2017. DOI: `10.48550/arXiv.1712.01815`. [Online]. Available: `http://arxiv.org/abs/1712.01815` (visited on 12/10/2024).

[2] *Lichess Accuracy metric • lichess.org*. [Online]. Available: `https://lichess.org/page/accuracy` (visited on 12/10/2024).

[3] cmwetherell, *Cmwetherell/elocator*, original-date: 2024-01-08T01:05:17Z, Jun. 2024. [Online]. Available: `https://github.com/cmwetherell/elocator` (visited on 12/10/2024).

[4] P. Jadhav, *ParthJadhav/Tkinter-Designer*, original-date: 2021-05-18T07:29:26Z, Dec. 2024. [Online]. Available: `https://github.com/ParthJadhav/Tkinter-Designer` (visited on 12/10/2024).