International Journal of Engineering Trends and Technology
ISSN: 2231–5381 / https://doi.org/10.14445/22315381/IJETT-V72I4P105

Volume 72 Issue 4, 43-50, April 2024
© 2024 Seventh Sense Research Group®

Original Article

Deep Learning Approaches to Predicting the Optimal Chess Moves from Board Positions

Muhammad Faiz Arsalan[1], Haryono Soeparno[2]

[1,2]*Department of Computer Science, Master of Computer Science, BINUS University, Jakarta, Indonesia.*

[1]*Corresponding Author : muhammad.arsalan@binus.ac.id*

Received: 21 November 2023 Revised: 29 March 2024 Accepted: 09 April 2024 Published: 24 April 2024

***Abstract* - *In this study, the authors explore advanced methodology, which consists of two methods to predict chess strategies utilizing a neural network approach: tensor construction and a novel method, which is the move-to-delta framework. The approach commences with dataset curation from the esteemed Lichess database, transitions through tensor construction using a refined piece-centric representation, and an innovative model training underpinned by the "move-to-delta" conceptual framework. Pivotal components of the methodology are the strategic utilization of seed variability ranging from number 1 to 100 and exploring the impact of four different batch sizes (64, 128, 256, and 512), illuminating the nuanced interplay of weight initialization in neural training. The model's performance is evaluated using two evaluation methods: the number of puzzles solved and performance metrics (MSE, MAE, and R-squared). Notably, the model initialized with seed number 33 and batch size 128 achieved exceptional capability, solving four positions out of the 25 Kaufman Test puzzles. This signifies an achievement that significantly surpasses extant chess engines, which, at best, resolve two Kaufman puzzles. This finding underscores the essential role of weight initialization, the usage of the move-to-delta framework, and the value of rigorous experimentation in the realm of chess move prediction through deep learning.*

Keywords - *Chess, Convolutional Neural Network, Deep learning, Kaufman Puzzle, Supervised learning.*

1. Introduction

Predicting the outcome of a chess game has been the pinnacle of testing the performance of artificial intelligence. Some use machine learning, such as the hidden Markov model [18], and some use MLP. In the recent decade, deep learning has changed the scene for chess engines. The crossroads of artificial intelligence and chess has been a transformative force in computer science. Renowned models like AlphaZero [1] and DeepChess [2] epitomize this evolution, with their ascent tracing back to pioneering efforts like the Giraffe engine in 2015 [3].

At their core, modern chess engines are more than just computational tools—they are deep strategic partners. Benefiting from the synthesis of human experience and machine precision, these engines dissect vast potential outcomes and reshape the fabric of chess strategies, pushing the boundaries of traditional human intuition.

Chess, a game marked by its intricate nuances and strategies, thrives on acute decision-making. The limitations of initial brute-force engines illuminated the need for nuanced, sophisticated approaches that could mirror the complexity and depth of the game itself. Historically, the allure of chess has been its intellectual challenge. With the advent of technology, this allure has expanded, encompassing the intriguing dance between human minds and computational might. This dynamic duo has catalyzed advancements across algorithms, methodologies, and evaluative parameters in computational research.

With all recent advancements, there are still inconsistencies in the results. Two Kaufman positions were solved using MLP [9]. Moreover, they claimed that the model that used MLP was far superior to the model using CNN. However, in a more recent paper, the model that used MLP was not able to solve a single Kaufman position [11]. This indicates that the results from both papers are not consistent.

These results sparked the investigation of which techniques and methods can produce consistent results. This paper aims to explore deep learning approaches to discover the best techniques and methods for finding the best optimal moves by using board positions.

2. Literature Review

2.1. Chess Engine Architecture

Chess engines have undergone a remarkable evolution, transitioning from early brute-force methods to sophisticated combinations of heuristics and algorithms.

The architecture of a typical chess engine consists of several key components that work in harmony: board representation, move generation, search, and evaluation [4].

There are numerous types of board representation. It is mainly divided into three groups: piece-centric, square-centric, and hybrid. A piece-centric representation maintains lists, arrays, or sets of all pieces that are still on the board, along with information about the squares they are now occupying. A square-centric representation applies the inverse association, checking whether a square is vacant or not. Lastly, hybrid representation typically mainly uses the square-centric representation with a list of pieces [5].

The evaluation function is a core component that assesses the strength of a given position. Heuristics such as material balance, piece square tables, mobility, and king safety contribute to the evaluation score. However, this score is often based on shallow calculations that do not capture complex positional nuances. The search component employs algorithms to search all possible moves.

2.2. Deep Learning in Chess

The application of deep learning in chess has opened new avenues for addressing the limitations of traditional evaluation functions. Neural networks, which consist of interconnected layers of artificial neurons, have demonstrated the ability to learn complex patterns and relationships from data. Various common deep learning models that are used are MLP, CNN, and RNN. One of the ways to evaluate these models is to use the Kaufman test.

2.2.1. Kaufman Test

The Kaufman test is a set of problems that was proposed by Lawrence Charles Kaufman, an American chess Grandmaster. Kaufman proposed 25 problems that are intended to test the strength of a chess engine. He published these problems in a report called the Computer Chess Report (CCR).

He published the first 20 problems in 1992 [6] and the last 5 in the late-1992/early 1993 [7]. Later on, because of the popularity of the problems, they reprint the whole set in a later edition of the CCR [8]. The Kaufman Test is often used to evaluate the performance of chess engines.

2.2.2. Deep Learning Models

Multilayer Perceptron (MLP) models have been used as a means to evaluate chess positions. The research was conducted with the aim of evaluating position [9] using 3.000.000 unique chess positions that are played by top chess players and utilizing a limited lookahead searching algorithm. Algebraic and bitmap were used as the input. Based on their research, the performance of MLPs trumps CNNs in terms of architecture in chess with 96%, 93%, and 68% accuracy across 3 out of 4 datasets. When tested using the Kaufman test, the model is only able to solve two positions, which are position numbers 3 and 6.

Another research conducted by [11] used thousands of games that were parsed and created the board representation. Each of those games was given a centipawn score generated by the Stockfish 10 engine. The first two hidden layers had 2048 neurons, and the rest of the four hidden layers had 1024 neurons, with a total of six hidden layers that made up the architecture. Each of those hidden layers used Rectifier Linear Unit (ReLU) as the activation function except the output layer that utilized hyperbolic tangent. The model failed to solve the Kaufman test.

The use of limited lookahead is also an interest in [10]. The difference between Maesumi's paper and Sabatelli's paper is the dataset, and instead of labeling it manually, he used a deep autoencoder. Features that are used by Maesumi are the positions of each piece type, the turn, castling rights for both colors, and whether a position is in check. In total, the board representation is a vector with 775 binary features. Like other papers, Maesumi used centipawn (cp) as the evaluation number. He classified the position into 3 types: Black winning (cp less than or equal to -150), drawish (cp is less than or equal to 150 or more than or equal to -150), and White winning (cp is more than or equal to 150). The model that Maesumi proposed has some lookahead information in its evaluation. When the output of his model and Stockfish are compared, his model can assess the positions without any lookahead algorithm. Next, he made a search algorithm with his model as its core. When the search algorithm is used with a depth of 5, 83% of the moves chosen have the same strength as moves chosen by Stockfish at depth 23.

The integration of Convolutional Neural Networks (CNNs) in chess analysis has been particularly impactful. CNNs are well-suited for spatial pattern recognition, making them adept at processing board positions. They can learn meaningful features from raw board representations, such as bitboards or encoded states, and capture intricate patterns that contribute to position strength.

The researchers [12] used bitmap input based on the position of this research for each unique piece. It is then converted to make a chess engine using convolutional neural networks (CNNs). It was trained using 20.000 games 8 x 8 x 6 images that have an ELO rating above 2000.

The architecture of this corresponds to the width and height of the board and the six unique pieces that are denoted with +1 and -1, white and black pieces, respectively. The network consists of two parts: the move selector and the piece selector. 26 games out of 100 games were drawn, and lost the rest when it was faced with the Sunfish chess engine. The author claimed that CNNs are useful in pattern recognition of small tactics.

In a similar manner [17], bitmap input was used to represent the chess board. The neural network that used the bitmap input has better results than using the algebraic input. In another paper [16], an 8 x 8 x 12 arrays are also used as the input of a recurring neural network. An 8 x 8 mask of the target and origin position of a chess piece is utilized as the output of the network.

CWU-Chess [15] used a different approach – using hand-crafted features as the input of the neural network. 10 distinct features were chosen: bishop pair, double pawns, numerical advantage, isolated pawns, pawns advance, passed pawns, mobility, defensive coordination, center control, and king safety. Genetic algorithm was chosen instead of the traditional CNNs. The chess engine can play the games within the first 4 generations.

[13] researched a similar game with Chess, Hex. The authors show that a compact representation utilizing the most common bridge pattern can achieve reasonable accuracy. The neural network is integrated with Monte Carlo tree search to enhance performance. Challenges include imperfect training data and the non-trivial combination of neural nets and search. The best accuracy achieved on test data is 54.8%. In addition, the paper shows that the neural network can achieve reasonable playing strength without the need for a search function, which is a significant advantage over traditional methods.

Go is another game that is often used to train deep learning models. A research paper [20] used an ensemble of CNNs as their deep learning architecture. The model was able to predict 36.9% of the expert moves. In a more recent paper [19], instead of ensemble methods, CNNs are used to train. The model can accurately predict 55% of the positions. Furthermore, it beats GnuGo (a search-based program) in 97% of games played.

Lastly, [14] Chess2vec is an innovative approach that converts chess pieces into vectors for move prediction and analysis. Using the matrix representation with position-dependant piece vectors, a multiclass test was conducted that accurately predicts 8.8% of the moves of Stockfish. These diverse approaches collectively contribute to the ongoing evolution of AI in the realm of chess, offering insights and advancements that continue to shape the field.

3. Data and Methods

This paper aims to develop a model that can determine the optimal chess moves from board positions. The proposed solution uses a novel framework namely the move-to-delta framework with chessboard positions that is integrated into a deep learning model. The deep learning model is trained using seed initialization and a single hyperparameter tuning which is the utilization of four different batch sizes (64, 128, 256, and 512). Callbacks are utilized to aid with the training process. EarlyStopping to stop the training process early based on the improvement of the metric that is used, ReduceLROnPlateau to reduce the learning rate on metric that plateaus and ModelCheckpoint to save the best model.

The training process starts with collecting and preparing the dataset. Utilizing the dataset, it is converted into the inputs for the model: tensor to represent the state of the chessboard and evaluation of the chessboard converted using the move-to-delta framework. After that, the dataset is split into training and validation datasets with an 80:20 ratio, respectively. The training process is divided into four based on the number of batch sizes. Each of those batch sizes trains the model by utilizing seeds ranging from 1 to 100. Each seed is trained using the Adam optimizer and 100 epochs outputting in a total of 400 different models.

3.1. Software

To realize this solution, a software environment must be used. The main programming language that is used is Python. Utilizing the various available packages, a few were chosen: pickle, which is a file type to store the converted dataset, TensorFlow which is a Python package to help create deep learning models, NumPy which is a Python package that contains a lot of mathematical functions, scikit-learn to help with the preprocessing, and lastly, python-chess which aids with any chess related problems.

3.2. Dataset Preparation

Initially, the dataset is acquired in Portable Game Notation (PGN) format, which includes chess games with move sequences and outcomes. The dataset is sourced from the Lichess database website, which houses 4 billion standard-rated games and 3 million puzzles. For this research, we selected data from “2023-March” and further filtered it only to include games with the Grandmaster title. The filtered set contains 3,862 games out of 108,201,825.

3.3. Tensor Construction and Mapping

Subsequently, this PGN data is converted into board representations using a piece-centric representation type. This board representation method encompasses piece types, their colors, and blank spaces. Tensor building involves converting these board representations into tensors suitable for neural network processing. This is achieved using a mapping that translates board pieces into numeric values, forming the input tensors. At the heart of this process lies the Forsyth–Edwards Notation (FEN), a standard representation system that captures the current state of a chess game. A typical FEN string succinctly represents piece positions, active color, castling rights, en passant target square, half-move clock, and full-move number. For instance, the starting position of a chessboard is represented as "rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1".

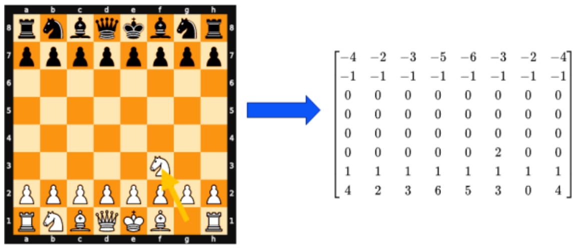
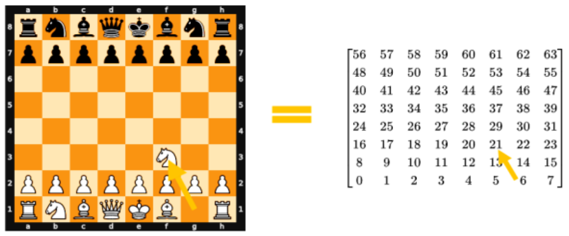


Fig. 1 Board representation

To transform the FEN notation into tensors, we utilize a predefined mapping system. Each unique character in the FEN, representing different chess pieces, is mapped to a specific numeric value in the range of 1 to 6 (see Figure 1). The value's sign indicates the piece's color: positive for white and negative for black. For instance, 'r' (black rook) is mapped to -5, while 'R' (white rook) is mapped to 5. Empty squares in FEN, denoted by numbers, are expanded into consecutive zeroes. So, '4' translates to '0000'. Using this mapping, the FEN string is then translated into a matrix. In the context of an 8x8 chessboard, this results in an 8x8 tensor. The values in this tensor, derived from our mapping, serve as inputs for the neural network.

3.4. Move-to-Delta

Traditional evaluation methods often look at board positions in isolation, neglecting the fluidity and dynamism of the game of chess. However, with the "move to delta" concept, we directly target the outcome of individual moves. This approach inherently factors in the broader game strategy and the ever-changing nature of chess battles. At the core of our training process is a unique labeling approach, termed the "move to delta" concept. It serves as a pivotal tool in understanding and quantifying the inherent value and strategic ramifications of each potential chess move. Here is how it functions: the model compares the evaluation score of a chessboard position before a move is made to the score after its execution (see Figure 2). This differential — the delta — encapsulates the move's efficacy and becomes the label for our training data. By training our model using these delta values, we enable it to grasp not just the intrinsic value of board positions but also the nuanced implications of each move within the grand scheme of the game. This gives the neural network a holistic understanding, sharpening its ability to recommend moves that could significantly shift the balance of power in a match.


Nf3 = 21 – 6 = 15
Fig. 2 Move-to-Delta

3.5. Evaluation Methods

After the training process, it is important to evaluate the models. The models will be evaluated in two methods: subjected to 28 chess puzzles and performance metrics. To adapt to real-world application and reliability, the model is subjected to 25 Kaufman tests, Plaskett's puzzle, and the two most common checkmate patterns: the Fool's mate and the Scholar's mate. Each of the puzzles is paired with the best move so that the model's best move can be compared to the best move of the puzzle.

In the beginning, the puzzle will be inserted into a search algorithm that searches all of the possible legal moves. Then, each of those legal moves is evaluated using the trained model to output an evaluation score. After that, the best evaluation score is matched against the best move from the puzzle. The best model is the model that solved the highest number of puzzles.

The other evaluation method is to measure the performance metrics of each of the models. The performance metrics that are used are Mean Squared Error (MSE), Mean Absolute Error (MAE), and R-squared (R^2). The MSE measures the average squared difference between the predicted value and the actual value, whilst the MAE measures the absolute difference between the predicted and the actual value formulated in Equations 1 and 2, respectively. R-squared measures a bit differently (see Equation 3). R-squared determines whether the model is a good fit or not. Within the range of 0 to 1, 1 is the best-fitted model, while 0 is the worst-fitted model.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{1}$$

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{2}$$

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2} \tag{3}$$

3.6. Model Architecture

A Convolutional Neural Network (CNN) is structured for the interpretation of 8x8 grid representations, a format that corresponds with the dimensions of a chessboard (see Figure 3). The network commences with an input layer to accept the predefined shape of the data. It integrates two convolutional layers with Rectified Linear Unit (ReLU) activations, interposed with a max pooling layer for the abstraction of data and a dropout layer to reduce overfitting risks.

A flattening layer transitions the network from two-dimensional feature maps to a one-dimensional vector. This vector feeds into a dense layer with ReLU activation for nonlinear transformation, followed by another dropout layer for regularization. The architecture concludes with a dense output layer, which serves to evaluate the positional strength of the chessboard which outputs an evaluation score.

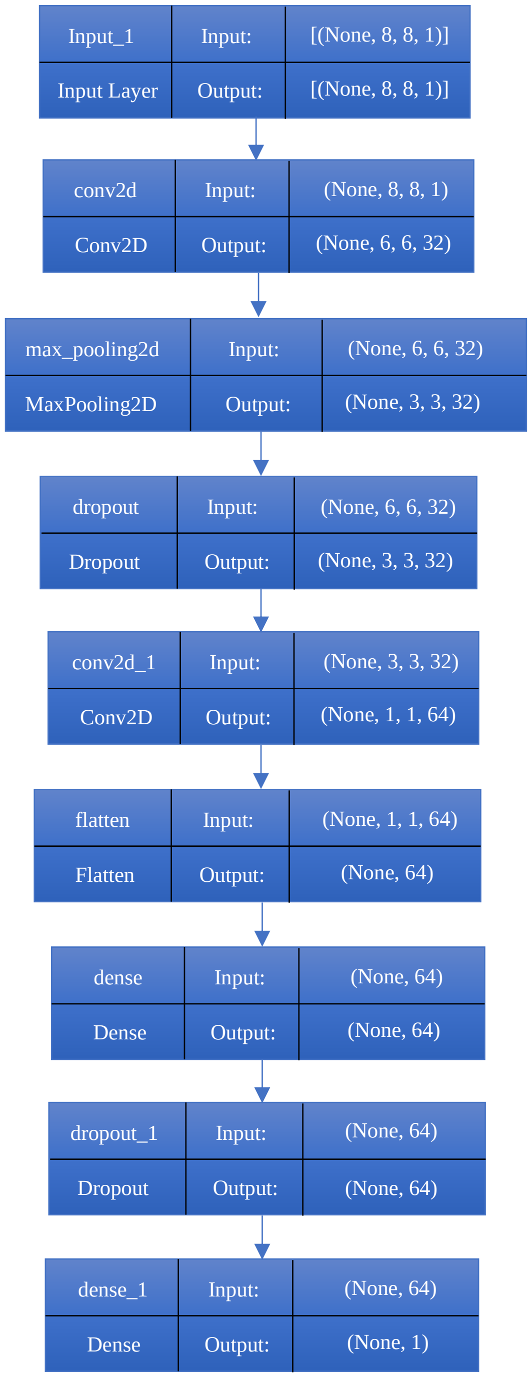


Fig. 3 Model architecture

4. Results and Discussion

Since the model is subjected to two methods of evaluation, the results are divided into three, with the discussion session after the first two results from the two evaluation methods. The first section consists of the results from the models that are subjected to 28 different puzzles. The second section consists of the performance metrics from the trained models. Lastly, the last section contains a discussion of both methods and additional observations.

4.1. 28 Puzzles Result

After the model is trained, all of the models are subjected to 28 puzzles with the goal of finding the model that has the highest number of puzzles solved. Figure 4 illustrates the total correct positions for each of the seed numbers for each batch size. The seed numbers are on the x-axis, while the total correct positions lie on the y-axis. Each seed number consists of 4 colored bars representing the batch sizes – the colors blue, orange, green, and red representing batch sizes 64, 128, 256, and 512, respectively. Next, Figure 5 illustrates the frequency of each position. The puzzle number is fitted on the x-axis, while the total correct positions are on the y-axis. Not all of the seed numbers yielded results; hence, the seed numbers do not have the full range from 1 to 100 illustrated in Figure 4. Furthermore, not all batch sizes within a seed number yield results. Only a few seed numbers that have all of the batch sizes solved at least 1 puzzle. Lastly, two models solved the highest number of puzzles: seed number 33 with a batch size of 128 and seed number 65 with a batch size of 64. Each of those models is able to solve 4 different puzzles. Both models are able to solve puzzle number 18, with puzzle number 20, 22, and 24 solved by the first model and puzzle numbers 9, 11, and 21 by the second model. 14 puzzles out of 28 puzzles were solved by various seed numbers illustrated in Figure 5. It consists of 12 Kaufman puzzles, the Fool's Mate, and the Plaskett's Puzzle. Not all positions are present on all of the batch sizes. Position 5 and 22 are not present on batch sizes 128 and 512, position 23 is only present on batch size 128, and puzzle 28 is only present on batch size 64. The highest frequency of the number of puzzles solved is puzzle number 18, especially on the batch size of 128. The puzzle numbers that are present are 5, 9, 11, 16, 18, 19, 20, 21, 22, 23, 24, 25, 26, and 28.

4.2. Performance Metrics

The performance metrics revealed an interesting observation. Using Table 1, the model is measured using MSE, MAE, and R-squared. It is evident that within the batch size of 64, the best model lies in seed number 91, with the lowest MSE and MAE and the highest R-squared. This pattern is also observed within the batch size of 256 where the lowest MSE and MAE, and the highest R-squared are in a single seed number, in this case, seed number 23 with the metrics of 164.352, 9.56, and 0.1343, respectively. This pattern breaks on the other two batch sizes where no seed number dominates the performance metrics. It is also evident that there is another pattern that is present. All of the batch sizes except for batch size 64 have seeds that have both the lowest MSE and the highest R-squared. Batch size 128 with seed number 80, batch size 256 with seed number 256, and batch size 512 with seed number 13. Further observations revealed that there is yet another pattern where the seed number has the highest MSE and MAE and the lowest R-squared on their respective batch sizes. This can be observed in batch size 64 with seed number 44, batch size 128 with seed number 21, and batch size 256 with seed number 28.

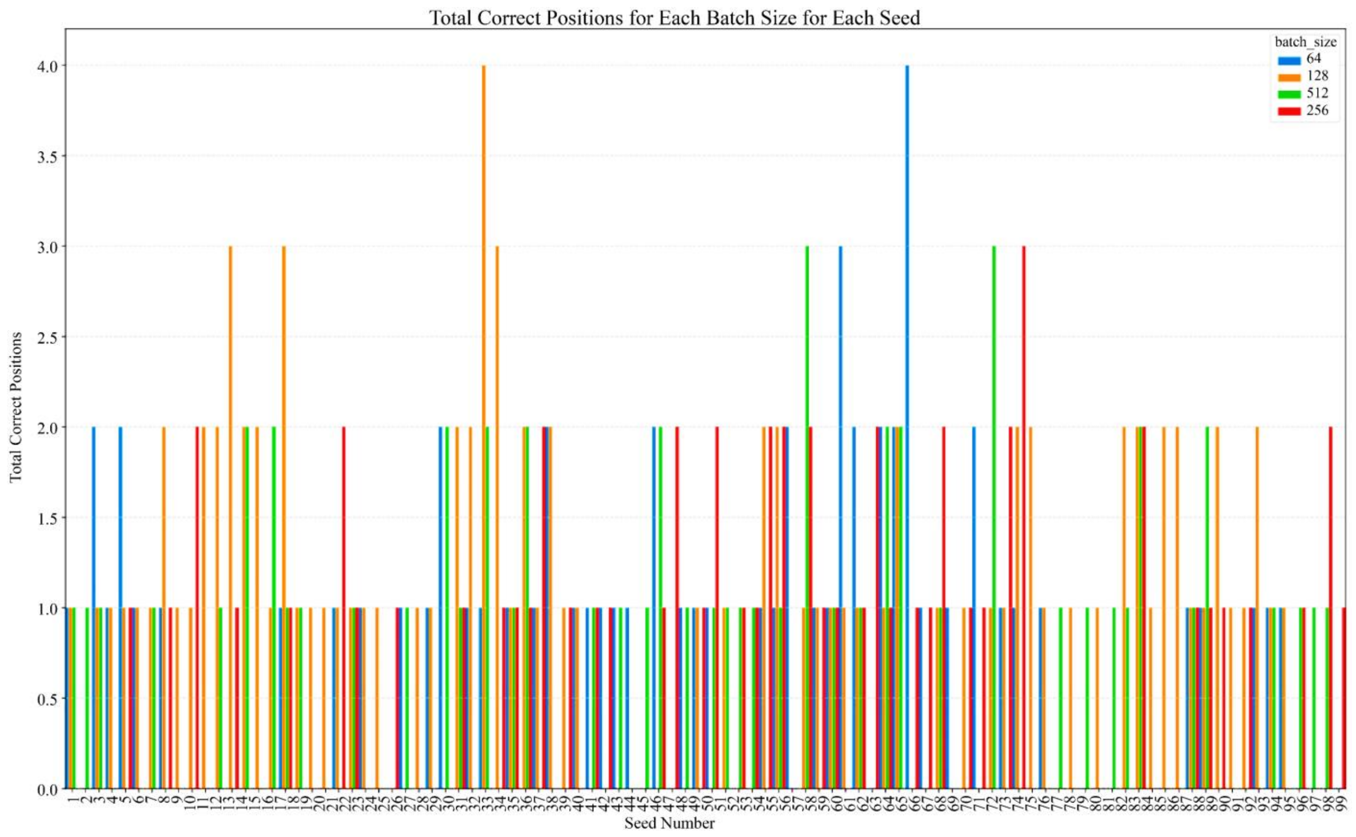


Fig. 4 Total Correct Positions for Each Batch Size for Each Seed

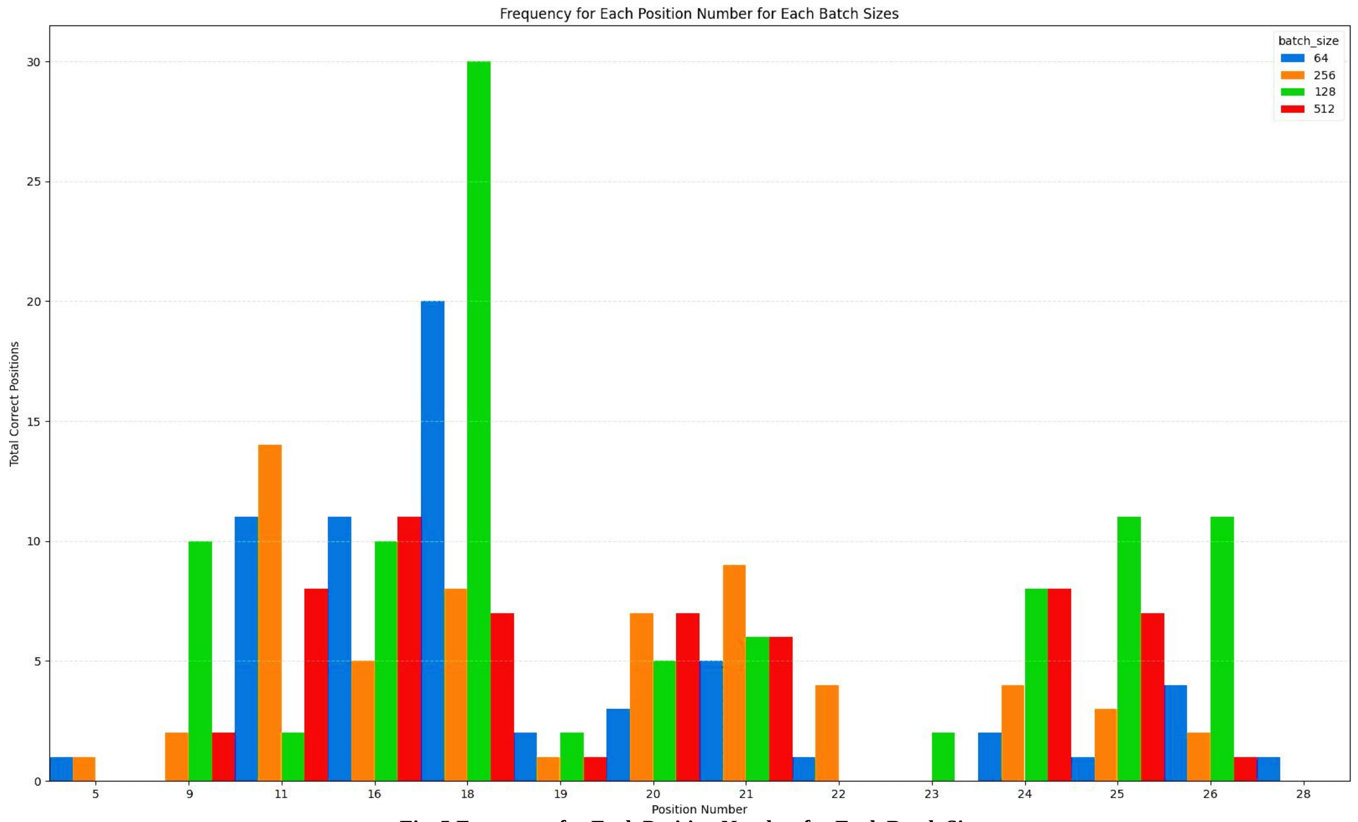


Fig. 5 Frequency for Each Position Number for Each Batch Size

Table 1. Performance metrics

Batch Size	MSE				MAE				R2			
	Min		Max		Min		Max		Min		Max	
	Seed	Value	Seed	Value	Seed	Value	Seed	Value	Seed	Value	Seed	Value
64	91	164.404	44	166.02	91	9.55	44	9.657	44	0.1256	91	0.1340
128	80	153.035	21	161.11	78	9.16	21	9.427	21	0.1514	80	0.1939
256	23	164.352	28	165.69	23	9.56	28	9.649	28	0.1273	23	0.1343
512	13	164.369	99	165.71	47	9.57	28	9.646	99	0.1272	13	0.1342

Table 2. Performance metrics with total positions

Batch Size	Seed	Total Positions	Perf. Metrics
64	44	1	Highest MSE and MAE Lowest R2
	91	-	Lowest MSE and MAE Highest R2
128	21	1	Highest MSE and MAE Lowest R2
	78	-	Lowest MAE
	80	-	Lowest MSE and Highest R2
256	23	1	Lowest MSE and MAE Highest R2
	28	-	Highest MSE and MAE Lowest R2
512	13	1	Lowest MSE and Highest R2
	28	-	Highest MAE
	47	-	Lowest MAE
	99	-	Highest MSE and Lowest R2

Table 3. Best models

Seed Number	Batch Size	MSE	MAE	R2	Total Position
65	64	165.064	9.589	0.131	4
33	128	156.667	9.291	0.175	4

4.3. Discussion

Determining the best model is based on the two methods that have been analyzed. The ideal model can solve several puzzles and has a good performance metric.

To begin, Table 2 provides the seeds from the best performance metrics and subjects them to the puzzles that the model is able to solve. It is evident that despite the good performance metrics, almost all of the seeds cannot solve one out of the 28 puzzles, with a few exceptions being seed number 44 with batch size 64, seed number 21 with batch size 128, seed number 23 with batch size 256, and seed number 13 with batch size 512.

Out of those 4 seed numbers, two of which have the highest MSE and lowest R-squared whilst the other two seed numbers have the exact opposite – lowest MSE and highest R-squared. It is to be expected that the lowest MSE and MAE with the highest R-squared has the highest number of puzzles solved yet the result does not reflect this. Since two models have the same highest number of number of puzzles solved, the performance metric can determine which one of the two models reigns as the champion. Using Table 3, it is evident that the model with seed number 33 with batch size 128 is the best based on the lowest MSE and MAE with the highest R-squared out of the two models with 156.667, 0.291, and 0.175, respectively.

Therefore, it can be concluded that models cannot be evaluated solely on performance metrics. It is important to blend both methods by prioritizing the number of puzzles solved first and then using performance metrics to determine the best model.

5. Conclusion

The primary criterion for selecting the best model was its capability to predict the optimal chess moves. Using an exhaustive comparison of two evaluation methods that compared models based on their number of puzzles solved and using MSE, MAE, and R-squared as the performance metric across a range of 100 different seeds and four different batch sizes revealed a clear frontrunner, the model trained using seed 33 with batch size 128. Notably, this model, when tested against 28 puzzles, managed to successfully solve four out of the 28 puzzles consisting of 4 Kaufman puzzles (position 18, 20, 22, and 24), which trumps the latest advancement that only able to solve two positions out of the 25 Kaufman puzzles (position 3 and 6). Furthermore, the usage of two methods is better to be used in harmony rather than on its own. It is important to prioritize the number of puzzles solved first and then utilize performance metrics as a supplement method. To conclude this paper, some points can be improved for future studies, such as increasing the size of the dataset, exploring other evaluation methods, and exploring other types of games that are similar to chess.

References

[1] David Silver et al., "A General Reinforcement Learning Algorithm that Masters Chess, Shogi and Go through Self-Play," *Science*, vol. 362, no. 6419, pp. 1140-1144, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[2] Omid E. David, Nathan S. Netanyahu, and Lior Wolf, "DeepChess: End-to-End Deep Neural Network for Automatic Learning in Chess," *Artificial Neural Networks and Machine Learning-ICANN, 2016: 25th International Conference on Artificial Neural Networks*, Barcelona, Spain, pp. 88-96, 2016. [CrossRef] [Google Scholar] [Publisher Link]

[3] Matthew Lai, "Giraffe: Using Deep Reinforcement Learning to Play Chess," *Arxiv*, pp. 1-39, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[4] Pieter Bijl, and Anh Phi Tiet, "*Exploring Modern Chess Engine Architecture*," Bachelor Thesis, Vrije Universiteit Amsterdam, pp. 1-30, 2021. [Google Scholar] [Publisher Link]

[5] Board Representation, Chess Programming Wiki, 2020. [Online]. Available: https://www.chessprogramming.org/Board_Representation

[6] Larry Kaufman, "*Rate Your Own Computer*," Computer Chess Reports, vol. 3, no. 1, pp. 1-24, 1992. [Google Scholar] [Publisher Link]

[7] Larry Kaufman, "*Rate Your Own Computer – Part II*," Computer Chess Reports, vol. 3, no. 2, pp. 1-23, 1992. [Publisher Link]

[8] Larry Kaufman, "*Reprint Problems*," Computer Chess Reports, vol. 4, no. 1, pp. 1-31, 1993. [Publisher Link]

[9] Matthia Sabatelli et al., "Learning to Evaluate Chess Positions with Deep Neural Networks and Limited Lookahead," *Proceedings of the 7th International Conference on Pattern Recognition Applications and Methods ICPRAM*, Funchal, Madeira, Portugal, vol. 1, pp. 276-283, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[10] Arman Maesumi, "Playing Chess with Limited Look Ahead," *Arxiv*, pp. 1-11, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[11] Fenil Mehta et al., "Predicting Chess Moves with Multilayer Perceptron and Limited Lookahead," *Journal of Engineering Research and Application*, vol. 10, no. 4, pp. 5-8, 2020. [Google Scholar] [Publisher Link]

[12] Barak Oshri, and Nishith Khandwala, "Predicting Moves in Chess Using Convolutional Neural Networks," *ConvChess*, pp. 1-8, 2016. [Google Scholar] [Publisher Link]

[13] Chao Gao, Ryan Hayward, and Martin Müller, "Move Prediction Using Deep Convolutional Neural Networks in Hex," *IEEE Transactions on Games*, vol. 10, no. 4, pp. 336-343, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[14] Berk Kapicioglu et al., "Chess2vec: Learning Vector Representations for Chess," *Arxiv*, pp. 1-5, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[15] Joseph Lemley et al., "CWU-Chess: An Adaptive Chess Program that Improves After Each Game," *2018 IEEE Games, Entertainment, Media Conference (GEM)*, Galway, Ireland, pp. 1-9, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[16] Avi Schwarzschild et al., "Can you Learn an Algorithm? Generalizing from Easy to Hard Problems with Recurrent Networks," *Advances in Neural Information Processing Systems*, vol. 34, pp. 6695-6706, 2021. [Google Scholar] [Publisher Link]

[17] Rafał Dreżewski, and Grzegorz Wątor, "Chess as Sequential Data in a Chess Match Outcome Prediction Using Deep Learning with Various Chessboard Representations," *Procedia Computer Science*, vol. 192, pp. 1760-1769, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[18] Zheyuan Fan, Yuming Kuang, and Xiaolin Lin, "*Chess Game Result Prediction System*," Stanford University, CS 229 Machine Learning Project Report, pp. 1-5, 2013. [Google Scholar]

[19] Chris J. Maddison et al., "Move Evaluation in Go Using Deep Convolutional Neural Networks," *Arxiv*, pp. 1-8, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[20] Ilya Sutskever, and Vinod Nair, "Mimicking Go Experts with Convolutional Neural Networks," *Artificial Neural Networks-ICANN, 18th International Conference, Prague, Czech Republic, Proceedings, Part II*, pp. 101-11, 2008. [CrossRef] [Google Scholar] [Publisher Link]