# Dataset Related Experimental Investigation of Chess Position Evaluation Using a Deep Neural Network

Dawid Wieczerzak and Paweł Czarnul(✉)

Faculty of Electronics, Telecommunications and Informatics, Gdańsk University of Technology, Narutowicza 11/12, 80-233 Gdańsk, Poland
dawwiecz@pg.edu.pl, pczarnul@eti.pg.edu.pl

**Abstract.** The idea of training Artificial Neural Networks to evaluate chess positions has been widely explored in the last ten years. In this paper we investigated dataset impact on chess position evaluation. We created two datasets with over 1.6 million unique chess positions each. In one of those we also included randomly generated positions resulting from consideration of potentially unpredictable chess moves. Each position was evaluated by the Stockfish engine. Afterwards, we created a multi class evaluation model using Multilayer Perceptron. Solution to the evaluation problem was tested with three different data labeling methods and three different board representations. We show that the accuracy for the model trained for the dataset without randomly generated positions is higher than for the model with such positions, for all data representations and 3, 5 and 11 evaluation classes.

**Keywords:** chess position evaluation · deep neural network · model evaluation · accuracy

## 1 Introduction

Artificial Neural Networks (ANNs) have become models able to predict or estimate otherwise unknown values for various applications, based on prior training using big data sets. Convolutional Neural Networks (CNNs) are widely used for recognition of patterns, that can be of interest in many applications. One of these could be an attempt to evaluate game positions based on their visual representation, for example in chess [15]. There are several interesting factors including the quality of evaluation versus the number of classes we might want to assign to as well as versus training and validation data sets, especially regarding their size and coverage. In the context of chess, several other uses of neural networks were proposed in the literature such as: determination of the optimum number of moves towards winning an endgame assuming optimum play of the other side (for some characteristics of a board position as input) [16]; using a CNN as a piece selector that determines which chess piece should be moved followed by a move selector to determine which move to make [12]; generation of natural-language commentary [4]. It should also be noted that high performance computing systems are important in the chess context as well, either for fast training of models or for running algorithms in parallel [5,10,17].

In this paper, we focused on training a deep neural network for numerical evaluation of chess positions indicating advantage of either the white or black player, based on a set of positions previously evaluated by the Stockfish chess engine[1]. Rather than focusing on tuning a particular neural network model, we adopted a model from the literature [15] and subsequently aimed at performing investigation on how using data sets of various scope (in terms of consideration of moves of various quality) impacts performance of a trained deep neural network model. Additionally, we investigated to what extent the number of classes as well as data representation in the model impacts accuracy of the final model.

## 2 Related Work

In paper [15] authors experimented with training ANNs for the purpose of evaluating chess positions. They used the Fics Games Database to generate around 3000000 chess positions which were evaluated by Stockfish. They created various datasets with various numbers of labels referring to position evaluation such as: 3 for dataset 1, 15 for dataset 2 and 20 for dataset 3 and normalized evaluation of [0,1] for dataset 4. For particular datasets, authors tested tuned Multilayer Perceptrons (MLPs) versus CNNs showing benefits of the former for the task achieving high test accuracies of 96.07%, 93.41% and 68.33% for datasets 1, 2 and 3 respectively for bitmap representation (distinguishes presence of particular chess pieces) which were higher than for an algebraic representation (distinguishes values of particular chess pieces - pawns 1, bishops and knights 3, rooks 5, queens 9 and kings 10) by approx. 2–5%. Finally, the authors mentioned that the trained ANN reached an Elo of approximately 2000 on the chess24.com server. It should be noted that an example of works that focus on obtaining positional values of the chess pieces for particular positions is [20] in which authors used neural networks for that and an evolutionary algorithm for adjustment which resulted in increasing the ranking of their chess engine from ranking 1745 to 2178.

Another paper on playing chess with limited look-ahead is [11]. The author used a large number of board positions – 20 million samples where boards extracted from publicly available databases were extended in such a way that for some positions random legal moves were made and positions evaluated using Stockfish 11. The author tested a classifier for labeling positions with winning for white, black and draw based on evaluation of $cp \leq -150$, $-150 \leq ... \leq 150$ and 150 respectively. A deep neural network with 5 layers, 25% dropout, Adam optimizer and categorical cross entropy, ReLU activation and softmax obtaining approximately 87% testing accuracy.

A different way of chess position assessment and incorporation into a chess playing engine was proposed in [6]. Specifically, they designed and implemented

[1] https://stockfishchess.org/.

a solution that learned to compare chess positions. They used the CCRL dataset (www.computerchess.org.uk/ccrl) with 640000 chess games, out of which white won 221695 games and black won 164387 games – only games that ended with a win were of interest. Firstly, they trained a deep belief network (DBN) called pos2vec that converted a position into a vector. Then, they created a network called DeepChess in which two copies of pos2vec were stacked side by side for position comparison and they trained fully connected layers on top of those for comparison. The authors reported both training and validation accuracies at the very high level of 98%. Subsequently, they conducted play experiments against Falcon for which the evaluation function was 4 times faster than that of the developed solution. Given that DeepChess performed on par, given 4 times more time outperformed Falcon. It also showed 70 more Elo strength than Crafty.

Another work in which the author attempted evaluation of chess positions using a CNN network is presented in [19], versus Stockfish evaluations. The author used the April 2019 https://database.lichess.org database out of which training data was generated from the first 100000 games when white was to move. Finally, 310690 samples were generated with numerical evaluations between $-255$ and $+255$ (boundary values for checkmates), forced checkmate $+/-127$ and normal evaluation capped onto the $[-63,63]$ range. The author used a model with four 2D convolutional layers: the first three with kernel size 3 by 3, the last: 2 by 2. The number of filters were 8, 16, 32 and 64 respectively, with ReLU-activation. 60% of the data set was used for training, 20% for validation and 20% for testing. Final loss and MAE for the test data set were 863.48 and 12.18 respectively. At the same time it was concluded that the model is not able to recognize combinations and tactics and a more complex model shall be tried for improved results. However, whether such can be obtained has to be investigated.

In work [9] the author used a CNN to predict the winning side for positions of a game that ended with a particular result (win for white or black). The model, in the implementation, included layers: Mocha.AsyncHDF5DataLayer, Mocha.ConvolutionLayer, Mocha.PoolingLayer, Mocha.ConvolutionLayer, Mocha.InnerProductLayer, Mocha.DropoutLayer, Mocha.InnerProductLayer, Mocha.BinaryCrossEntropyLossLayer. The data set used for training included games played by opponents with ranking 2000 or higher downloaded from FICS games database[2], finished with checkmates. Data representation used 6 channels corresponding to the boards storing $\{-1, 0, 1\}$ information concerning particular piece types. Finally, obtained validation and test accuracies were 73.5% and 71.8% respectively.

An interesting idea of chess position evaluation was introduced in [12]. Instead of providing numerical board evaluation authors proposed a method for predicting a probability distribution over the grid for pieces to move. For each chess piece they trained a separate model based on a CNN. This approach allowed to predict situations such as escape when a piece is under attack or the king needs to move. The results show that this evaluation method performs significantly better for pieces with local movement (pawn, knight, king). The authors also noticed

[2] https://www.ficsgames.org/download.html.

that a downside of this approach was that highly specific move combinations between nets were not learned. A newer model [13] proposed by other authors based on a similar architecture and board representation showed evaluation as a single numerical value. The output was passed through a mini-max algorithm to determine the best move. A chess engine based on this model showed simple tactics such as sacrificing a piece or forks. In 100 games against the Stockfish engine the system was able to win 3% and draw 2% of games.

In paper [14] authors, motivated by the fact that chess engines can beat even top human players, they assessed the quality of play of many human players, even from various generations, using the Stockfish engine as a quality benchmark. Specifically, score of each move by a human player was compared versus a chess engine move which allowed to compute average error, whether the human player selected first, second etc. engine's preference etc. Out of the world championship (WCC) players, best were Carlsen and Caruana with errors 0.0674 and 0.0709 respectively. Best move percentage winners were Gelfand and Kramnik with 59.9% and 59.2% respectively and average numbers of blunders per WCC were best for Caruana (1.0) and Carlsen (1.3). Work [2] provides selected results of large-scale analysis of chess games with chess engines – authors gathered and analyzed 4.78 million unique games publicly available on some Web repositories. They provided information on Elo distribution, Elo differences between players, plys per game depending on player's Elo differences, percentage of win for white player depending on Elo, first moves depending on game date.

## 3 Data Used for Experiments

We used the Lichess Elite Database [1] that includes a collection of lichess.org games from https://database.lichess.org/ that was filtered to include only games played by players with 2400+ ranking against players with 2200+ ranking, without bullet games.

### 3.1 Data Preparation

Games were downloaded in the PNG format and each position of the games was saved in a database in the FEN format. For selected tests, the original set of positions was also augmented with randomly generated positions in the following way. For each of the positions acquired from the database from Lichess.org 3 moves were generated randomly from all legal moves. This way for each position, 3 potentially unpredictable moves were generated. In some positions, because of checks or specific situations, the number of legal moves was smaller than 3. Such positions were skipped for new position generation. Repeating positions were removed from both the original as well as randomly generated position sets. Afterwards, counts of the two sets were equalized. This way, we acquired two sets with over 1.6 million unique positions each – later marked as no rand and rand respectively. The reason for considering the rand dataset was our aim of additional testing the solution with a presumably more diverse data set including board positions potentially reached by weaker players.

Afterwards Stockfish was used to label all positions from previously mentioned sets. Labels generated by Stockfish contained board evaluations expressed as centipawns (*cp*). Value of 100 *cp* corresponds to a difference of one pawn and this metric shows a current difference in strategic and material strength between players. As an example, when the evaluation is +100 *cp* it means that the moving side has a potential advantage of one pawn. Evaluations of all positions were stored from the point of view of white regardless of the player to move. We also scaled the evaluations by dividing them by 100 and thus obtaining what we call a value in scaled centipawns (*scp*). For our evaluations we used Stockfish 13 with a depth of 28.

## 3.2 Board Representation

Positions processed in the previous step were converted into a vector representation making it usable as an input for neural networks. We used a bitboard representation which turns each position in FEN format into binary vector with total length of 768 bits. This method of transforming chess position into a vector was used in some previous works [6,11,15]. Another similar bitboard approach has been used in many chess position analyses based on CNNs [12,15,19].

We also introduced modifications into the bitboard representation which gives us two additional representations. In total we tested 3 board representations: bitboard representation, algebraic representation, piece value representation.

A bitboard vector consists of 12 chessboards linked with one another that form a 64-bit position vector. Each of the boards, which are considered as a feature, stores position of a given piece (type). The first 6 features represent positions of the white player pieces while the other 6 of their opponent – black. Pieces are represented in the following order: pawns, knights, bishops, rooks, queen, king. A piece position inside each 64 bit vector is represented as 1 when it belongs to the player who should move or −1 when it belongs to the opposite player. The total length of the vector is 768 because it stacks 64 bit features for 12 different pieces.

The algebraic representation is an extension of the binary one. We introduced this modification in order to see what the effect of differentiating chess pieces on position evaluation will be. Beside presence of particular pieces it also considers different pieces by assigning them following integer numbers starting from 1. In this method pawns are represented as 1, knights as 2, bishops as 3, rooks as 4, queens as 5 and kings as 6. Similarly to the previously described method, opposite player's side is represented by negative numbers.

In the last representation piece strength and its potential value were taken into account. We used a common assignment of point values which is 1 for pawns, 3 for knights and bishops, 5 for rooks and 9 for queens. Because of its non exchangeable nature the king is not considered in most evaluation systems. In this piece value representation we decided to assign 10 points to the king.

It stemmed from the important strategic role of the piece, also considered in [20]. Point values replaced binary presence of each piece and negative values to distinguish moves of opposite sides have also been used. We shall note that this piece strength representation corresponds to the one called algebraic in [15], as described in Sect. 2.

### 3.3 Data Labeling

For classification of positions using games with previously added Stockfish evaluations we followed the approach from [15] experimenting with different numbers of classes and data representations. We created three labeling methods for the classification task.

*Method 1*: In this method each of the positions was assigned to one of three classes: Winning, Losing or Draw. Labels were assigned according to scaled centipawn evaluations with the following conditions: positions were considered as Winning when its *scp* evaluation was greater than 1.5, losing when its *scp* was lower than −1.5 and draw when *scp* was between those two values.

*Method 2*: This method extends *Method 1* by dividing both the Winning and Losing classes into two separate classes for a total of 5 different labels including Draw. The division was done in such a way that the first Winning label contains positions with *scp* between 1.5 and 4.5 and the second Winning label contains positions with *scp* greater than 4.5. The same has been done for labels in the Losing class where division point was set to −4.5 *scp*. Labeling conditions for Draw class remained the same as in *Method 1*.

*Method 3*: In this method even more labels were created. All classes including Draw have been extended by creating new labels as follows: In Winning class, with 2 starting from 1.5 *scp*, four new labels were created so that the last labeling window contains positions with evaluations greater than 7.5 *scp*. Labels in the Losing class were assigned in the same way. If scp decreases by 2 starting from −1.5 *scp*, a new label was created. Draw class, which originally contained positions with its *scp* between −1.5 and 1.5, was divided into equal intervals each of them 1 *scp* wide. This method creates 11 different labels in total: four labels in Winning class, four labels in Loosing class and 3 Draw labels.

## 4 Test Methods

In this section we describe data analysis methods concerning the data described in Sect. 3. We present an ANN architecture used for testing different inputs in detail, subsequently we discuss the experiments and the training process of the model.

### 4.1 Neural Network Architecture

In order to address classification tasks we created a 3 hidden layer MPL based on the architecture and hyper parameters proposed in [15]. Similarly to the originally proposed classifier, hidden layers consisted of 1048, 500 and 50 hidden units. Each of 3 hidden layers has been activated by the Rectified Linear Unit (ReLU) activation function. Due to targeting classification tasks the final output layer has been connected to Softmax activation. Furthermore, in order to achieve better model generalization Batch Normalization and Dropout regularization were applied to all hidden layers of the network. We set the probability of Dropout to 0.2 as recommended in [18].

### 4.2 Experiment

As a result of our experiments we wanted to assess the impact of different chess board representations on classification performance. In order to do that, we divided the experiment into three steps corresponding to different board representations proposed in the previous section. For each board representation three different classification tasks were tested. We used three previously mentioned labeling methods: *Method 1*, *Method 2* and *Method 3* respectively. As input data firstly we used the dataset without random positions and then the dataset extended with randomly generated positions.

This test configuration gives us 2x3 separate network training cases in each experimental step.

### 4.3 Training Method

We have split each relevant data set into training, validation and test sets in proportion of 8:1:1. In each training the *Adam* algorithm was used as an optimizer and it was initialized with the following parameters: $lr = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.99$, $\epsilon = 1e-8$. We trained the networks with *minibatches* of 128 samples and *categorical cross entropy* as a loss function. The whole training was stopped after the validation loss has not improved by at least 0.00001 within the last 100 epochs. For each epoch we measured the following metrics: accuracy, precision, recall, f1 [8].

In all experiments we used Tensorflow and Python 3.8 as a programming base running on computers with Intel i7-7700 CPU, 32GiB RAM and GeForce GTX 1070.

## 5 Results

In Figs. 1, 2 and 3 we summarize results of training after the stop condition has been met for each given configuration i.e. 3, 5 and 11 classes respectively. There are six configurations in total i.e. for the algebraic, bitboard and piece strength data representations, each for the no rand and rand data set. For each configuration we present validation accuracy and f1 metrics.

In order to see the progress of training, as an example for the 5 class configuration and the best bitboard representation, in Fig. 4 we show how validation accuracy and f1 scores change over 135 epochs.

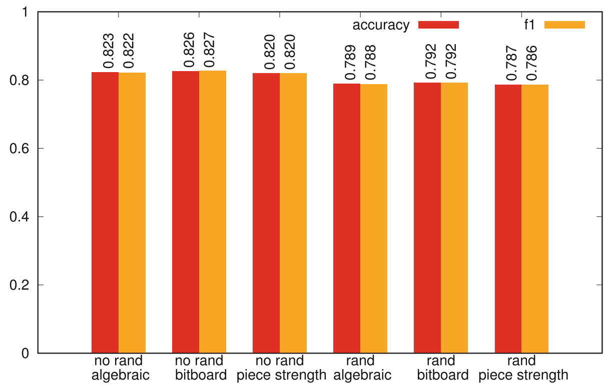

**Fig. 1.** validation accuracy and f1 metrics, 3 classes

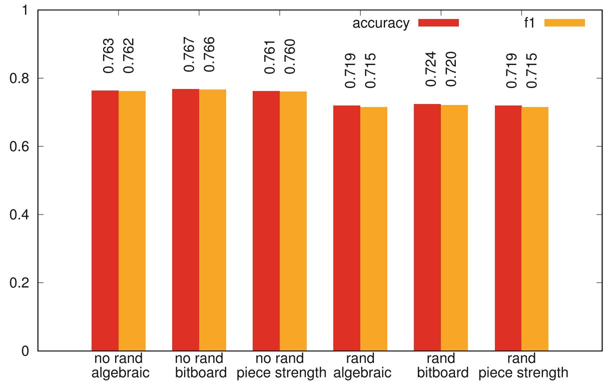

**Fig. 2.** validation accuracy and f1 metrics, 5 classes

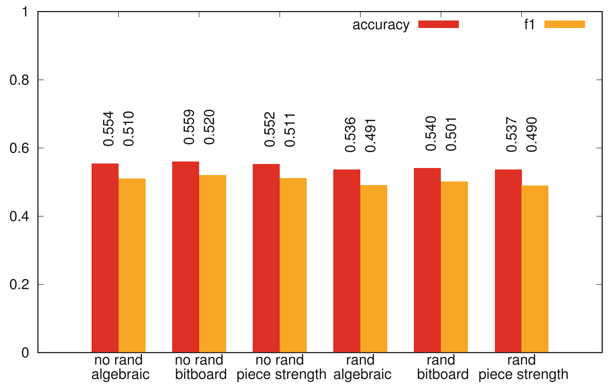

**Fig. 3.** validation accuracy and f1 metrics, 11 classes

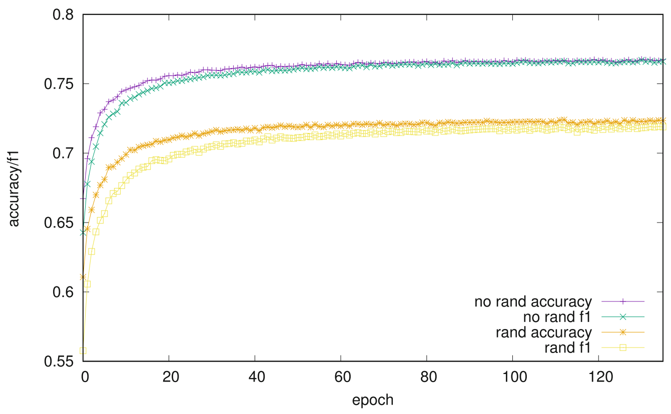

**Fig. 4.** validation accuracy and f1 metrics, 5 classes, bitboard representation

Following these tests, for the best representation (bitboard) in Table 1 we provide final precision and recall values for the three cases (3, 5 and 11 classes) for the two data sets. Finally, in Table 2 we included accuracy values computed for the test dataset and the bitboard representation.

**Table 1.** Precision and recall values for various configurations, bitboard representation, validation

| Number of classes | Data without random pos. | | Data with random pos. | |
|---|---|---|---|---|
| | precision | recall | precision | recall |
| 3 | 0.83205 | 0.82112 | 0.80039 | 0.78325 |
| 5 | 0.79523 | 0.73922 | 0.76249 | 0.68148 |
| 11 | 0.69183 | 0.41733 | 0.69250 | 0.39315 |

**Table 2.** Test accuracy for various configurations, bitboard representation

| Number of classes | Data without random pos. | Data with random pos. |
|---|---|---|
| 3 | 0.82528 | 0.79158 |
| 5 | 0.76727 | 0.72316 |
| 11 | 0.55795 | 0.53964 |

## 6 Discussion

Based on the presented results, we can conclude the following:

1. For all tests with 3, 5 and 11 classes, configurations with data without random positions yield slightly but visibly better accuracy and f1 score values than corresponding configurations trained using data with added random positions.
2. For almost each configuration and either data without or with random positions, the order of data representations from best to worst accuracies is generally: bitboard, algebraic and piece strength with bitboard being the best one for all the cases. For 5 and 11 classes piece strength and algebraic configurations resulted in virtually same results, with minimal differences.
3. As expected, we see a visible drop in accuracies as the number of classes is increased.
4. Taking into account results for the best tested representation bitboard we see that with an increasing number of classes, differences between precision and recall values for particular data sets are increasing and are visibly larger for the data set with random positions. Additionally, while for 11 classes precision values for the two data sets are very close, there is a visible difference of approximately 0.024 for recall values.

We shall note that, based on our experiment and particular data sets, the accuracy/f1/precision/recall scores for the rand data set, presumably a more diverse one, turned out to be worse than for the no rand data set. On the other hand, while we did not focus on ultimate improvement of the model per se and rather focused on comparison per various data sets in these experiments, we shall note that accuracy values obtained in [15] for 3 classes were higher. Further investigation could be performed on if and how this could be related to the different training data sets and/or Stockfish settings used for evaluation, in both cases, etc. In [15] for the other data sets (for larger numbers of classes) a different MLP structure was used.

## 7 Summary and Future Work

In the paper we investigated accuracy, precision/recall and f1 metrics for training an artificial model for evaluation of chess positions – for two data sets: one – with games by 2400+ players playing against 2200+ ranking players and another – the same one augmented with randomly generated positions by making random moves from already known positions. We tested three different data representations such as bitboard, algebraic and with consideration of piece strength values – results showed that there were measurable albeit very small differences with best results for the bitboard version. We investigated assignment of numerical evaluations into 3, 5 and 11 classes. We found out that the dataset with randomly generated positions (that intuitively corresponds to positions that could also be reached by weaker players) resulted in test accuracy scores smaller than that of the data set with positions obtained by stronger players. This suggests that in this particular case it is more difficult to obtain high accuracies for a data set with presumably more diverse positions. On the other hand, based on that, in the future, it would be an interesting research task to investigate whether it can be generalized and how using even more restricted data sets affects network performance metrics. This might refer to certain phases of the game played by very good players, e.g. endgames, with possibly even selected sets of pieces on the board. Another interesting topic would be training the models with consideration of a training data set extended with similar positions [7] to those originally in the dataset. Furthermore, a test on whether the observations from this paper would also be applicable to more fine-tuned models would be of interest.

## References

1. Lichess elite database. https://database.nikonoel.fr/
2. Acher, M., Esnault, F.: Large-scale analysis of chess games with chess engines: A preliminary report. CoRR abs/1607.04186 (2016). http://arxiv.org/abs/1607.04186
3. Baldi, P., Sadowski, P.J.: Understanding dropout. In: Advances in Neural Information Processing Systems, vol. 26 (2013)
4. Butner, C.: Chesscoach is a neural network-based chess engine capable of natural-language commentary (2021). https://pythonrepo.com/repo/chrisbutner-ChessCoach-python-natural-language-processing
5. Czarnul, P.: Benchmarking parallel chess search in Stockfish on intel Xeon and intel Xeon phi processors. In: Shi, Y., et al. (eds.) ICCS 2018. LNCS, vol. 10862, pp. 457–464. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-93713-7_40
6. David, O.E., Netanyahu, N.S., Wolf, L.: DeepChess: end-to-end deep neural network for automatic learning in chess. In: Villa, A.E.P., Masulli, P., Pons Rivero, A.J. (eds.) ICANN 2016. LNCS, vol. 9887, pp. 88–96. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-44781-0_11
7. Ganguly, D., Leveling, J., Jones, G.J.: Retrieval of similar chess positions. In: Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval, SIGIR 2014, pp. 687–696. Association for Computing Machinery, New York, NY, USA (2014). https://doi.org/10.1145/2600428.2609605
8. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge (2016). http://www.deeplearningbook.org
9. Int8: Chess position evaluation with convolutional neural network in Julia (2016). https://int8.io/chess-position-evaluation-with-convolutional-neural-networks-in-julia/
10. Jouppi, N.P., et al.: A domain-specific supercomputer for training deep neural networks. Commun. ACM **63**(7), 67–78 (2020). https://doi.org/10.1145/3360307
11. Maesumi, A.: Playing chess with limited look ahead. CoRR abs/2007.02130 (2020). https://arxiv.org/abs/2007.02130
12. Oshri, B., Khandwala, N.: Predicting moves in chess using convolutional neural networks (2016)
13. Panchal, H., Mishra, S., Shrivastava, V.: Chess moves prediction using deep learning neural networks. In: 2021 International Conference on Advances in Computing and Communications (ICACC), pp. 1–6. IEEE (2021)
14. Romero, O., Cuenca, J.F., Parra, L., Lloret, J.: Computer analysis of world chess championship players. In: ICSEA: The Fourteenth International Conference on Software Engineering Advances, pp. 200–205 (2019). ISBN: 978-1-61208-752-8

15. Sabatelli., M., Bidoia., F., Codreanu., V., Wiering., M.: Learning to evaluate chess positions with deep neural networks and limited lookahead. In: Proceedings of the 7th International Conference on Pattern Recognition Applications and Methods - ICPRAM, pp. 276–283. INSTICC, SciTePress (2018). https://doi.org/10.5220/0006535502760283
16. Samadi, M., Azimifar, Z., Jahromi, M.Z.: Learning: an effective approach in endgame chess board evaluation. In: Sixth International Conference on Machine Learning and Applications (ICMLA 2007), pp. 464–469 (2007). https://doi.org/10.1109/ICMLA.2007.48
17. Silver, D., et al.: A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. Science **362**(6419), 1140–1144 (2018). https://doi.org/10.1126/science.aar6404
18. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. **15**(1), 1929–1958 (2014)
19. Vikstrom, J.: Training a convolutional neural network to evaluate chess positions. KTH Royal Institute of Technology, School of Electrical Engineering and Computer Science, Stockholm, Sweden (2019)
20. Vázquez-Fernández, E., Coello Coello, C.A., Sagols Troncoso, F.D.: Assessing the positional values of chess pieces by tuning neural networks' weights with an evolutionary algorithm. In: World Automation Congress 2012, pp. 1–6 (2012)