Silesian
University
of Technology

MASTER THESIS

Testing the level of chess game effectiveness depending on the type of used neural network

Michał DYLA
Student identification number: ⟨274132⟩

Programme: Control, Electronic, and Information Engineering
Specialisation: Data Science

SUPERVISOR
⟨dr inż. Tomasz Grzejszczak⟩
DEPARTMENT ⟨Department of Automatic Control and Robotics⟩
Faculty of Automatic Control, Electronics and Computer Science

CONSULTANT
⟨mgr Eryka Probierz⟩

Gliwice 2022

Thesis title

Testing the level of chess game effectiveness depending on the type of used neural network

Abstract

(Thesis abstract – to be copied into an appropriate field during an electronic submission – in English.)

Key words

(2-5 keywords, separated by commas)

Tytuł pracy

Thesis title in Polish

Streszczenie

(Streszczenie pracy – odpowiednie pole w systemie APD powinno zawierać kopię tego streszczenia.)

Słowa kluczowe

(2-5 slow (fraz) kluczowych, oddzielonych przecinkami)

Contents

Chapter 1

Introduction

Chapter 2

Chess playing AI

Before further discussion about thesis topic, it is require to analyze main components and issues related to it. To make those speculations easier to understand and internalize, they will be divided into two main sections: chess game logic and application logic. Both of those topics will be described in each distinct sections, starting with chess game logic.

2.1 Chess

Chess is a board game in which two players compete against each others by using 16 chess pieces [12]. Each game is started by white site player an after that both players performs their actions sequentially, one by one. There are three ways to end the game. First option is to left enemy king piece without any moves. This manoeuver is called **checkmate**. One more thing that needs to be achieved to perform checkmate is to put enemy king in the **check** situation (threatened with capture by enemy piece) [12, 35].

Second method to end the game is to wait until time will end. Standard time limit used on a lot of major chess tournaments is 90 min. That means that both players have 90 minutes to finish game. When time will end and player still performing his move, he loses the game. Usage of the time limit force players to thing and act fast but still according to their game plan.

Last option to end chess game is to force enemy player to resign. In that case, situation is simple and player who resigned, loses the game.

As it was mentioned before, every player need to use their 16 chess pieces and adapt the strategy to make opponent lose. In the tab. 2.1 are shown all chess pieces types with their unique move patterns and special properties. Quick remark: in the table, pieces are not differentiate by its color because either white and black piece work the same way. Mentioned chess pieces are deployed on the chess board of dimensions 8×8 (as it is shown on fig. 2.1), then the game starts by white site player move [12, 3, 29].

Castling is an manoeuver which includes king piece and on of the rooks. It consist in

Table 2.1: The list of chess pieces.

symbol	**name**	**moving pattern**	**special properties**
♔ ♚	king	rectilinear or diagonal movement, only by one square	castling
♕ ♛	queen	rectilinear or diagonal movement, over any number of squares	none
♗ ♝	bishop	diagonal movement, over any number of squares	none
♘ ♞	knight	rectilinear movement, by one square, then diagonal movement in the same direction, by one square	jumping over other pieces
♖ ♜	rook	rectilinear movement, over any number of squares	castling
♙ ♟	pawn	move forward, by one square or diagonal movement by one square while capturing	promotion, *en passant*, in case of first move can move by one or 2 squares forward

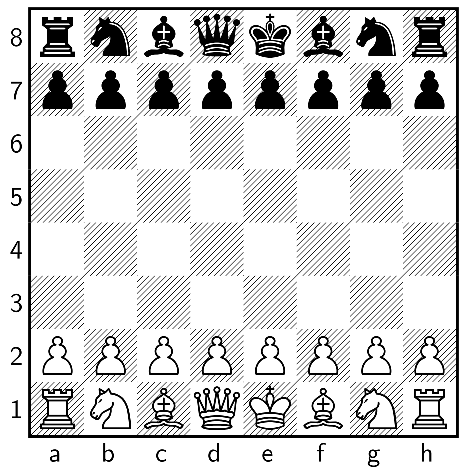

Figure 2.1: Chessboard layout at the beginning of the game.

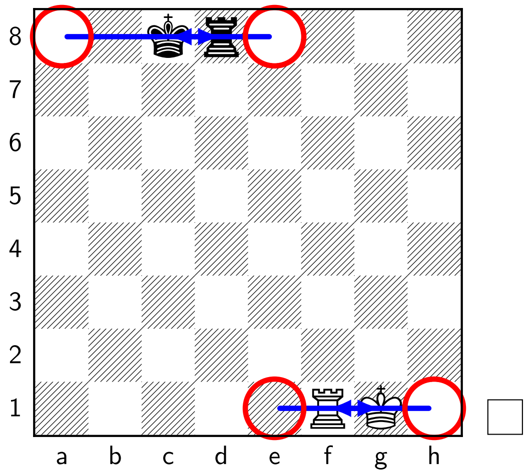

Figure 2.2: Castling manoeuver (short castling on white site, long castling on black site).

moving king horizontally, by two squares, towards the participating rook and then placing rook on square which was passed by king. Requirements to perform castling manoeuver:

- both pieces needs to be in the same color,
- castling needs to be first move performed by both pieces in this game,
- squares between both pieces needs to be blank and not attacked by enemy pieces,
- king cannot be under check and performing castling manoeuver cannot result in this situation.

There are two types of castling (short castling and long castling) which are presented on fig. 2.2. In the past there were third type of castling which has been performing by rook created by promotion manoeuver. Unfortunately, this manoeuver has been outlawed in 1972.

Jumping over other pieces is an manoeuver which can be performed only by knight piece. It consist in moving knight piece on the destination square even if path is blocked by other piece. Jumping manoeuver is presented on fig. 2.3.

Promotion in an specific manoeuver which can be performed only by pawn piece. It happens when one of pawns reach enemy site of the chessboard. In that situation player chooses any piece of the same color (except king) on which he want to replace pawn which performed promotion. This manoeuver allows for situation in which, for example there will be more than 1 queen in the same game. Promotion manoeuver is presented on fig. 2.4.

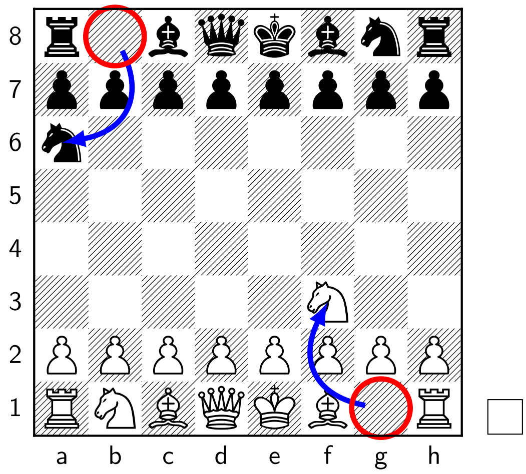

Figure 2.3: Jumping manoeuver.

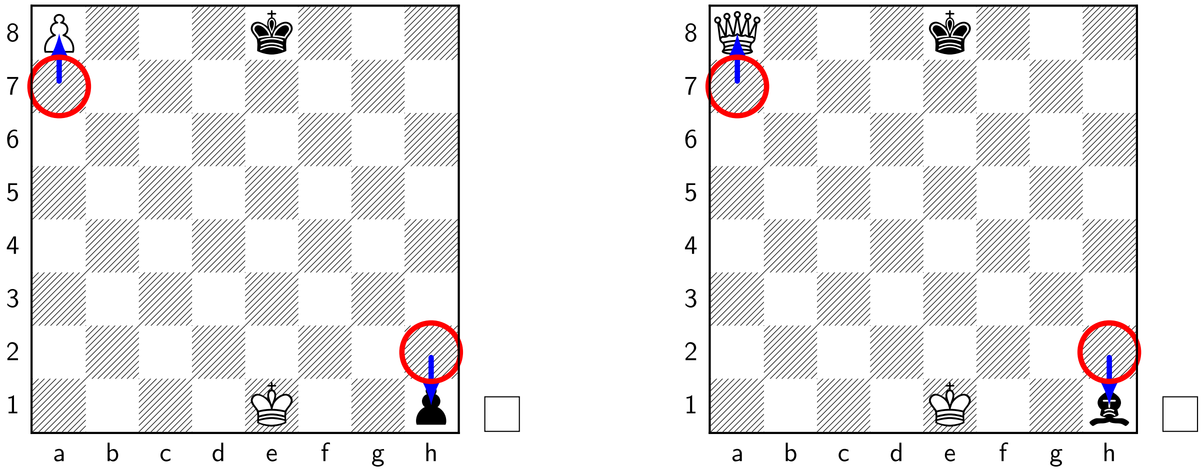

Figure 2.4: Promotion manoeuver (white pawn from square a7 promoted to queen, black pawn from square h2 promoted to bishop).

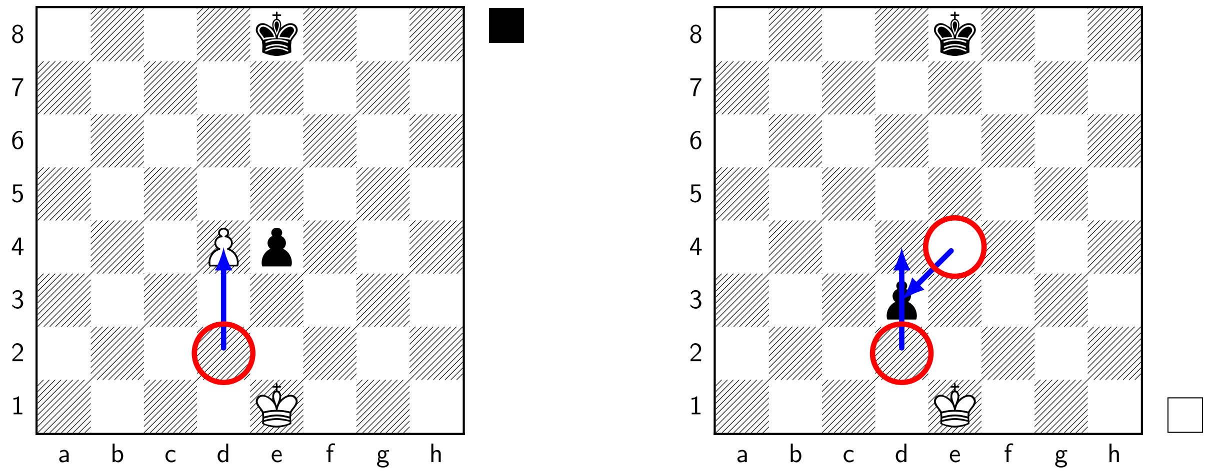

Figure 2.5: *En Passant* manoeuver.

En Passant is an special variant of capturing, assigned to pawn piece. This capturing can be performed if enemy pawn made move by two squares and crossed square that is attacked by performing pawn. In that situation, capturing pawn is moved on attacking square and enemy pawn gets removed from chessboard. One last requirements to perform *en passant* is to perform it directly after enemy pawn move. *En Passant* manoeuver is presented on fig. 2.5.

That is all of the chess game rules. At the beginning, chess may seem like very easy game but it happens to be very difficult problem ro be resolved by machine. Even thou, it is hard to „teach" machine to play chess, there are a lot of chess engines which realize this functionality [19]. The most known chess playing softwares are: „AlphaZero" created by Google company [10, 28], „Stockfish" created by Marco Costalba, Tord Romstad and Joona Kiiski [28, 30] and „Leela Chess Zero (Lc0)" created by Gary Linscott and Alexander Lyashuk.

2.2 Game trees

After describing problem that needs to be solved, it is necessary to describe solution to the problem. There is no analytic solution for the chess game so for solving this problem comprehensive approach is mostly used. Core element in all, previously mentioned chess playing softwares, is a game tree. Game tree is a graph data structure which consists of all possible moves that players can make. It is safe to say that usage of game tree is the most efficient algorithm which allows machine to make decisions. A lot of chess playing softwares uses this methodology with great success [10, 30, 28].

Game tree consists of two main components:

Node is an representation of situation on the chessboard. Each node is created on proper tree level which reflect particular player turn.

Branch represent each move that player can make in particular situation.

Game tree structure have one last property which is **game complexity**. This property is an number of nodes in last layer of complete game tree [19].

To simplify further description of this issue, instead of using chess as an example, easier game called „Hexapawn" will be used. It is a board game based on chess but its rules are much more simple. Each player have 3 pawns (functioning in the same way like pawns in chess), on the opposite sites of 3×3 board. Player can win by one of 3 ways:

- reach enemy site of the board with one of the pawns,
- capture all of enemy pawns,
- leave opponent with no moves.

Hexapawn was created by american mathematician Martin Gardner in March 1962 [14]. Hexapawn has been created to demonstrate first AI machine. This game fits perfectly for this usage because of its relatively small game tree. For the purpure of this thesis, Hexapawn has been simplified to just 2 pawns for each player and board of dimensions 2×3. Rest of the rules, mentioned before, are unchanged.

2.2.1 Game tree building process

Due to the relatively low degree of complexity, previously mentioned game will be used as an example in game tree building process. As in all tree based graphs, building process starts from **root node** which is a first node that will be basis of the entire structure. After generating root node, next level of nodes is generated and attached to root node. Process of generating entire tree level base on actual situation on the board. In that case number of nodes in the layer depends on number of moves that player can perform in given situation. After first level of the game tree has been generated, the same process is applied to further levels. It is important to remember that players performs their moves alternately, which mean next generated tree level will be generated with second player point of view. Completely generated game tree for used version of Hexapawn is presented on fig. 2.6. Last thing worth mentioning is the the fact that each game tree node consists not only from representation of chessboard, but also from evaluation value which has been assigned to this situation. However, this is a topic related directly to the functional aspect of the game tree which will be further discussed in section 2.2.2.

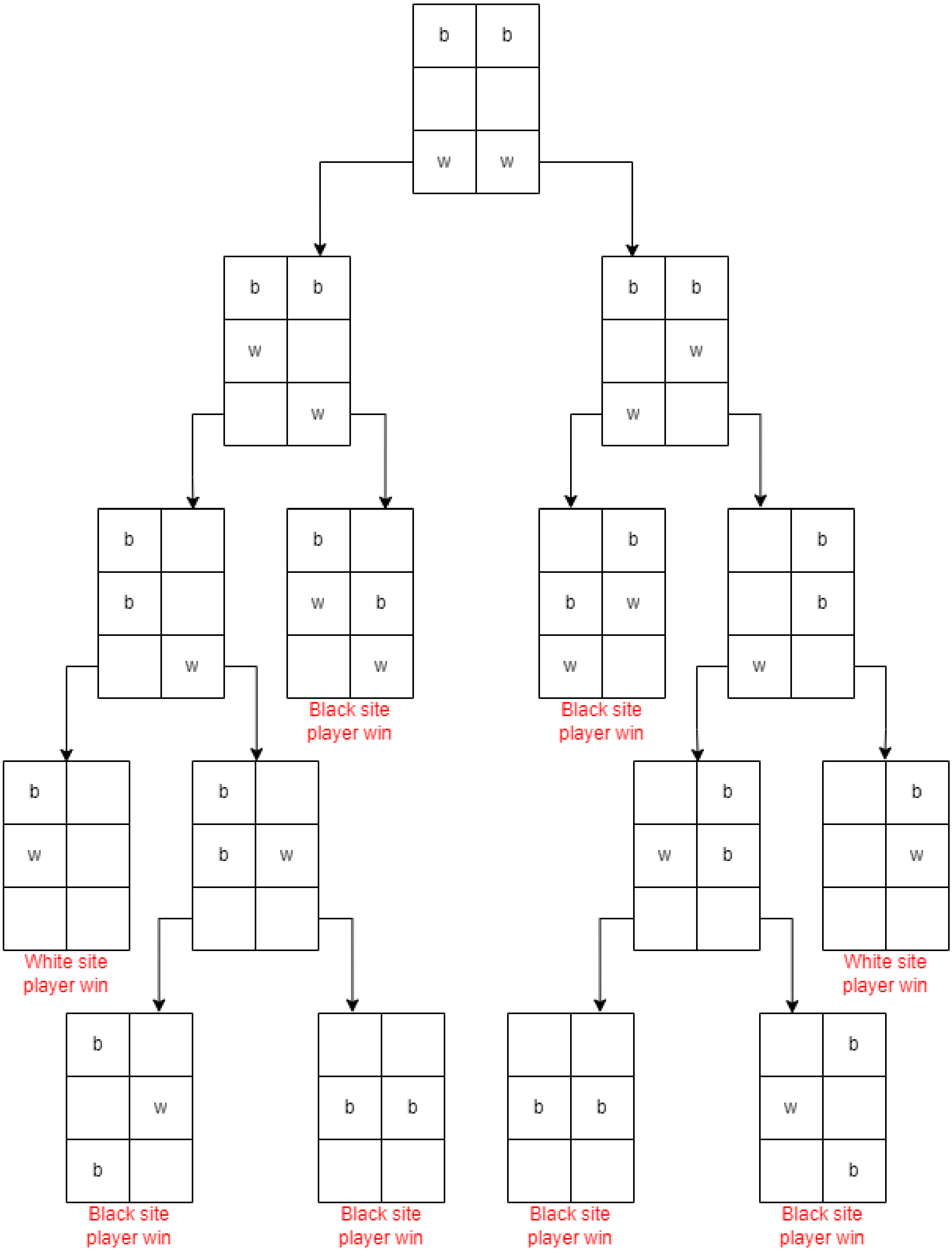

Figure 2.6: Complete game tree for simplified Hexapawn (w - white pawns, b - black pawns).

2.2.2 Min-max algorithm

Even if game trees are crucial components while constructing AI, this component won't allow created instance for making decisions. To make those kind of operations **min-max tree** can be used. This structure i basically a game tree but every tree node consists also from evaluation value. Second difference between game tree and min-max tree is the fact that in second structure uses search algorithm called **min-max algorithm**. Before describing how min-max algorithm works it is important to explain how to evaluate each tree node. To calculate evaluation value for each nodes in the game tree evaluation functions are used. This type of function take as an input content of the tree node and return evaluation that needs to be assign to this node. More about evaluation functions in scope of this thesis can be found in section 2.2.3.

To explain how min-max algorithm works, generated game tree from fig. 2.6 will be used. Because generated tree present whole game (it is possible to see all outcomes), very simple evaluation function has been used. If given node resulted in white site player win, evaluation will be equal to 1. Otherwise, evaluation will be equal to -1. The calculated values were assigned to proper nodes as it is presented on fig. 2.7.

As it can be seen on fig. 2.7, it present also usage of min-max algorithm which will be now described.

Structure analysis begins from is leafs. In the first iteration two nodes containing values -1 and -1 are compared. Because this layer represent opponent move, node with the lowest value got chosen (marked with red color). It happens because by definition, the opponent will make optimal moves, leading to a favorable situation for him. In the given situation, when both values are equal, first encountered value get chosen. After node comparison, chosen value is moved to node above for further comparisons. The same actions has been performed for rest nodes in this layer. After all nodes in last layer get compared, layer above needs to be analyze next. In case of this layer, comparing process is similar to te previous layer. Because analyzing layer represent AI move, among comparing nodes, the one with the highest value gets chosen (from both first nodes, value 1 gets chosen and marked with green color). Similar like in last layer all chosen values are moved to layer above for further comparisons. By using this workflow, all root nodes evaluations needs to be calculated. Situation presented on fig. 2.7 shows that both root nodes have the same evaluation value so first encountered value get chosen [6].

2.2.3 Game tree optimizations

Simplify version of game Hexapawn is simple enough it is possible to generate full game tree (resolve the game). Unfortunately, game that is a topic of this thesis is much more complex which means, resolving chess game is unreachable. Basing on average branching factor for chess game $b \approx 35$, and average game length $m \approx 70$, Victor Allis estimate

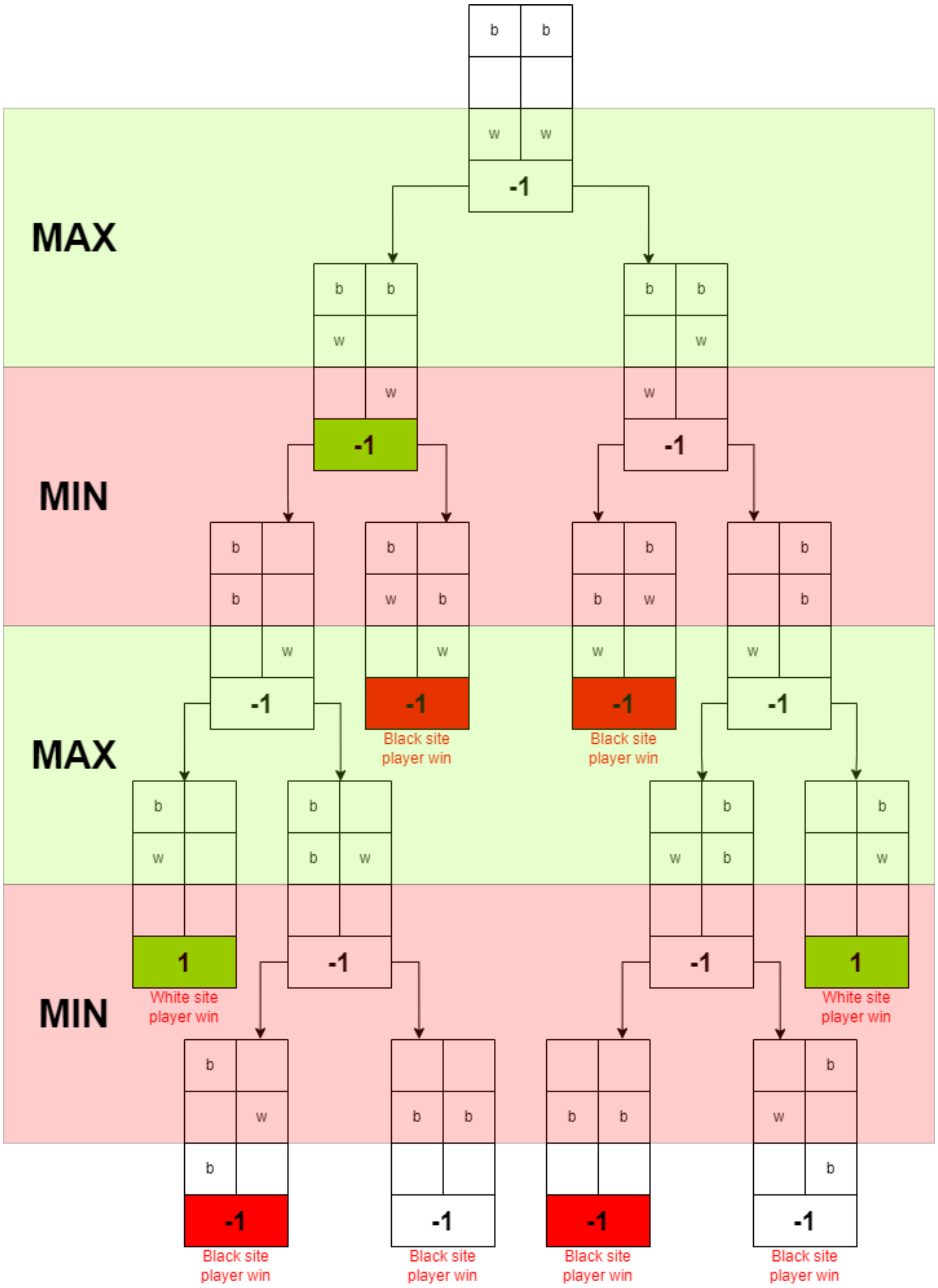

Figure 2.7: Min-max tree for simplified Hexapawn.

complexity of the average game of chess to be 10^{123} [1]. That big complexity is potentially problematic because of big memorial and computational complexity of entire structure. Because of this complexity, a common practice is optimizing game tree size [34].

The most commonly used optimization method is limiting depth of generating structure. In a lot of chess playing softwares, algorithm that generate game tree, do it until some constant value (most commonly this value is 3) [10, 1, 34]. This solution determine how many moves can be handled by created structure but prevents from solving the game. Usage of this optimization method force to use evaluation method because by analyzing only fragment of the game tree it is impossible to know all outcomes. Evaluation functions can be based on heuristic equations [11] or can be more complex. In this project more complex evaluation method has been used. Evaluation will be handled by built and learned neural network instance which will be described further more in section 2.3. There are much more optimization method that can be used, like α-β pruning, but in scope of this thesis, only game tree depth limitation has been used.

2.3 Neural networks

As it was mentioned before, as an evaluation function for generated game tree, neural network will be used. While analyzing the problem it turned out that there are two types of neural network which can work with good effectiveness. As a scope of this thesis is to test which one of those two types of neural network is more suitable for given problem of chess game. Two neural networks that will be described are Artificial Neural Network (ANN) and Convolutional Neural Network (CNN).

2.3.1 Artificial neural network

To simplify further descriptions, it is better to start with ANN topic as it is the simplest version of any neural networks. Neural network is and mathematical structure model which uses basic processing elements (neurons) to perform some kind of operation on input data. An inspiration for this model is real-life neural systems. There are a lot of different types of neural networks but all of them consists of 2 main components: neuron and weight [24]. To avoid misunderstanding, in the further part of this thesis, both „artificial neural network" and „neural network" will relate to the same type of neural network. Artificial neural network in its most basic form can consists of following elements:

Weight represent the connection between neurons. Each weight have also value assign to it which represent how strong particular connection is. Weights values are first of parameters that are modified in learning process. Learning process for ANN will be further described in this section (2.3.3).

Neuron is the most basic element of neural network. It is element performing mathematical operation on input data and as an output it return just single value. Output value of neuron k in layer m ($n_k^{(m)}$) can be calculated using following equation:

$$n_k^{(m)} = f\left(b_i + \sum_{i=0}^{l} w_{m-1,i} n_i^{(m-1)}\right), \tag{2.1}$$

where:

$n_i^{(m-1)}$ – output value of neuron i in layer $(m-1)$, $w_{m-1,i}$ – weight value from neuron i in layer $(m-1)$, b_i – value of bias i, l – number of neurons in layer $(m-1)$, f – activation function.

As it can be seen in equation (2.1) was used activation function. In this thesis, for artificial neural network, sigmoid function has been used ($a = \frac{1}{1+e^{-n}}$). This activation function squash input value in range $(0, 1)$ but another, often used range o this function is $(-1, 1)$ [23]. In case of this thesis, second range has been used.

Layer works as and organization unit for neurons which define in which order, those neurons will be processed. Usually, artificial neural network consists of 3 types of layer [26]:

- Input layer – represent group of neurons containing input data. Neurons in this layer do not have weights and their value are not passed through activation function.
- Hidden layer – represent group of neurons which are located between first and last layer of the network. Number of hidden layers and number of their neurons are not strictly defined and often it needs to be experimented with, to find the best setup for given problem.
- Output layer – represent group of neurons located at the end of the network which is also network answer for given problem.

Bias is a additional neuron in each layer which is used for output regulations. This neuron also have weight value, that is why this weight value can be skipped in output calculations. Value of the bias is added to final sum of neuron values and their weights as it has been shown in equation 2.1. Value of the bias is the second modifiable parameters, changed in learning process.

For better understanding of neural network structure, on fig. 2.8 has been shown simple example of this kind of network. As it can be seen, this network consists of 3 layers (1 input layer, 1 hidden layer and 1 output layer) and value from input layer gets propagated through all other layers. Process shown of fig. 2.8 is called **feed forward algorithm** and

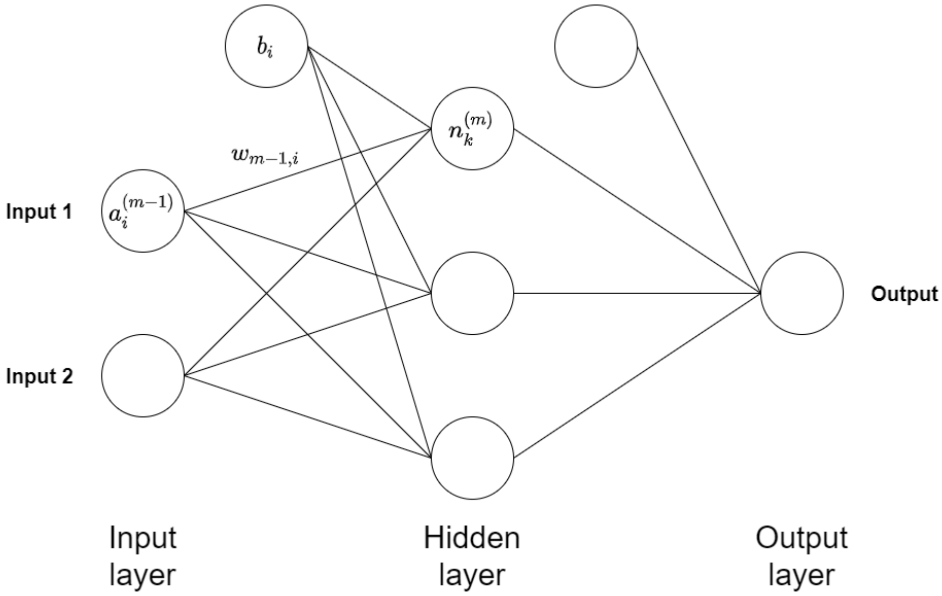

Figure 2.8: Artificial neural network example.

constitute whole process of resolving problems in neural networks. Feed forward process begins from loading input data into input layer. Then, by using equation presented in 2.1, those values are propagated through all hidden layers (if their exist) and lastly, final answer can be seen in output layer [26, 33, 24]. It was decided to use this type of neural network, for chess game problem, because it is the most basic type of neural network so it will be very good comparison point for other used models.

2.3.2 Convolutional neural network

As it was mentioned before, scope of this thesis is to compare two neural network models and decide which one performs better in case of chess game problem. Second type of neural network that will be used for this task is Convolutional Neural Network. Why this type of network? Convolutional neural networks (CNN) act as pattern detecting algorithm. The most common usages of this model are: cancer detection, picture classifications, face detection, recognizing hand-written digits etc. [21]. Because of this fantastic performance while detecting patterns, this type of neural network can also be able to detect patterns on chessboard which will result in good evaluation of chessboard situations. To see how good this model will perform with this problem, it will be compared with the most basic neural network described in section 2.3.1.

Now, when structure of artificial neural network has been described, it is possible to explain CNN structure. Convolutional neural network can consists of 4 types of layer:

Fully-connected/Dense layer is the same type of layer that exist in artificial neural network. This layer has been described in 2.3.1. In case of CNN, dense layer are used as an input an output layers.

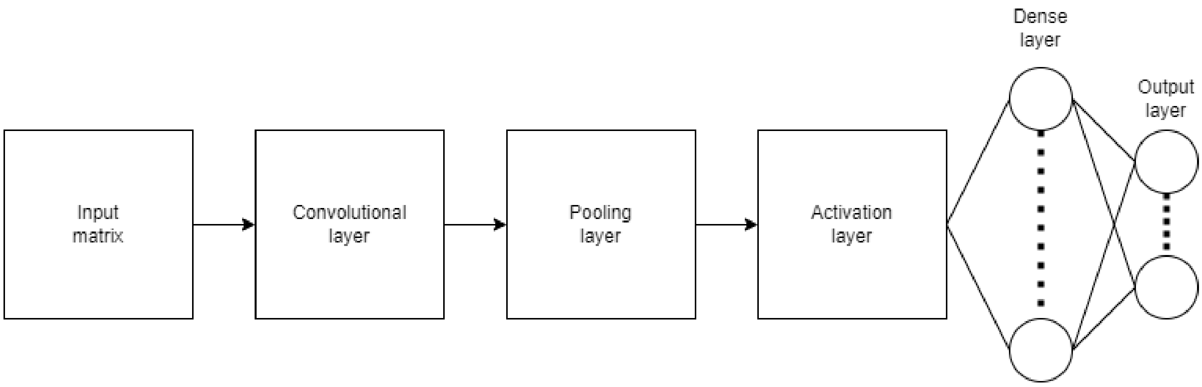

Figure 2.9: Convolutional neural network example.

Convolutional layer is the most important component of this network, which is responsible for detecting patterns. In contrast to dense layer, where an input is a vector of values, in convolutional layer input is an matrix of values. On this input matrix is performed **Convolution** operation and acquired matrix is passed to next layer. Convolutional layer will be further described in section 2.3.2.

Pooling layer is a type of layer which reduce size of input matrix. This layer is used to reduce time and memory consumption and to make sure that only „important" sections of the input matrix will be used for further computations. More information about this layer can be found in section 2.3.2.

Activation layer is a type of layer which apply activation function on the input matrix. There are a lot of possible choices for activation function, like sigmoid function (see section 2.3.1), but in scope of this thesis, ReLU (Rectified Linear Unit) activation function has been used. ReLU function can be described by equation $f(x) = max(0, x)$. To simplify, all negative numbers are changed to 0 and all positive numbers stays unchanged [4].

Basic structure of convolutional neural network is presented on fig. 2.9

Convolutional layer

As it was mentioned before, convolutional layer is capable of detecting patterns (features) such as edges on given input picture. This process can be done with use of filters (also known as kernels). Kernels are small matrices containing numbers which also are modifiable parameters used in learning process. Each convolution layer consists of some number of kernels from which each of them is used to detect some kind of pattern. For example, to recognize hand-written 1, two kernels can be used. First for detecting diagonal line and second one to detect straight line.

To check if given pattern exist in input data, each convolution layer perform **Cross-correlation** operation on input data and all kernels. Cross-correlation can be performed

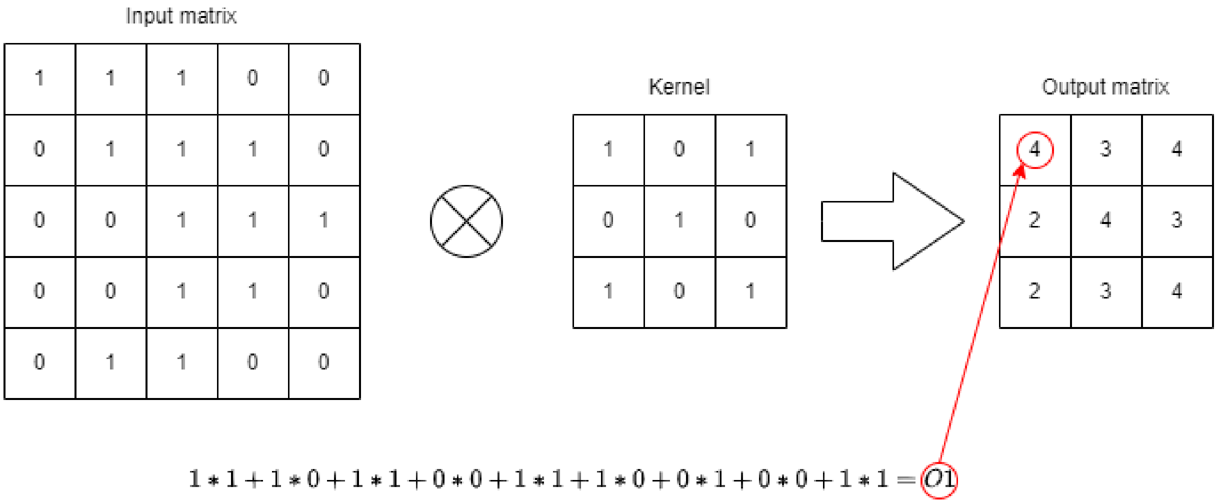

Figure 2.10: Cross-correlation example.

by „sliding" given kernel matrix over input data and computing **Frobenius inner product** (summing calculated element-wise multiplication products) [27] for all intersections. That calculated values creates output matrix of the cross-correlation operation. Mathematical equation for cross correlation can be written as follows:

$$G[i,j] = \sum_{u=0}^{l} \sum_{v=0}^{k} h[u,v]F[i+u.j+v], \tag{2.2}$$

$$G = h \bigotimes F, \tag{2.3}$$

where:

> $G[i,j]$ – output matrix value of indexes i and j, l – size of the input matrix, k – size of the kernel, h – kernel, F – input matrix [7].

Example of cross-correlation operation can be seen on fig. 2.10. Important thing to mention is the fact that cross-correlation operation can be performed with different **stride** (size of the step with which kernel is sliding over input matrix). The most common stride to use is 1 but it can be changed. In case of this thesis, all cross-correlation operations are performed with stride of 1. It is possible to calculate size of cross-correlation output matrix by using equation:

$$n_{out} = floor((l-k)/s) + 1, \tag{2.4}$$

where:

> n_{out} – size of output matrix, l – size of the input matrix, k – size of the kernel, s – stride.

In each convolutional layer first think that is performed on input data is a cross-correlation between input matrix and all the kernels that the layer consists of. Important

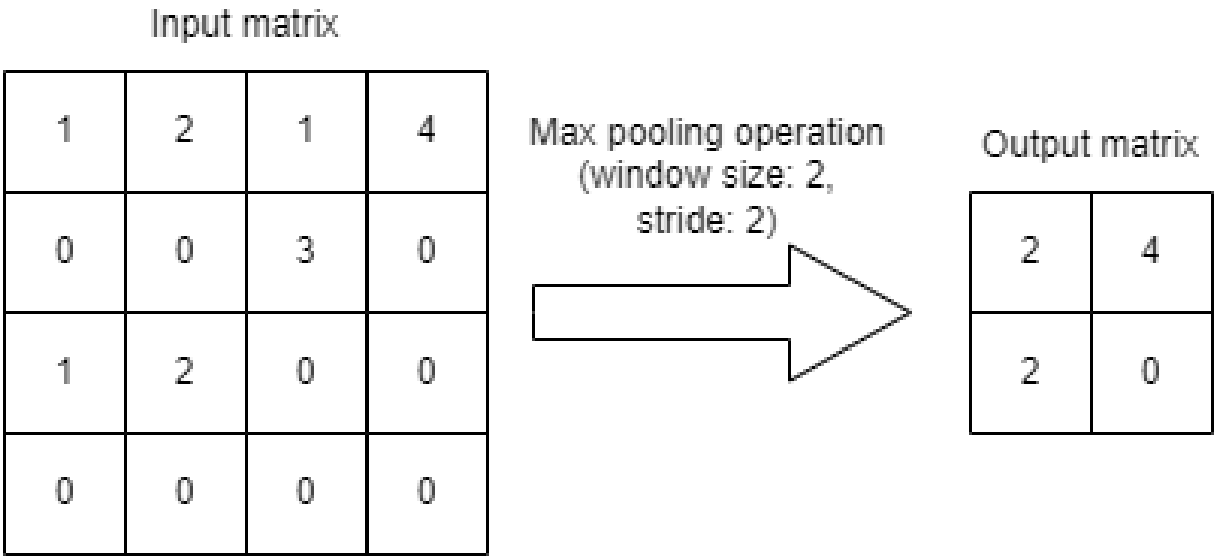

Figure 2.11: Max pooling example.

thing to mention is the fact that the output of the convolutional layer can be different. Number of matrices produced by layer is determine by number of kernels inside that layer. After performing cross-correlation operation, to each output matrix is added bias matrix [2]. In conclusion, output of convolutional layer with one kernel can be described by equation:

$$Conv_{out} = h \bigotimes F + B, \tag{2.5}$$

where:

> $Conv_{out}$ – convolutional layer output matrix, h – kernel, F – input matrix, B – bias matrix.

Pooling layer

Pooling layer is the second distinctive element of convolutional neural network. As it was mentioned before, main purpure of pooling layer is to reduce size of the matrix that is used for calculations. There are a lot of possible functions that can be performed in this layer but two the most common are **Max Pooling** and **Average Pooling** [15]. In case of this thesis, Max Pooling method has been used.

Max pooling method is also used to highlight the most important parts of the feature map. This operation works in the similar way to cross-correlation operation. Each pooling layer consists of window of the specific size which slide over input matrix, with specific stride (similar like in cross-correlation operation). While sliding over input matrix, algorithm pick maximal value and pass it into output matrix. Max pooling example can be seen on fig. 2.11

2.3.3 Learning process for neural network instance

After describing two types of neural network that will be used in following thesis, last thing that needs to be explained is learning process. This process looks similar in both neural networks (ANN adn CNN) so it will bee described as one. When neural network is created, values of all weights and biases are set randomly and that will result in network answers also being random [22]. To make network answers more sensible it needs to be „taught" how to approach the problem. Method of machine learning that will be used to improve neural network answers is called **supervised learning**. This type of learning method is based on examples and target output. Dataset that is used for training needs to have input data and target data to calculate how „bad" was model answer [20]. Learning process consists in periodic algorithm which will result in updating values of weights and biases. This process begins with splitting data set into two sets (training set and test set). Next, each example of training set need to be inputted into model and output needs to be read. When model output will be acquired it needs to be compared with target data for given example. One of the methods of performing this comparison is to calculate **loss function**. This is place is the first one in which learning process for ANN and CNN is slightly different. For both models loss function will be calculated, but formula for calculating this function will be different in each type of network. Loss function for artificial neural network will be calculated using formula:

$$\lambda_{ANN} = t - ANN_{out}, \tag{2.6}$$

where:

> λ_{ANN} – value of loss function for artificial neural network, t – target value for given example, ANN_{out} – output of the model.

Loss function for convolutional neural network will be calculated using formula:

$$\lambda_{CNN} = -\sum t \log CNN_{out}, \tag{2.7}$$

where:

> λ_{CNN} – value of loss function for convolutional neural network, t – target value for given example, CNN_{out} – output of the model.

By calculating loss functions for both models, it is possible to check how incompatible models answers are from desirable output.

Unfortunately, knowing how bad specific model performs in given task, won't result in making it better. To improve performance of given model, **Backpropagation algorithm** can be used [26, 31]. Backpropagation algorithm is the most important component in whole learning process because this methodology allows for improving how model perform. Main idea of this algorithm is backward propagation of computed error (loss function), which is based on calculating **gradient** value for given loss function (∇f). Result of this operation is the set of values which shows how values of output layer should change to decrease loss function. Unfortunately, there is no possibility to change neurons values directly. There is possibility to impact neuron value by modifying following components:

- values of input weights,
- value of biases,
- values of neurons in previous layer.

The same methodology can be applied to all layers in the network. Last important thing, worth mentioning, is the fact that learning process for both ANN and CNN looks the same but equations, used for calculating gradient are different for each network type.

In conclusion, learning process of neural network model can be described by 5 steps:

1. Load training data into model,
2. Using feedforward algorithm obtain output of the model,
3. Calculate loss function for given example,
4. Using backpropagation algorithm, calculate gradient values for all weights and biases,
5. Update values of all weights and biases.

After processing all examples from training set, test set is used to check model accuracy. If accuracy is to low, it is necessary to repeat training process. This sequence of action can be repeated until obtained accuracy will be acceptable (potentially, global minimum of the loss function will be achieved) [2, 26]. To improve, obtained results and to increase probability of finding global minimum of the loss function, there is 1 modification of backpropagation algorithm that will be used in scope of this thesis. There is parameter called **learning rate** [5]. This is value which define how big changes will be applied to weights and biases. Because final goal of the backpropagation algorithm is finding global minimum of the loss function, if the changes will be to big, it can result in constant passing the proper minimum value. Usage of proper learning rate value reduce probability of it happening.

2.4 Other approaches to the problem

After describing problem approach that will be used in scope of this thesis, it is important to mention what other solutions to the Chess AI problem can be found. First approach is very similar to described solution because it uses all mentioned components except of neural network instance. This solution assumes usage of complex mathematical equations to evaluate chessboard situation. This type of approach is caller /textbfheuristic approach [17, 18]. The main problem of heuristic approach is the fact that this solution is very time consuming and require big knowledge of chess game to implement correctly. Another disadvantage of this approach is the fact that it is very linear solution and it is hard to implement some unpredictability which can result in small effectivity against more advanced chess players.

Second possible approach looks much more promising but it is also much harder to implement. This approach assumes usage of self-improving neural network. Mentioned before AlphaZero software uses this exact method for playing [10]. Basic idea behind this approach is to create some kind of algorithm which will be building and modifying neural network structure while playing with real opponent. This method is very popular and very efficient in game theory because it can result in creating „unbeatable" AI. Main advantage of this solution is the fact that the more games will be played by AI instance, the smartest it gets [32]. Usage of this method can result in very interesting results but it is also very hard to implement. Another problem regarding this solution is time consumption and quality of „training resources". If created AI will play with the best players, it will learn a lot but in the other case it can stop improving.

Chapter 3

Subject of the thesis

Building Chess AI is very difficult and time consuming process if it is require to achieve good effectiveness. In scope of this thesis an approach has been used that includes the following components:

- game tree as an decision making structure,
- min-max algorithm as an search algorithm,
- game tree depth limit as an optimization method,
- artificial neural network as an first evacuation function,
- convolutional neural network as an second evaluation function.

3.1 Decision making system - implementation

Starting game tree structure is generated at the beginning of the program using recursive method. At the beginning of the program, there is option to choose game tree depth limit. This configuration specify how many moves needs to be included in game tree structure. When last layer of the structure will be achieved, game tree will be destroyed and regenerated with the same depth. As it was described in section 2.2.3, it is impossible to generate full game tree of chess, so that is why game tree depth limitation has been used. The most commonly used game tree depth limit is 3. It is possible to specify different value of this parameter but it is important to keep in mind that it will impact time needed to make decision. Example structure of game tree used in the the project can be seen on fig. 3.1.

Created game tree is used as an input to the min-max search algorithm. Detail explanation of this algorithm can be found in section 2.2.2. There are 2 main assumptions that needs to be explain while describing usage of min-max algorithm in scope of this thesis. If game take place in scenario „player vs AI", human player always is assign to

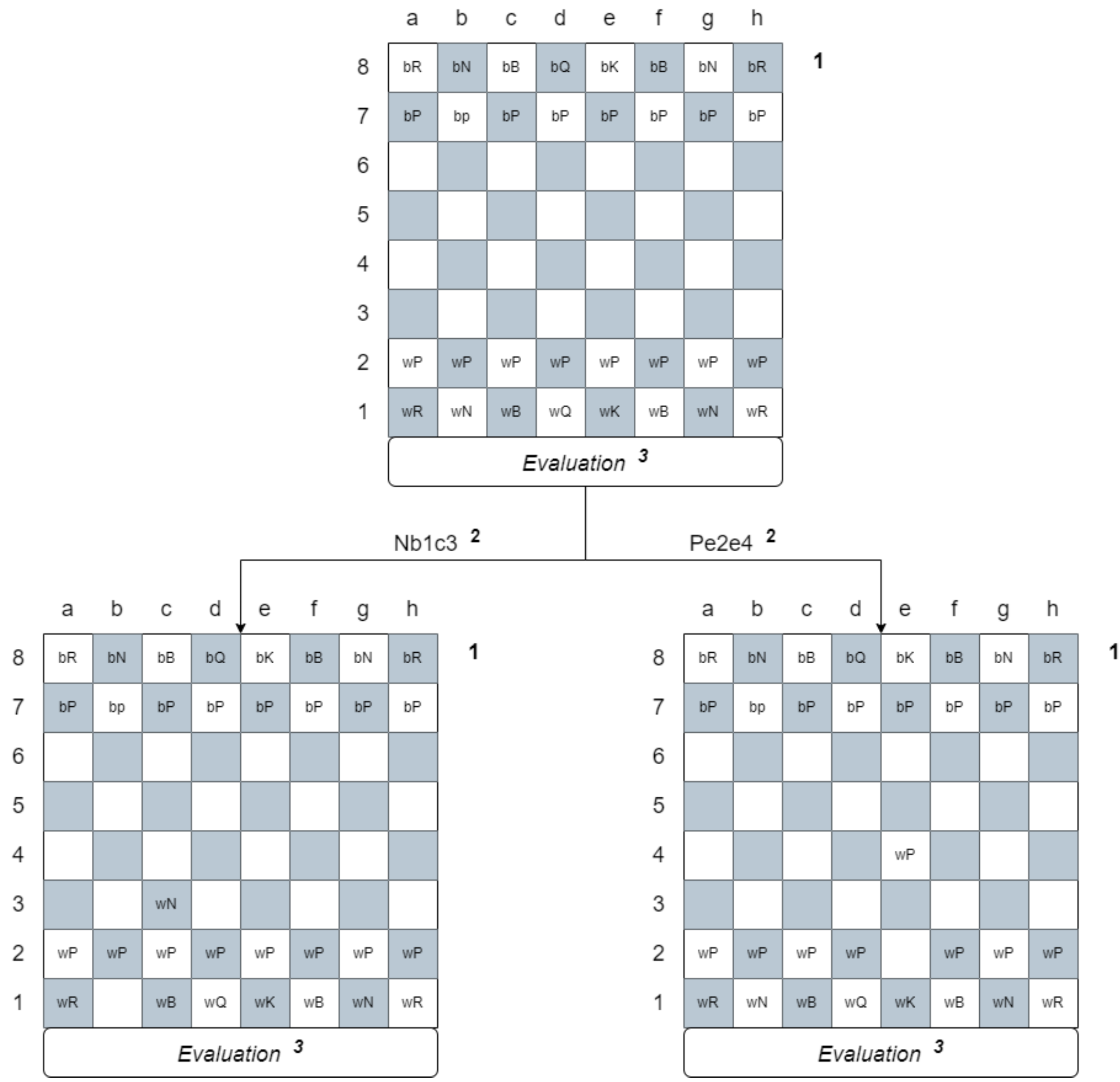

Figure 3.1: Game tree fragment example (1 - chess board representation, 2 - move command, 3 - evaluation value).

Table 3.1: The list of chess pieces types.

symbol	number value	string value
♔ ♚	7 / -7	wK / bK
♕ ♛	5 / -5	wQ / bQ
♗ ♝	4 / -4	wB / bB
♘ ♞	3 / -3	wN / bN
♖ ♜	2 / -2	wR / bR
♙ ♟	1 / -1	wP / bP

white site of the board which also result in making first move. If game scenario is set to be „AI vs AI", each of the instances gets assign randomly to one of the site. Sequence in which min-max algorithm work can be specified as follows:

1. Find the most beneficial sequence of moves in generated game tree.
2. Get chessboard situation after opponents move.
3. If gathered node is not one of game tree nodes, go back to point 1. Otherwise, regenerate game tree and go back to point 1.

Important thing to mentioned is the fact that in scope of this thesis there are no optimization method used for min-max algorithm. It can result in increasing time of making decisions while playing. It was decided to use min-max tree structure because it is core element of a lot of other chess playing AI and it it the mos beneficial solution.

3.2 Evaluation system - implementation

To make all experiments more accurate, it ha been decided to implement very basic models of ANN and CNN. To increase accuracy even more, both instances were coded from scratch and trained on the same datasets. Using neural network model as an evaluation method is not a new approach but usage of CNN and ANN, on the other hand is, because the most often used models are genetic algorithm based one. It is hard to predict how effective this approach would be, before analyzing tests results, but it can be very interesting how it perform.

Before further discursion it is necessary to describe how chessboard is represented in the final program. There are 3 methods that chess pieces are stored: number type, string type and object type. All of those types are shown in tab. 3.1 (object type will be skipped in the table). Data structure containing numerical values representing chess pieces will be used as an input in both instances of created neural networks.

3.2.1 Artificial neural network - architecture

As it was mentioned before, each of neural network instances was implemented from scratch to make experiments more reliable. When creating artificial neural network instance, there are two main aspects that needs to be specified. First and the most crucial topic is to design architecture of the network. Because problem that needs to be solved is evaluating chessboard situation, created network needs to take 64 values as an input (chessboard have dimension of 8×8) and needs to return one value which will be assigned as an evaluation value to the specific chessboard situation. To keep created model as simple as possible, it has been decided to include only two hidden layers with 64 neurons each. When structure of the network has been created, there is one last important configuration to establish. As it was mentioned in section 2.3.3, every instance of neural network needs to have learning rate parameter specified. During training process, it was established that learning rate value for artificial neural network should be equal to 0.001. This value resulted in the most efficient training process for this model. It is possible that ANN evaluation function will not provide spectacular results for given problem but it is important to remember that it has been used as an basic point of comparison for second evaluation. This method allows to decide if usage of CNN, for evaluating chessboard situations, is an justified or beneficial solution to the problem.

3.2.2 Convolutional neural network - architecture

While planing structure for convolutional neural network, the main goal was to keep structure of the model simple. This decision has been made because if basic structure of CNN model won't perform better than basic structure of ANN, it is a proof that given solution is bad. Second reason for keeping model structure simple is less time consuming training process for both models. There are two elements in convolutional neural network configuration that are similar to the first used model. The best value for learning rate parameter is equal to 0.001 and sizes of input and output layers are also the same like in artificial neural network. It has been decided that CNN model will consists of two convolutional layers, with 13 and 10 kernels respectively, two pooling layers, two activation layers and one fully connected layer. Full structure of the used CNN can be found on fig. 3.2. In the contrary to ANN hidden layers, which number of neurons are not backed by any theoretical aspects, convolutional layers was design based on chess game theory. According to publications about chess there are 13 the best openings which give player the most benefits. This is why first convolutiona layer consists of 13 kernels. Second convolutional layer consists of 10 kernels because this is the number of the most commonly used, beneficial chessboard arrangements [29, 12, 3]. The assumption is that both convolutional layers will be responsible for detecting mentioned chessboard arrangements.

Figure 3.2: Structure of the implemented CNN.

3.3 Used tools

In the scope this thesis, all components of the final project has been implemented from scratch using C++17 language. As an IDE Visual Studio Code (VSCode) has been chosen. To build and prepare project for compilation Premake software has been used. This thesis has been written using LaTeXlanguage and VSCode IDE. For creating all diagrams used in this thesis, Draw.io online software has been used. In the creative process of this project, the sources included in the bibliography and following documentations were used: [9, 8, 36, 25].

Chapter 4

Experiments

The first important thing that needs to be discussed before describing experiments is machine on which experiments have been performed. Experimental machines properties looks as follows:

- Operating system: Windows 10 Professional,
- Processor: Intel(R) Core i5-11400F
  - 11th generation,
  - 2.60GHz
- RAM: 16 GB DDR4,
- Graphical card: NVIDIA GeForce GTX 970,
- Hard drive: 500 GB SSD.

Those parameter can be changed but it is important to remember that used game tree structure is very memory consuming and neural network training require a lot of computation power. It is recommended to use specified parameters of higher. Last important thing to mention is the fact that result application has been created for Windows operating system only and wasn't tested on other operating systems.

4.1 Methodology

Performed tests was divided into three sets of tests:

Neural network training this set of tets was based on training created neural network instances. It allows for finding the most optimal values of learning rate parameter. After that, AI instances were tested for number of iterations (epochs) require to get the best accuracy and number of data require for training. More information about datasets used for training can be found in section 4.2.

Manual testing this set of tests relied on playing against AI instances in „player vs AI" scenario. This set of tests covered test case in which player play against trained and untrained AI instance. Important thing to mention is the fact that this set of tests required both neural networks to be trained on the same size of data set. Otherwise, experiment would be unreliable.

Automated testing this set of test was performed in „AI vs AI" scenario. The main goal of this testing was to perform small „tournament" to check which AI instance performs better. Similarly to te manual testing, this set of test has been performed with trained and untrained neural networks. Both neural networks was trained with the same data set.

4.1.1 Chess application

To perform mentioned tests, it was necessary to have application that allows for chess game. Application has been created and while starting it it is possible to specify which execution scenario needs to be performed. Because application uses command line interface, execution configuration is passed by input parameters. By specifying `-exScenario` parameter, application can be started in one of 3 modes:

- parameter value: `0` - application will start in mode „player vs player",
- parameter value: `1` - application will start in mode „player vs AI",
- parameter value: `2` - application will start in mode „AI vs AI".

Default value of this parameter is set on `0`. If `-exScenario` parameter will have value `1` or `2` application will ask for game tree limit to be specified (fig. 4.1). If user won't specify this parameter, its value will be set on 3.

Chess application - manual control

Choosing proper execution mode for the application, have impact on method of controlling application. If `-exScenario` parameter will have value `0` or `1`, application will show proper menu presented on fig. 4.2 Application main menu allows user to choose one of three options:

New game (command `N` / `n`) which reset all environment configurations and start new, fresh game. Choosing this option allows to set new game tree depth.

Move (command `M` / `m`) which allows user to perform move. Choosing this option allows to select option `?` which print out information about legal moves syntax.

Quit (command `Q` / `q`) which close application.

```
              .d8888b.  888
             d88P  Y88b 888
             888    888 888
             888        88888b.   .d88b.  .d8888b  .d8888b
             888        888 "88b d8P  Y8b 88K      88K
             888    888 888  888 88888888 "Y8888b. "Y8888b.
             Y88b  d88P 888  888 Y8b.          X88      X88
              "Y8888P"  888  888  "Y8888   88888P'  88888P'

Welcome in Chess AI program!

 How deep should the AI look for moves?
 Warning: values above 3 will be very slow. [n]? -->
```

Figure 4.1: Menu allowing for specify game tree depth.

Last thing worth mentioning is move command syntax. For move command to be accepted by the system, it is require to specify horizontal coordinate (a . . . h) first and vertical coordinate (1 . . . 8) second.

- **Syntax: <piece-id><starting-position><ending-position>**
- where:

  piece-id := {K – king, Q – queen, B – bishop, N – knight, R – rook, P – pawn}

  starting-position / ending-position := {a1. . . h8}

  description: Performs traditional move. Traditional capturing is also handled by this command.

  example: Nb8a6.

- **Syntax: 0-0**

  description: Performs short castling.

- **Syntax: 0-0-0**

  description: Performs long castling.

- **Syntax:** **`P<starting-position>-><piece-id>`**

  where:

  starting-position := {a1. . . h8}

  piece-id := {K – king, Q – queen, B – bishop, N – knight, R – rook, P – pawn}

```
                .d8888b.  888
               d88P  Y88b 888
               888    888 888
               888        88888b.   .d88b.  .d8888b  .d8888b
               888        888 "88b d8P  Y8b 88K      88K
               888    888 888  888 88888888 "Y8888b. "Y8888b.
               Y88b  d88P 888  888 Y8b.          X88      X88
                "Y8888P"  888  888  "Y8888   88888P'  88888P'

Captured pieces (WHITE):                        # List of captured white
                                                # colored pieces
Captured pieces (BLACK):                        # List of captured black
                                                # colored pieces
    --   --   --   --   --   --   --   --
8 | bR | bN | bB | bQ | bK | bB | bN | bR |     # Chessboard representation
    --   --   --   --   --   --   --   --
7 | bP | bP | bP | bP | bP | bP | bP | bP |
    --   --   --   --   --   --   --   --
6 |    |    |    |    |    |    |    |    |
    --   --   --   --   --   --   --   --
5 |    |    |    |    |    |    |    |    |
    --   --   --   --   --   --   --   --
4 |    |    |    |    |    |    |    |    |
    --   --   --   --   --   --   --   --
3 |    |    |    |    |    |    |    |    |
    --   --   --   --   --   --   --   --
2 | wP | wP | wP | wP | wP | wP | wP | wP |
    --   --   --   --   --   --   --   --
1 | wR | wN | wB | wQ | wK | wB | wN | wR |
    --   --   --   --   --   --   --   --
    a    b    c    d    e    f    g    h

Turn 1(WHITE)                                   # Turn counter
Commands: (N)ew game     (M)ove  (Q)uit         # Main menu

Insert command here --> M                       # Field for user input
Type '?' to see move command help. Insert command here -->
```

Figure 4.2: Application main page.

description: Promote pawn to chosen piece.

example: Pb7->Q.

- **Syntax:** `P<starting-position>x<first-coordinate-ending-position>`

  where:

  starting-position := {a1. . . h8}

  first-coordinate-ending-position := {a. . . h}

  describing: Performs en-passant manoeuver.

  example: Pc4xd.

4.1.2 Chess application - automated control

If `-exScenario` parameter will have value `2`, application will also show chessboard abd capture lists but control will be limited. After each turn, user will be able to continue game or stop it and exit application. This option was implemented to provide infinite games in which both AI instances play on the same level.

4.2 Data sets

To train created neural network instances, PGN files were used. Those are text file that contains records of chess games. PGN files consists of basic information about game, sequence of moves performed in this game and final outcome of the game. Example of the PGN file can be found on fig. 4.3. Presented example consists of field „Result". This field contains information about result of the game. To present information about winner, one of the following values can be used:

- `1/2-1/2` - draw,
- `0-1` - black site player win,
- `1-0` - white ste player win.

Unfortunately, PGN files in their plain format couldn't been used for training neural networks instances. As it was mentioned in section 3.2.1, as an input for neural networks, chessboard representation is used. To make training process possible, gathered PGN files needed to be processed and converted into chessboard situations. For converting PGN files, specially implemented class has been used. This class takes as an input PGN file and convert sequence of moves on respective chessboard situations. After converting sequence of moves, every chessboard situation needed to have evaluation value assigned to

```
[Event "FICS unrated blitz game"]   # Tournament name
[Site "FICS freechess.org"]             # Place of the tournament
[FICSGamesDBGameNo "410614205"]
[White "oldman"]                        # White site player
[Black "qoheleth"]                        # Black site player
[WhiteElo "2585"]                        # ELO rank of white site player
[BlackElo "1720"]                        # ELO rank of black site player
[Date "2017.01.19"]                        # Date of the game
[Time "18:37:00"]                        # Time of the game
[WhiteClock "0:05:00.000"]
[BlackClock "0:05:00.000"]
[ECO "C40"]                                # ECO code
[Result "1-0"]

1. e4 e5 2. Nf3 a5 3. Nxe5 h5 4. Bc4 b5 5. Bxf7+ Ke7 6. Qf3 Nf6 7. Qb3 Nc6
8. Ng6+ Kd6 9. Nc3 a4 10. Qxb5 {Black resigns} 1-0
```

Figure 4.3: PGN file example.

it. Evaluation value of each chessboard situation has been calculated using formula:

$$Ev = sign\left(1/t_c\right), \tag{4.1}$$

where:

> Ev – evaluation value, $sign$ – sign of the winning site (if black site „-”, if white site „+”), t_c – number of turns to end of the game.

In conclusion, every training example consists of chessboard situation and evaluation value. Last thing worth mentioning is the fact that prepared dataset has been divided into train and test set in proportions 70% and 30%, respectively.

4.3 Results

Like it can be seen in section 4.1, tests has been divided into three main parts. To facilitate reading, results description will also be divided into three sections.

4.3.1 Neural networks training

First test that has been performed in this test set was time consumption in relation to number of processed examples.Because number of epochs is irrelevant to this experiment, all tests runs has been executed with number of epochs equal to 1. In total, 5 test runs has been performed with data set split ratio of 0.7 and following data set sizes: 500, 1000, 2000, 5000 and 10000. Results of the experiment are shown on fig. 4.4. As it can be seen,

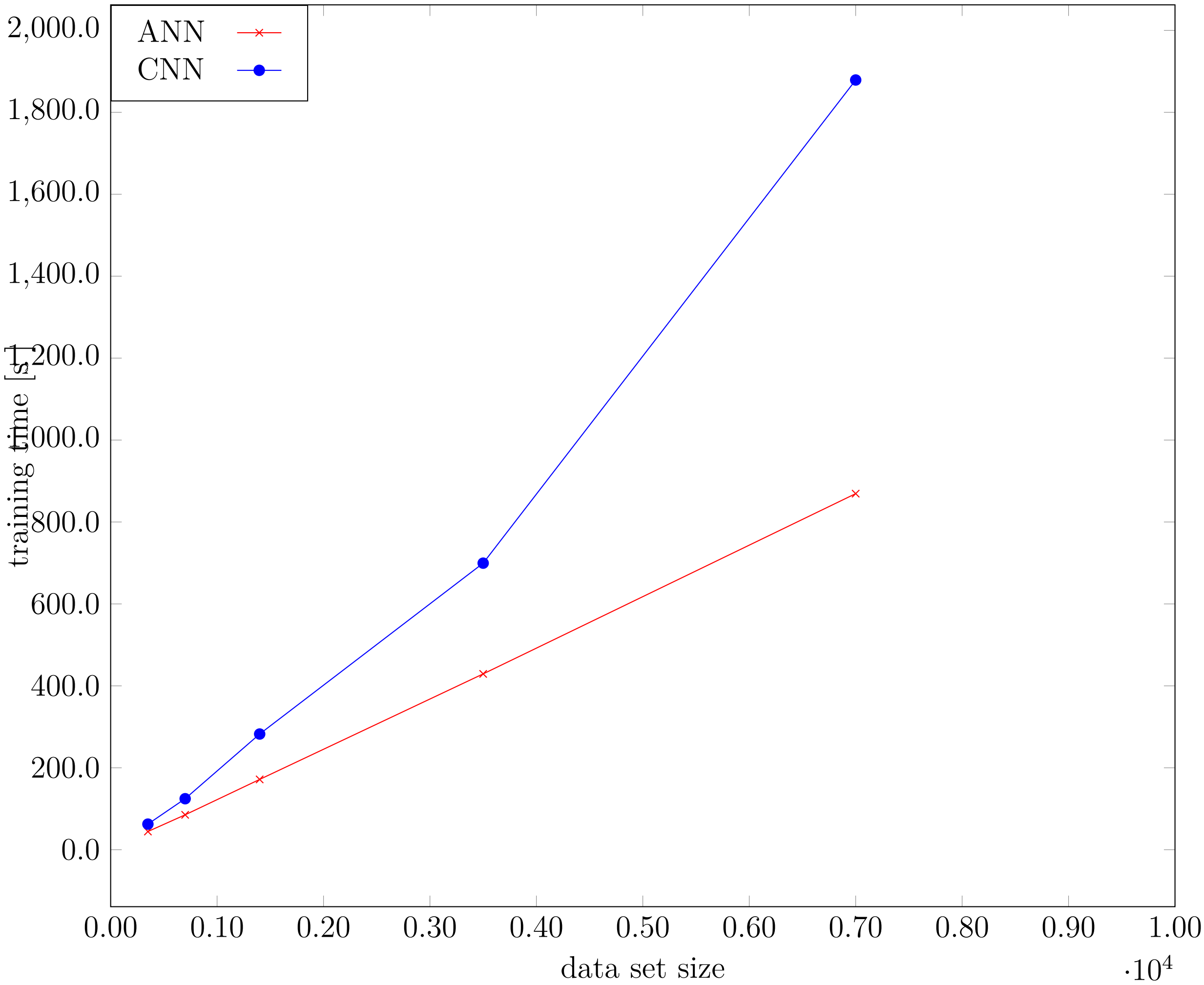

Figure 4.4: Training time consumption for neural network instances.

training process for convolutional neural network is more time consuming. This result is very much expected because bath feedforward and backpropagation algorithm, for this type of neural network, require performing more complected mathematical operations. The bigger training data set, time necessary for executing necessary algorithms increase drastically. Another interesting observation is the fact that for artificial neural network training time change in linear way. However in case of convolutional neural network, training time change in more logarithmic way. This experiment expose potential problematic characteristic of the CNN. It is known that size of training data sets for this type of neural networks needs to be much bigger than for artificial neural networks [16, 13]. This characteristic can result in enormous computing power requirement, if the model will be very complicated, which personal machines may not guarantee. For training very complex CNN models, it is require to use commercial, claster based, machines.

Next experiment which has been performed in this test set is finding optimal number of epochs and data set size to acquire acceptable accuracy in both models. Like it was mentioned in section 3.2, both instances of neural networks will be trained on the same data set. It has been decided to perform training for both instances with the same number of epochs. This can result in ANN being better trained than CNN but it was important for this thesis to prepare and configure both AI instances in the same way to make tests the most accurate and informative.

The experiment consisted of making training sessions and recording final accuracies. Training sessions were intermittent in one of two scenarios:

- if acceptable accuracy were acquire ($\sim 70\%$),
- if there were no significant changes in accuracy.

Results of the training sessions are presented in tab. 4.1 As it can be seen, only 1 epoch is not enough to train either of the neural network instances. Acquired 20% accuracy is not enough to resolve given problem. Second iteration, in which number of epochs has been increase to 50 looks much more promising. For artificial neural network it was possible to acquire acceptable accuracy but unfortunately it wasn't possible for convolutional neural network. results presented in last section of the tab. 4.1, present that the best accuracy for ANN was 82%. In this section it also was possible to achieve acceptable accuracy for CNN. Studying acquired results, it has been decided that the best configuration for the AI training will be 100 epochs and data set of size 1000. This configuration didn't provide the best accuracy but achieved results are acceptable and exerts an acceptable load of the local machine.

Table 4.1: The list of chess pieces types.

epochs	data set size	ANN		CNN	
		accuracy	acceptable	accuracy	acceptable
1	500	12%	NO	14%	NO
1	1000	14%	NO	10%	NO
1	2000	16%	NO	20%	NO
1	5000	14%	NO	16%	NO
1	10000	20%	NO	19%	NO
50	500	43%	NO	27%	NO
50	1000	39%	NO	34%	NO
50	2000	54%	NO	36%	NO
50	5000	62%	NO	39%	NO
50	10000	78%	YES	45%	NO
100	500	77%	YES	62%	NO
100	1000	80%	YES	73%	YES
100	2000	75%	YES	82%	YES
100	5000	82%	YES	80%	YES
100	10000	79%	YES	84%	YES

4.3.2 Manual testing of AI instances

First experiment that has been performed in this test set, was to check the impact of game tree depth on time needed to make decision. Gathered results has been presented on fig. 4.5. As it can be seen, both AI instances needs similar time to make decision. Slightly increase can be seen in case of instance using convolutional neural network but beginning three values are very similar. Testing has been performed up to game tree size of 6. It has been performed this way because used local machine had not enough RAM memory to generate bigger game tree structure. By analyzing given plot, it is possible to choose the most optimal game tree size. The goal is to pick the biggest size possible witch acceptable decision making time. Because first big „spike" in time variable can be seen in game tree size of 4, it will be the best option to choose game tree size of 3. Quick remark, results shown on fig. 4.5 explain perfectly, why the most often used game tree size is 3 [28, 18]. All of the following experiments will be performed with static game tree size of value 3.

As it was mentioned in section 4.1, this test set relies on playing against created AI instances in „player vs AI" mode. As an first experiment, untrained AI instances has been tested. Unfortunately, results were terrible. Both AI instances performed randomly and all moves that it performed were suboptimal. This behavior is absolutely correct because like it was mentioned in section 2.3.3, all values for weights and biases are initialized with random values. Because parameters values are random, the same property is applied to neural network decisions. All games played in this scenario has been won by the human player. Additionally, only two times it was possible to win using **scholar's mate**. This

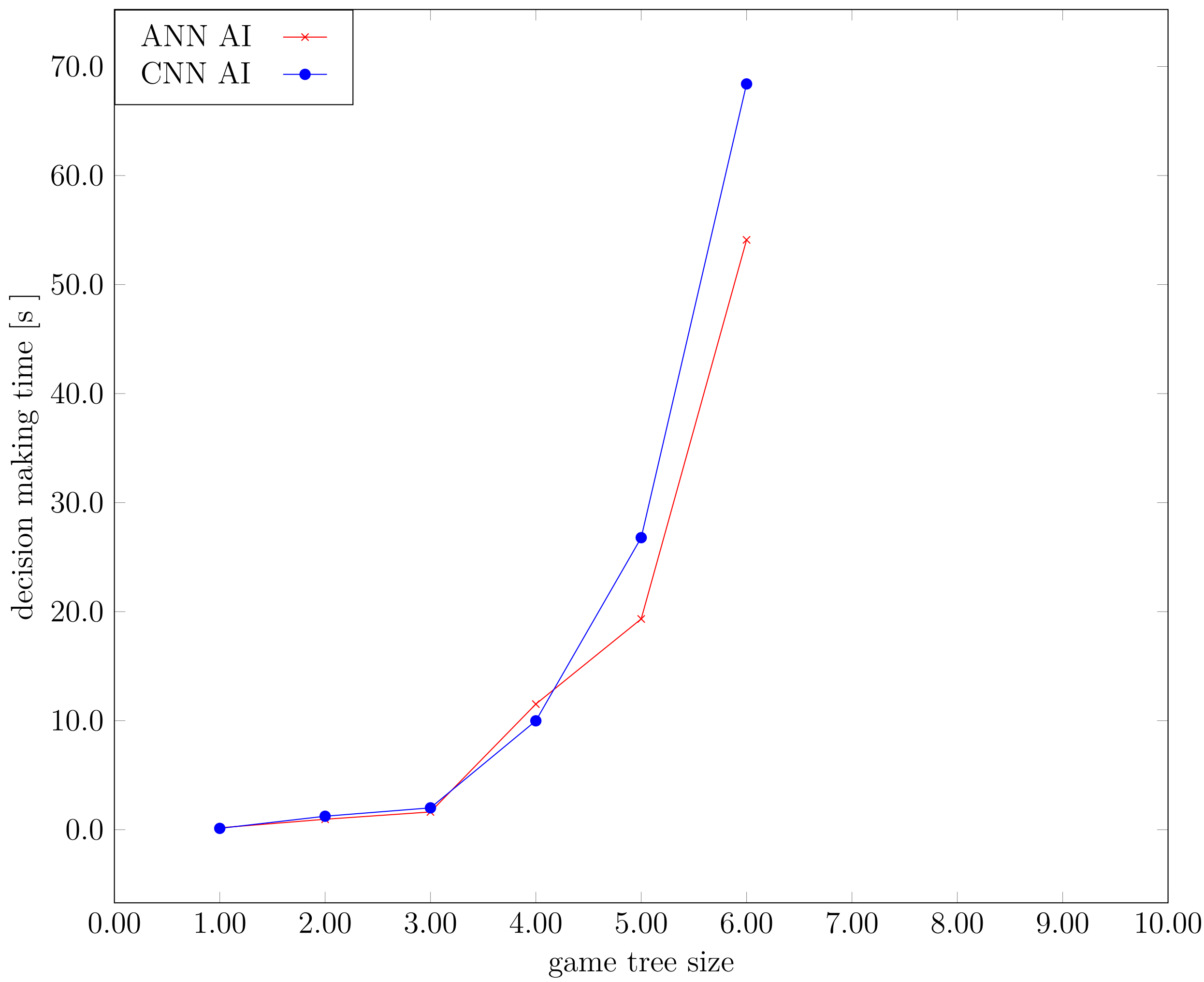

Figure 4.5: Time consumption in regards to game tree size.

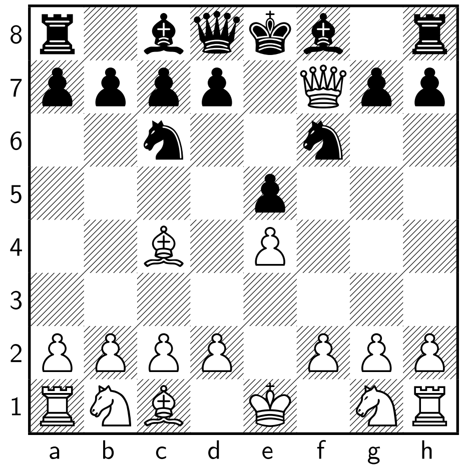

Figure 4.6: Scholar's mate example.

is special type of checkmate, which has been achieved with only 4 moves. Scholar's mate has been presented on fig. 4.6. Achieving this situation was hard because it require enemy to perform one specific moves, at the beginning of he game, which was almost impossible with randomly playing AI.

Experiment performed on trained AI instances provided much more interesting results. Before moving to results description, it is important to mention who performed role of the player in this experiment. All manual games has been played by author of this thesis. ELO ranking score of this player is unknown but it is player who have over 10 years of experience with chess. Considering this existence and gathered knowledge, tis player can be ranked as intermediate player. This is very good scenario for testing created AI instances. First parameter that will be used for specify play effectivity is **win ration**. Its value shows how many games has been won by AI. Win ration for both AI instances looks as follows:

- Win ratio for AI using ANN: $\sim 60\%$,
- Win ratio for AI using CNN: $\sim 72\%$.

Achieved win ratio, for both neural networks looks very promising. It is even more promising when putting into consideration fact that this scores has been achieved while playing with intermediate chess player. Interesting property that has been observed during games is the fact that AI, containing artificial neural network, starts to make optimal moves in around $4th$ turn. Such behavior may indicate that this AI instance needs to be train more with increase number of „opening" situation. This behavior can also be a good thing because it can put potential opponent in situation of feeling secure, which can result in making mistakes. In case of AI using convolutional neural network, also interesting behavior has been discovered. Player noticed that this AI instance was often aiming for performing scholar's mate. As it was mentioned in section 3.2.2, first convolutional layer should aim in recognizing good opening patterns. Mentioned behavior can indicate that this goal has been achieved.

4.3.3 Automated testing of AI instances

This test set has also been divided into two sections. Both of the experiments was based on testing created AI instances in „AI vs AI" mode. Unfortunately, first experiment that involved testing untrained AI instances, again didn't provided good results. All of played games has been interrupted after 50 turns due to lack of the progression. Both AI instances has been playing randomly as it was described in section 4.3.2.

Second experiment provided much more promising results. Before presenting gathered results it is important to specify rules of the performed experiment. Because „AI vs AI" mode provide user with limited control options (interrupt or continue game), following rules has been included:

- interrupt game if 5 sequential moves to not result in any progression,
- interrupt game at the end of $50th$ turn,
- interruption of the game will result in assuming that game result is draw.

Taking into consideration mentioned rules, 30 games has been performed. Results of those games are presented in tab. 4.2 (result meaning: „1-0" - AI using ANN wins, „0-1" - AI using CNN wins, „1-2/1-2" - draw). By analyzing gather results it is possible to calculate win ratio of both AI instances. Draws were treated as both instances wins. Calculated win ratio looks as follows:

- Win ratio for AI using ANN: $\sim 57\%$.
- Win ratio for AI using CNN: $\sim 67\%$.

Win ratios for both AI instances looks very promising and as it can be seen, both instances are on similar level. By analyzing played games, it was noticed that AI, that uses convolutional neural network, plays much better at the beginning of the game which in most cases resulted in wining whole game. In the section 4.3.2, it was mentioned that random plays at the beginning of the game, which artificial neural network performed, can be a good thing. Unfortunately, performed experiments shown that this tactic can work only against human opponent. This strategy can work better the lower the player experience level.

Table 4.2: Results of AI vs AI games.

game id	result	description
1	0-1	None
2	0-1	None
3	1-0	None
4	1-2/1-2	interrupted because of turn number limit
5	0-1	None
6	1-0	None
7	1-2/1-2	interrupted because of turn number limit
8	0-1	None
9	1-0	None
10	1-0	None
11	1-2/1-2	interrupted because lack of progression
12	0-1	None
13	0-1	None
14	0-1	None
15	1-0	None
16	1-2/1-2	interrupted because lack of progression
17	1-2/1-2	interrupted because lack of progression
18	1-0	None
19	1-0	None
20	0-1	None
21	1-0	None
22	1-2/1-2	interrupted because of turn number limit
23	0-1	None
24	1-0	None
25	0-1	None
26	0-1	None
27	0-1	None
28	1-2/1-2	interrupted because lack of progression
29	1-0	None
30	0-1	None

Chapter 5

Summary

- synthetic description of performed work
- conclusions
- future development, potential future research
- Has the objective been reached?

Bibliography

[1] Louis Victor Allis. *Searching for Solutions in Games and Artificial Intelligence.* Limburg: Ponsen & Looijen, 1994. ISBN: 97-890-9007-4-887.

[2] Alexander Amini. *Deep Computer Vision.* 2020. URL: `https://www.youtube.com/watch?v=iaSUYvmCekI&t=18s&ab_channel=AlexanderAmini` (visited on 28/01/2020).

[3] Stuard Margulies Bobby Fisher and Don Mosenfelder. *Bobby Fischer Teaches Chess.* New York: Bantam, 1982. ISBN: 97-805-5326-3-152.

[4] Jason Brownlee. *A Gentle Introduction to the Rectified Linear Unit (ReLU).* 2020. URL: `https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/` (visited on 09/01/2020).

[5] Jason Brownlee. *Understand the Impact of Learning Rate on Neural Network Performance.* 2019. URL: `https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/` (visited on 25/01/2019).

[6] Murray Campbell and T. Anthony Marsland. 'A comparison of minimax tree search algorithms'. In: *Artificial Intelligence* 20 (1983), pp. 347–367.

[7] Le Zhang Chen Wang and Lihua Xie. 'Kernel Cross-Correlator'. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 32.1 (2018).

[8] *Core Documentation.* 2022. URL: `https://www.latex-project.org/help/documentation/` (visited on 01/01/2022).

[9] *Cplusplus.com.* 2022. URL: `https://cplusplus.com` (visited on 01/01/2022).

[10] Julian Schrittwieser David Silver Thomas Hubert and Demis Hassabis. *AlphaZero: Shedding new light on chess, shogi, and Go.* 2018. URL: `https://www.deepmind.com/blog/alphazero-shedding-new-light-on-chess-shogi-and-go` (visited on 06/12/2018).

[11] Sacha Droste and Johannes Fürnkranz. 'Learning of Piece Values for Chess Variants'. In: *Droste 2008 LearningOP.* 2008.

[12] James Eade and Al Lawrence. *Chess Player's Bible*. London: Apple Press, 2015. ISBN: 97-818-4543-6-018.

[13] David Foster. *Generative Deep Learning: Teaching Machines to Paint, Write, Compose, and Play*. Farnham: O'Reilly UK Ltd., 2019. ISBN: 97-814-9204-1-948.

[14] Martin Gardner. 'Mathematical Games'. In: *Scientific American* 312 (1962), pp. 138–144.

[15] Hossein Gholamalinejad and Hossein Khosravi. *Pooling Methods in Deep Neural Networks, a Review*. Shahrood: Publisher, 2020.

[16] Adam Gibson and Josh Patterson. *Deep Learning: A Practitioner's Approach*. Sebastopol: O'Reilly Media, 2017. ISBN: 97-814-9191-4-250.

[17] Kieran Greer. 'Computer chess move-ordering schemes using move influence'. In: *Artificial Intelligence* 120.2 (2000), pp. 235–250.

[18] Lauri Hartikka. *A step-by-step guide to building a simple chess AI*. 2017. URL: `https://www.freecodecamp.org/news/simple-chess-ai-step-by-step-1d55a9266977/` (visited on 30/03/2017).

[19] Duca Iliescu and Delia Monica. 'The Impact of Artificial Intelligence on the Chess World'. In: *JMIR serious games* 8.4 (2020).

[20] Sanjeev Kulkarni and Gilbert Harman. *An Elementary Introduction to Statistical Learning Theory*. New Jersey: Wiley, 2011. ISBN: 97-811-1802-3-464.

[21] Ajitesh Kumar. *Real-World Applications of Convolutional Neural Networks*. 2021. URL: `https://vitalflux.com/real-world-applications-of-convolutional-neural-networks/` (visited on 06/11/2021).

[22] Frank La. 'How Do Neural Networks Learn?' In: *MSDN* 34.4 (2019).

[23] dr. Mark Humphrys. *Sigmoid activation function*. 2000. URL: `https://humphryscomputing.com/Notes/Neural/sigmoid.html` (visited on 22/07/2000).

[24] Michael Nielsen. *Neural Networks and Deep Learning*. 2019. URL: `http://neuralnetworksanddeep`
`com/` (visited on 01/12/2019).

[25] *Premake Documentation*. 2022. URL: `https://premake.github.io/docs/` (visited on 01/01/2022).

[26] Tariq Rashid. *Make Your Own Neural Network*. Scotts Valley: CreateSpace Independent Publishing Platform, 2016. ISBN: 97-815-3082-6-605.

[27] PD Robinson and AJ Wathen. *The Frobenius inner product, and variational bounds on the traces of inverse matrices*. Bristol: Univ of Bristol, 1996.

[28] Yehoshua Rubin. 'A New Paradigm: Chess AI In Game Analysis'. In: *Google* (2022).

[29] Joshua Sheng and Guannan Song. *Mastering Chess Logic*. London: Everyman Chess, 2021. ISBN: 97-817-8194-6-237.

[30] Nick Polson Shiva Maharaj and Alex Turk. 'Chess AI: Competing Paradigms for Machine Intelligence'. In: *CoRR* 2109 (2021).

[31] Pavithra Solai. *Convolutions and Backpropagations*. 2018. URL: `https://pavisj.medium.com/convolutions-and-backpropagations-46026a8f5d2c` (visited on 19/03/2018).

[32] Kenneth O. Stanley and Risto Miikkulainen. 'Evolving Neural Networks through Augmenting Topologies'. In: *Evolutionary Computation* 10.2 (2016), pp. 99–127.

[33] Pejman Tahmasebi and Ardeshir Hezarkhani. 'Application of a Modular Feedforward Neural Network for Grade Estimation'. In: *Natural Resources Research* 20 (2011), pp. 25–32.

[34] Michael Tarsi. 'Optimal Search on Some Game Trees'. In: *Journal of the ACM* 30.3 (1938), pp. 389–396.

[35] Chess.com Team. *How to Play Chess: 7 Steps To Get You Started*. 2021. URL: `https://www.chess.com/learn-how-to-play-chess` (visited on 18/08/2021).

[36] *Visual Studio Code Documentation*. 2022. URL: `https://code.visualstudio.com/docs` (visited on 01/01/2022).

Appendices

Technical documentation

List of abbreviations and symbols

AI (Artificial Intelligence) is a type of computer software which is capable of learning how to resolve problems.

Search algorithm is a type of algorithm which is use for searching process in data structures. Examples of search algorithms are: min-max, max, min etc.

Game tree leaf is a node in the last layer of the structure.

Feature map is a matrix that is an output of convolutional layer.

IDE (Integrated Development Environment) is a software application that provides comprehensive facilities to computer programmers for software development.

PGN (Portable Game Notation) is a standard text based notation to records chess games (those files includes not only moves but additional data related to this game).

ELO is an rating system which allows for calculating the relative skill levels of players in games such as chess. This rating system is also often use in video games.

Epoch is a name of the single model training iteration. This training iteration consists of **train** (process all training data and update weights values) and **test** (process test set and calculate accuracy of the model).

List of additional files in electronic submission (if applicable)

Additional files uploaded to the system include:

- source code of the application,
- test data,
- a video file showing how software or hardware developed for thesis is used,
- etc.

List of Figures

List of Tables