2023 International Conference for Advancement in Technology (ICONAT)
Goa, India. Jan 24-26, 2023

CHESS AI: Machine learning and Minimax based Chess Engine

Jyoti Madake, Chinmay Deotale, Geetai Charde, Shripad Bhatlawande
Department of Electronics and Telecommunication Engineering
Vishwakarma Institute of Technology
Pune, India
{jyoti.madake, chinmay.deotale19, geetai.charge19, shripad.bhatlawande}@vit.edu

2023 International Conference for Advancement in Technology (ICONAT) | 978-1-6654-7517-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICONAT57137.2023.10080746

***Abstract* - Designing Chess Engine has been a main focus of research for a long time. The paper employs a novel combination approach of Machine learning based estimator with artificial intelligence (AI) to build chess AI. The Minimax Algorithm is a decision theory-based technique implemented for reducing the load on the chess engine's hardware. Also, Alpha-Beta Pruning algorithm is implemented to eliminate any nodes in the search tree that aren't essential and hence makes the AI efficient. Trained estimators achieved a high accuracy of 96.77% for calculating the probability of 'good move'. A variable depth of search tree based on the number of legal moves has also been employed for the minimax algorithm.**

Keywords - Chess engine, Machine learning, Minimax, Alpha-Beta pruning.

I. INTRODUCTION

Chess has been played for almost 1500 years. The game's earliest predecessor was started in India by name "Chaturanga" - a Sanskrit word for four divisions (of the military)": infantry, cavalry, elephantry, and chariot. The objective of the game is to capture your opponent's 'King' piece. When either a king is defeated or the stalemate condition is met, the game ends. the FIDE- World Chess Federation set the rules for the game [1].

Building a powerful Chess Engine to play against computers has been proposed with the advancement in Machine learning and Artificial Intelligence field. The paper proposes a Hybrid approach to build a Chess engine by integrating Machine learning based estimator with Minimax and Alpha-Beta pruning.

II. LITERATURE REVIEW

Many techniques have been offered to Build Chess AI and Chess Engine. Based on current Chess board positions, a DEEPCHESS model can predict the winner using a Deep Neural network has been proposed [2]. The proposed method uses deep unsupervised and supervised neural networks for pre-training and to select preferable positions from two input positions respectively. A Convolutional Neural Network (CNN) model trained using Board Set Datasets was used in a similar way [3]. Validation against the stock fish chess engine provided an accuracy of 39.16% accuracy. Another method using automatic feature extraction and pattern recognition where the possibility of using probability thresholds instead of depth to shape search trees has also been explored [4].Unsupervised module that groups comparable chess games together using a similarity distance function to reduce the amount of sample games to learn or study has been proposed. Method also employs a Supervised Module which uses an associative classifier to anticipate and discover common chess board configurations and/or winning moves, according to the proposal. [5]. A neural network built on a pre-trained VGG network and a multilayer perceptron network is used to identify synthetically generated chess images. Hough Lines Detection and the K-Means Clustering Technique have been used in the proposed system [6].

The influence of Elo ratings on chess game outcomes is investigated, as well as whether this metric may be used to forecast future performance in matches [7]. Perceptual chunks are used to predict optimal areas to move using methodologies based on chess cognitive neuroscience, and again it was cross-validated against game records including upwards of five million positions [8]. Deep learning is a more explored approach to build a model for Chess Engines. The Loss function improves the accuracy of evaluation functions represented by Deep Neural Networks (DNN). To test the performance of the trained evaluating functions, 1-1 accuracy, Top-1 accuracy and self-plays were used [9]. Jonathan Baxar et al., 2000 used the concept of game-tree search using Minimax and evaluation function for checking whether the move is good or not [10].

In addition to chess, several other board games like as gomoku, block go, and soji models use a strategy quite similar to that described here. An strategy based on reinforcement learning has been investigated in a variety of board games, including checkers [11], backgammon [12], and others. On the RenjuNet dataset, an approach was used to train Deep Neural Networks (DNN) using supervised learning in order to forecast the movements that Gomoku players will make. This was done in order to improve accuracy. The completed neural network has an accuracy of approximately 42% when it comes to motion prediction [13]. A variation of Go known as Block Go. They begin by explaining the complexities of Block Go, a game that falls in between checkers and othello. After then, a Deep Convolutional Neural Network (D-CNN) was used to analyse Block Go [14]. It was decided to extract move sequences from the Shogi player's game record that occurred quite frequently. When determining the next move, it was compared to the Shogi program's game record, and was found that it overestimated the opponent's capabilities [15].

III. METHODOLOGY

As discussed in the Literature review section, building any on board games mostly involved strategies like Reinforcement

learning, Ml model trained on Human data or AI based algorithms like Minimax. The paper discusses a hybrid approach of Ml model trained on Human data and Artificial Intelligence (AI) based algorithms like Minimax along with Alpha-Beta pruning for building chess engines.

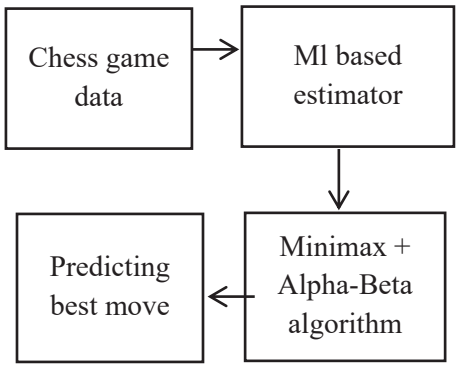

Fig. 1. Block Diagram for system

The strategy is implemented in two stages as shown in fig.1. Initially, the engine will reject 50% of the potential moves given on a board using the machine learning model. It accomplishes this by calculating the probability of a move to be a 'good move'. Based on this, all of the moves are ordered and only the top 50th percentile is considered for further computation. On the remaining movements, the engine will apply the minimax algorithm. The search tree will go to a certain depth, depending on the complexity of the board and return the best move possible using the evaluation function.

A. *Data Acquisition and labeling*

The dataset was acquired from [1], it contains thousands of games in the portable game notation (.pgn) format played by chess grandmasters. The features were further extracted from the dataset and labeled as good move or bad move. If the result of the current match in consideration was won by the player, then that particular move is labeled as a good move. In case the match resulted in draw or loses, the move is labeled as a bad move. Due to dynamics in Chess games most of the game results in a 'Draw' and hence we could find a disproportionate distribution of Good Moves and Bad Moves in fig.2.

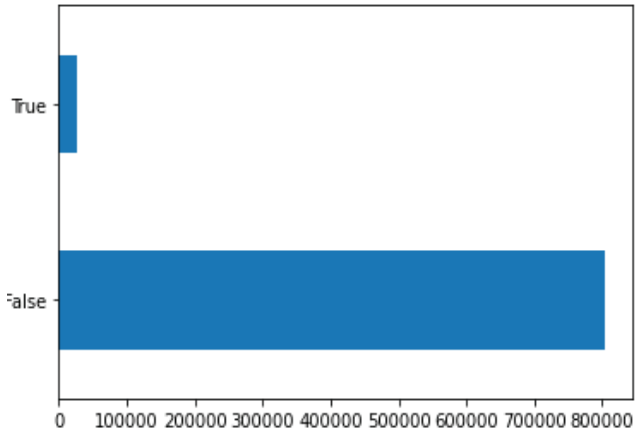

Fig. 2. Distribution of Labeled Data

B. *Data Description*

Labeled Data from CSV forms a dimension of (8 million, 193).

TABLE I. DESCRIPTION OF THE DATA COLUMNS

Column Number	Column Description
1- 64	Current Board
65 - 128	Starting point of the Move
129 - 192	End point of the move
193	Target Variable/ Label

As shown in Table 1, the first 64 columns are corresponding to the current board's 64 squares (8*8 boards). The starting position of a move is represented by columns from 65 to 128. The respective column contains 1.0 if a piece was moved from that square, otherwise the column value is 0.0. The ending position of a move is represented by columns from 129 to 192: The respective column contains 1.0 if a piece was moved from that square; otherwise the column value is 0.0. The last column (193) is the column for the target variable.

C. *Model Training*

Linear estimator was used to train over the chess game data. Model uses equation of –

$$Y = W * X + B \quad (1)$$

Where, weight (W), Bias (B), inputs (X) and the outputs (Y). If we have n features the general equation is like this.

$$Y = B + W_1X_1 + W_2X_2 + \cdots W_nX_n \quad (2)$$

Where W1, W2, Wn are weights for X1, X2, Xn inputs.

It uses sum of squared errors as the loss function and Gradient Descent as optimizer. To provide data for training, evaluation and prediction, the Input function was defined for the model. It returns a two element tuple. The first element is a feature dictionary, with the key containing the feature's name and the value as an array holding all of the feature's values. The second element is an array that contains the label values for each example. Model was trained using a batch number of 9 and epoch size of 10. The trained model was used to get the probability of the possible legal moves for it to be a good move.

D. *Minimax & Alpha-Beta Algorithm*

Minimax algorithm is based on multiplayer zero sum game theory. The Minimax method goes up to a certain depth in the recursive tree of legal steps and evaluates the leaves (nodes) using an evaluation function. Depending on the turn (maximizing or minimizing), it can then return the value of the largest or smallest child to the parent node.

To the given depth, the recursive tree of all possible moves is explored, and the position is evaluated at the tree's "leaves". After that, depending on whether it's a white or black to move

it returns either the smallest or the largest child value to the parent node. (i.e we try to either maximize or minimize the outcome at each alternate level respectively).

Alpha-beta pruning, a search technique was applied to reduce the number of nodes in its search tree that are examined by the minimax algorithm. When at least one possibility has been found, it stops evaluating a move. It proves that the move is worse than the move examined previously. Such moves do not need to be further evaluated. Standard minimax tree also returns the same move as the minimax method. However, it prunes away branches that cannot possibly influence the final decision [Russell, Stuart J. *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010.].

Algorithm 1: Minimax and Alpha-Beta pruning
Input: board_position, depth, maximizing_player=1,alpha , beta
Output: Best possible move

Initialization:
1. Minimax (Input)
2. If (d==0) or board_position lead game over:
3. **return** evaluation (board_position)
4. If maximizing_player (x): //(max player)
5. best_move = -∞ (-9999)
6. **For each child of board_position:**
7. best_move = max of (best_move, minimax (d-1,Bp,a,b,!x))
8. a = max of (a, best_move)
9. **If (beta (b) <= alpha (a)):**
10. **return** best_move
11. else: //(min player)
12. best_move = +∞ (+9999)
13. **For each child of board_position:**
14. best_move = min of (best_move, minimax (d-1,Bp,a,b,!x))
15. b = min of (b, best_move)
16. **If (beta (b) <= alpha (a)):**
17. **return** best_move
18. return best_move

E. Evaluation function

The Chess pieces are evaluated on the basis of two primary conditions. 1. Type of piece and 2. Position of that piece. The value of position based on the type of piece is shown in Table 3.

TABLE II. VALUATION

Chess Piece	White	Black
Pawn	10	-10
Bishop	30	-30
Knight	30	-30
Rook	50	-50
Queen	90	-90
King	900	-900

Second factor is like an evaluation factor which considers the location of the pieces. For example, a knight in the centre of the board is preferable than a knight on the board's edge (since it has more possibilities and hence is more active).

```
([[-5.0, -4.0, -3.0, -3.0, -3.0, -3.0, -4.0, -5.0],
  [-4.0, -2.0, 0.0, 0.0, 0.0, 0.0, -2.0, -4.0],
  [-3.0, 0.0, 1.0, 1.5, 1.5, 1.0, 0.0, -3.0],
  [-3.0, 0.5, 1.5, 2.0, 2.0, 1.5, 0.5, -3.0],
  [-3.0, 0.0, 1.5, 2.0, 2.0, 1.5, 0.0, -3.0],
  [-3.0, 0.5, 1.0, 1.5, 1.5, 1.0, 0.5, -3.0],
  [-4.0, -2.0, 0.0, 0.5, 0.5, 0.0, -2.0, -4.0],
  [-5.0, -4.0, -3.0, -3.0, -3.0, -3.0, -4.0, -5.0]], np.float)
```

Knight

The pinnacle of a pawn is right before the last square from the opponent's side (i.e., white's pawn is on the 7th row). They will receive 0 points since they will be promoted to a different piece type right away.

```
([[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
  [5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0],
  [1.0, 1.0, 2.0, 3.0, 3.0, 2.0, 1.0, 1.0],
  [0.5, 0.5, 1.0, 2.5, 2.5, 1.0, 0.5, 0.5],
  [0.0, 0.0, 0.0, 2.0, 2.0, 0.0, 0.0, 0.0],
  [0.5, -0.5, -1.0, 0.0, 0.0, -1.0, -0.5, 0.5],
  [0.5, 1.0, 1.0, -2.0, -2.0, 1.0, 1.0, 0.5],
  [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]], np.float)
```

Pawn

Despite the fact that both the rook and the bishop have a large range of moves, the proportion of their evaluation is vastly different. In the case of the rook, they nonetheless pose a significant danger to opponents at the different end of the board. Unlike the rook, the bishop has a number of drawbacks while playing on the board's edge, whether on the friendly or opponent's side. Because bishops can only move one way when in the corner, it is considered equivalent to a knight.

```
([[-2.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -2.0],
  [-1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -1.0],
  [-1.0, 0.0, 0.5, 1.0, 1.0, 0.5, 0.0, -1.0],
  [-1.0, 0.5, 0.5, 1.0, 1.0, 0.5, 0.5, -1.0],
  [-1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, -1.0],
  [-1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, -1.0],
  [-1.0, 0.5, 0.0, 0.0, 0.0, 0.0, 0.5, -1.0],
  [-2.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -2.0]], np.float)
```

Bishop

```
([[0.0, 0.0, 0.0, 0.0, 0.0,  0.0, 0.0, 0.0],
  [0.5, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.5],
  [-0.5, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.5],
  [-0.5, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.5],
  [-0.5, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.5],
  [-0.5, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.5],
  [-0.5, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.5],
  [ 0.0, 0.0, 0.0, 0.5, 0.5, 0.0, 0.0, 0.0]], np.float)
```

Rook

When placed on any square, queen pieces have no noticeable difference in value. However, unlike the rook, the queen does not have any squares that would place them at a disadvantage, presuming no piece is blocking their path.

```
[-2.0, -1.0, -1.0, -0.5, -0.5, -1.0, -1.0, -2.0],
[-1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -1.0],
[-1.0, 0.0, 0.5, 0.5, 0.5, 0.5, 0.0, -1.0],
[-0.5, 0.0, 0.5, 0.5, 0.5, 0.5, 0.0, -0.5],
[0.0, 0.0, 0.5, 0.5, 0.5, 0.5, 0.0, -0.5],
[-1.0, 0.5, 0.5, 0.5, 0.5, 0.5, 0.0, -1.0],
[-1.0, 0.0, 0.5, 0.0, 0.0, 0.0, 0.0, -1.0],
[-2.0, -1.0, -1.0, -0.5, -0.5, -1.0, -1.0, -2.0]], np.float)
```

Queen

```
[[-3.0, -4.0, -4.0, -5.0, -5.0, -4.0, -4.0, -3.0],
 [-3.0, -4.0, -4.0, -5.0, -5.0, -4.0, -4.0, -3.0],
 [-3.0, -4.0, -4.0, -5.0, -5.0, -4.0, -4.0, -3.0],
 [-3.0, -4.0, -4.0, -5.0, -5.0, -4.0, -4.0, -3.0],
 [-2.0, -3.0, -3.0, -4.0, -4.0, -3.0, -3.0, -2.0],
 [-1.0, -2.0, -2.0, -2.0, -2.0, -2.0, -2.0, -1.0],
 [2.0, 2.0, 0.0, 0.0, 0.0, 0.0, 2.0, 2.0],
 [2.0, 3.0, 1.0, 0.0, 0.0, 1.0, 3.0, 2.0]], np.float)
```

King

When positioned in the first two rows of the game, the king position has a higher value when it is safe.

Using this Evaluation matrix Chess pieces evaluation is defined using following equation (1)-

Chess piece's evaluation= Chess piece's value + Valuation at [x][y] (1)

F. Game Handling

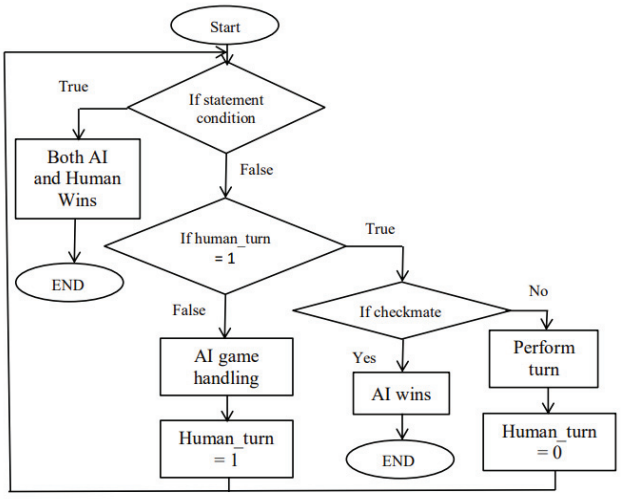

Fig. 3. Game handling

As shown in fig 3. AI and player play alternate turns starting with checking the stalemate condition. If the current board positioning leads to stalemate the game is considered as "Draw". If not, AI or player concludes his turn depending upon whose turn it actually was. In case of a Human turn, first checkmate condition is checked and if not, the checkmated human pushes his turn in the current board. Turn is later shifted to AI. If a human is checkmated, the game concludes as "AI wins". In case of AI, the condition of checkmate is checked followed by AI game Handling shown in fig. 4. Turn is later shifted to AI. In case AI is a checkmate, Human is considered as Winner for the game.

In AI's turn, a list of all the legal moves is compiled followed by checking of moves which will checkmate the player. If such a move is not available in the list, this list is passed through a pre-trained model to predict the probability of individual moves for it to be a good move. The output from the model is further sorted according to the probability and the top 50 percentile of moves are given to the Minimax and Alpha-Beta Pruning algorithm explained in Algorithm 1.

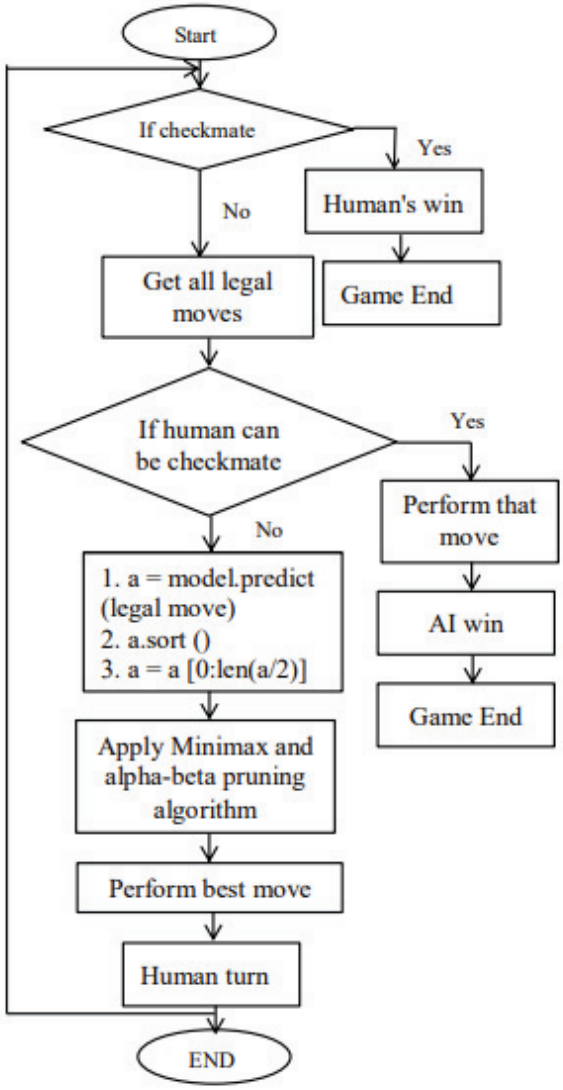

Fig. 4. AI Handling

IV. RESULTS

The dataset was split into two parts: training and testing. When evaluated on a testing dataset, the trained estimator model provided a significantly high accuracy of 96.77%. The system required 2 hours 17 minutes for training when tested on a laptop with processor specification of intel core i5 8th Generation with CPU clock speed of 1.8 GHz and 8 GB RAM.

The distribution of probability predicted by the model on testing data is shown in fig.5 below.

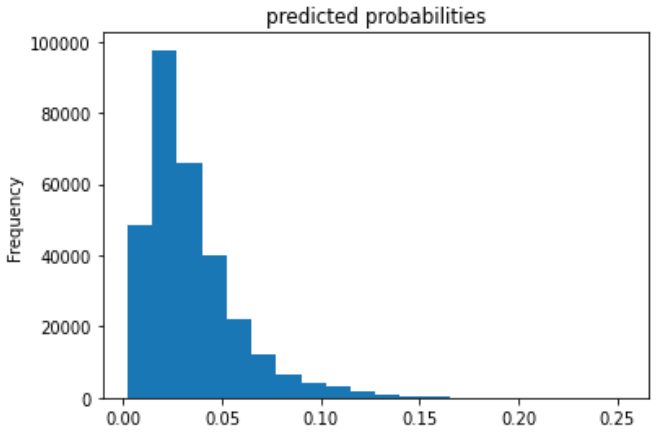

Fig. 5. distribution of predicted probability

The chess game engine was tested with multiple values of depth for Minimax & Alpha-Beta pruning algorithm and different percentile of moves considered. This process was tested on starting moves where only 20 moves are possible and further evaluated using Execution time and which starting move was performed.

TABLE III. EVALUATION TABLE BASED ON DIFFERENT DEPTHS AND PERCENTILE OF MOVE CONSIDERED

d	n	Top x %ile of move considered	Traversing and execution time(in seconds)	Move performed
1	20	1.0	2.085386	b1-c3
1	20	0.75	2.0377166	b1-c3
1	20	0.50	2.0067090	b1-c3
3	20	1.0	5.8768	b1-c3
3	20	0.75	5.2999	e2-e4
3	20	0.50	5.0838	e2-e4
5	20	1.0	38.1191	e2-e4
5	20	0.50	29.5312	e2-e4
5	20	0.75	32.9986	b1-c3
7	20	0.50	99.4254	e2-e4
7	20	0.75	102.3329	e2-e4
7	20	1	117.99565	e2-e4

**Note- d: depth for minimax tree traversing, n: number of possible moves.*

Table 4 shows that as the depth of the Minimax Tree increased, the CPU required a larger amount of time for the computation of all leaf nodes. Also, the increase in execution time was exponential with respect to the increase in the depth. In the case of percentile of moves considered, three cases of 100, 75 and 50 percentile were taken. Although there wasn't a significant decrease in execution time even when the algorithm was applied on a lesser number of moves, there were different moves played by the AI in a few cases.

Execution time is heavily dependent on the number of legal moves and the depth of the Search tree in the minimax algorithm, a novel method of variable size of depth and variable percentile of moves was utilized. If the number of legal moves in the list was greater than 30 minimax with a depth of 3 and 50 percentile moves were taken (which mostly occurs in the start and mid-game scenarios). Whereas in case the legal moves were in between 10 to 30 minimax with a depth of 5 and 75 percentile moves was used (which mostly occurs in the latter half of the game) In the rest of the cases depth was chosen to 7 and all the possible legal moves were considered by the AI (which mostly occurs in end game scenarios).

V. CONCLUSION AND FUTURE SCOPE

The paper proposes a novel approach integrating Machine learning with AI based algorithms like Minimax and Alpha-Beta pruning for developing Chess game engine. Approach also utilizes concepts of variable depth and variables considered legal moves to make AI faster and efficient. The trained estimator provided a high accuracy of 96.77% when evaluated on testing data. The paper also discusses the Game Handling involved in Chess. The developed AI Analyses the current board and predicts the best move possible for the respective state.

REFERENCES

[1] https://www.fide.com/fide/about-fide
[2] David, Omid E., Nathan S. Netanyahu, and Lior Wolf. "Deepchess: End-to-end deep neural network for automatic learning in chess." In International Conference on Artificial Neural Networks, pp. 88-96. Springer, Cham, 2016.
[3] Panchal, Hitanshu, Siddhant Mishra, and Varsha Shrivastava. "Chess Moves Prediction using Deep Learning Neural Networks." In 2021 International Conference on Advances in Computing and Communications (ICACC), pp. 1-6. IEEE, 2021.
[4] Lai, M.: Giraffe: Using deep reinforcement learning to play chess. Master's Thesis, Imperial College London (2015)
[5] Brown, James A., Alfredo Cuzzocrea, Michael Kresta, Korbin DL Kristjanson, Carson K. Leung, and Timothy W. Tebinka. "A machine learning tool for supporting advanced knowledge discovery from chess game data." In 2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA), pp. 649-654. IEEE, 2017.
[6] Neto, Afonso de Sá Delgado, and Rafael Mendes Campello. "Chess position identification using pieces classification based on synthetic images generation and deep neural network fine-tuning." In 2019 21st Symposium on Virtual and Augmented Reality (SVR), pp. 152-160. IEEE, 2019
[7] Thabtah, Fadi, Arun J. Padmavathy, and Andrew Pritchard. "Chess Results Analysis Using Elo Measure with Machine Learning." Journal of Information & Knowledge Management 19, no. 02 (2020): 2050006.
[8] T. Bossomaier, J. Traish, F. Gobet and P. C. R. Lane, "Neuro-cognitive model of move location in the game of Go," The 2012 International Joint Conference on Neural Networks (IJCNN), 2012, pp. 1-7, doi: 10.1109/IJCNN.2012.6252377.
[9] H. Zhu and T. Kaneko, "Comparison of Loss Functions for Training of Deep Neural Networks in Shogi," 2018 Conference on Technologies and Applications of Artificial Intelligence (TAAI), 2018, pp. 18-23, doi: 10.1109/TAAI.2018.00014.
[10] Baxter, J., Tridgell, A., Weaver, L.: Learning to play chess using temporal differences. Mach. Learn. 40(3), 243–263 (2000)
[11] Tesauro, G.: Practical issues in temporal difference learning. Mach. Learn. 8(3–4), 257–277 (1992)

[12] Schaeffer, J., Hlynka, M., Jussila, V.: Temporal difference learning applied to a high-performance game-playing program. In: Joint Conference on Artificial Intelligence (2001)
[13] K. Shao, D. Zhao, Z. Tang and Y. Zhu, "Move prediction in Gomoku using deep learning," 2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC), 2016, pp. 292-297, doi: 10.1109/YAC.2016.7804906
[14] S. Yen, C. Lin, G. Cheng and J. Chen, "Deep learning and block Go," 2017 International Joint Conference on Neural Networks (IJCNN), 2017, pp. 2698-2701, doi: 10.1109/IJCNN.2017.7966187.
[15] T. Kinebuchi and T. Ito, "Shogi Program That Selects Natural Moves by Considering the Flow of Preceding Moves," 2015 3rd International Conference on Applied Computing and Information Technology/2nd International Conference on Computational Science and Intelligence, 2015, pp. 79-84, doi: 10.1109/ACIT-CSI.2015.22.